



Network Functions Virtualisation (NFV) Release 5; Protocols and Data Models; NFV descriptors based on TOSCA specification

Disclaimer

The present document has been produced and approved by the Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference

RGS/NFV-SOL001ed531

Keywords

data, information model, model, NFV

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Important notice

The present document can be downloaded from the
[ETSI Search & Browse Standards](#) application.

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format on [ETSI deliver](#) repository.

Users should be aware that the present document may be revised or have its status changed,
this information is available in the [Milestones listing](#).

If you find errors in the present document, please send your comments to
the relevant service listed under [Committee Support Staff](#).

If you find a security vulnerability in the present document, please report it through our
[Coordinated Vulnerability Disclosure \(CVD\)](#) program.

Notice of disclaimer & limitation of liability

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.

No recommendation as to products and services or vendors is made or should be implied.

No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.

In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

Copyright Notification

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.

The content of the PDF version shall not be modified without the written authorization of ETSI.

The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2025.
All rights reserved.

Contents

Intellectual Property Rights	29
Foreword.....	29
Modal verbs terminology.....	29
1 Scope	30
2 References	30
2.1 Normative references	30
2.2 Informative references.....	31
3 Definition of terms, symbols and abbreviations.....	32
3.1 Terms.....	32
3.2 Symbols.....	32
3.3 Abbreviations	33
4 Overview of TOSCA model.....	33
5 General concept of using TOSCA to model NFV descriptors	33
5.1 Introduction	33
5.2 Network Service Descriptor	33
5.3 Virtualised Network Function Descriptor	34
5.4 Physical Network Function Descriptor.....	34
5.5 toasca_definitions_version and Namespace prefix	34
5.6 Imports statement	35
5.6.1 VNFD TOSCA service template	35
5.6.2 NSD TOSCA service template	35
5.6.3 PNFD TOSCA service template	36
5.7 Type extension	37
5.7.1 Introduction.....	37
5.7.2 Rules	37
5.7.3 VNFD Types.....	37
5.7.4 NSD types.....	38
5.7.5 Security-sensitive properties in extended data types	39
5.8 Non-Backward Compatible changes	39
5.9 Use of TOSCA functions	39
6 VNFD TOSCA model.....	40
6.1 Introduction	40
6.2 Data Types.....	43
6.2.1 toasca.datatypes.nfv.CpProtocolData	43
6.2.1.1 Description	43
6.2.2 toasca.datatypes.nfv.AddressData	43
6.2.2.1 Description	43
6.2.3 toasca.datatypes.nfv.L2AddressData	43
6.2.3.1 Description	43
6.2.4 toasca.datatypes.nfv.VirtualNetworkInterfaceRequirements	43
6.2.4.1 Description	43
6.2.4.2 Properties	43
6.2.4.3 Definition	44
6.2.4.4 Examples.....	44
6.2.4.5 Additional Requirements	44
6.2.5 toasca.datatypes.nfv.L3AddressData	44
6.2.5.1 Description	44
6.2.6 toasca.datatypes.nfv.RequestedAdditionalCapability.....	45
6.2.6.1 Description	45
6.2.6.2 Properties	45
6.2.6.3 Definition	46
6.2.6.4 Examples.....	46
6.2.6.5 Additional Requirements	46

6.2.7	tosca.datatypes.nfv.VirtualMemory	46
6.2.7.1	Description	46
6.2.7.2	Properties	47
6.2.7.3	Definitions	47
6.2.7.4	Examples	48
6.2.7.5	Additional Requirements	48
6.2.8	tosca.datatypes.nfv.VirtualCpu	48
6.2.8.1	Description	48
6.2.8.2	Properties	49
6.2.8.3	Definition	49
6.2.8.4	Examples	50
6.2.8.5	Additional Requirements	50
6.2.9	tosca.datatypes.nfv.VirtualCpuPinning	50
6.2.9.1	Description	50
6.2.9.2	Properties	51
6.2.9.3	Definition	51
6.2.9.4	Examples	51
6.2.9.5	Additional Requirements	52
6.2.10	tosca.datatypes.nfv.VnfcConfigurableProperties	52
6.2.10.1	Description	52
6.2.10.2	Properties	52
6.2.10.3	Definition	52
6.2.10.4	Examples	53
6.2.10.5	Additional Requirements	54
6.2.11	tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties	55
6.2.11.1	Description	55
6.2.11.2	Properties	55
6.2.11.3	Definition	55
6.2.11.4	Examples	55
6.2.12	tosca.datatypes.nfv.VduProfile	55
6.2.12.1	Description	55
6.2.12.2	Properties	55
6.2.12.3	Definition	56
6.2.12.4	Examples	58
6.2.12.5	Additional requirements	59
6.2.13	tosca.datatypes.nfv.VlProfile	59
6.2.13.1	Description	59
6.2.13.2	Properties	59
6.2.13.3	Definition	60
6.2.13.4	Examples	60
6.2.13.5	Additional Requirements	61
6.2.14	tosca.datatypes.nfv.VirtualLinkProtocolData	61
6.2.14.1	Description	61
6.2.14.2	Properties	61
6.2.14.3	Definition	61
6.2.14.4	Examples	62
6.2.14.5	Additional Requirements	62
6.2.15	tosca.datatypes.nfv.L2ProtocolData	62
6.2.15.1	Description	62
6.2.15.2	Properties	62
6.2.15.3	Definition	63
6.2.15.4	Examples	64
6.2.15.5	Additional Requirements	64
6.2.16	tosca.datatypes.nfv.L3ProtocolData	64
6.2.16.1	Description	64
6.2.16.2	Properties	64
6.2.16.3	Definition	65
6.2.16.4	Examples	65
6.2.16.5	Additional Requirements	66
6.2.17	tosca.datatypes.nfv.IpAllocationPool	66
6.2.17.1	Description	66
6.2.17.2	Properties	66

6.2.17.3	Definition	66
6.2.17.4	Examples	66
6.2.17.5	Additional Requirements	66
6.2.18	tosca.datatypes.nfv.InstantiationLevel	67
6.2.18.1	Description	67
6.2.18.2	Properties	67
6.2.18.3	Definition	67
6.2.18.4	Examples	67
6.2.18.5	Additional Requirements	67
6.2.19	tosca.datatypes.nfv.VduLevel	68
6.2.19.1	Description	68
6.2.19.2	Properties	68
6.2.19.3	Definition	68
6.2.19.4	Examples	68
6.2.19.5	Additional Requirements	68
6.2.20	tosca.datatypes.nfv.VnfLcmOperationsConfiguration	69
6.2.20.1	Description	69
6.2.20.2	Properties	69
6.2.20.3	Definition	70
6.2.20.4	Examples	71
6.2.20.5	Additional Requirements	71
6.2.21	tosca.datatypes.nfv.VnfInstantiateOperationConfiguration	71
6.2.21.1	Description	71
6.2.21.2	Properties	71
6.2.21.3	Definition	72
6.2.21.4	Examples	72
6.2.21.5	Additional Requirements	72
6.2.22	tosca.datatypes.nfv.VnfScaleOperationConfiguration	72
6.2.22.1	Description	72
6.2.22.2	Properties	72
6.2.22.3	Definition	73
6.2.22.4	Examples	73
6.2.22.5	Additional Requirements	73
6.2.23	tosca.datatypes.nfv.VnfScaleToLevelOperationConfiguration	73
6.2.23.1	Description	73
6.2.23.2	Properties	73
6.2.23.3	Definition	73
6.2.23.4	Examples	74
6.2.23.5	Additional Requirements	74
6.2.24	tosca.datatypes.nfv.VnfHealOperationConfiguration	74
6.2.24.1	Description	74
6.2.24.2	Properties	74
6.2.24.3	Definition	74
6.2.24.4	Examples	75
6.2.24.5	Additional Requirements	75
6.2.25	tosca.datatypes.nfv.VnfTerminateOperationConfiguration	75
6.2.25.1	Description	75
6.2.25.2	Properties	75
6.2.25.3	Definition	75
6.2.25.4	Examples	76
6.2.25.5	Additional Requirements	76
6.2.26	tosca.datatypes.nfv.VnfOperateOperationConfiguration	76
6.2.26.1	Description	76
6.2.26.2	Properties	76
6.2.26.3	Definition	76
6.2.26.4	Examples	77
6.2.26.5	Additional Requirements	77
6.2.27	tosca.datatypes.nfv.ScaleInfo	77
6.2.27.1	Description	77
6.2.28	tosca.datatypes.nfv.ScalingAspect	77
6.2.28.1	Description	77
6.2.28.2	Properties	77

6.2.28.3	Definition	78
6.2.28.4	Examples	78
6.2.28.5	Additional Requirements	79
6.2.29	tosca.datatypes.nfv.LinkBitrateRequirements	79
6.2.29.1	Description	79
6.2.30	tosca.datatypes.nfv.ConnectivityType	79
6.2.30.1	Description	79
6.2.31	tosca.datatypes.nfv.VnfConfigurableProperties	79
6.2.31.1	Description	79
6.2.31.2	Properties	79
6.2.31.3	Definition	81
6.2.31.4	Examples	82
6.2.31.5	Additional Requirements	83
6.2.32	tosca.datatypes.nfv.VnfAdditionalConfigurableProperties	83
6.2.32.1	Description	83
6.2.32.2	Properties	83
6.2.32.3	Definition	83
6.2.32.4	Examples	84
6.2.32.5	Additional Requirements	84
6.2.33	tosca.datatypes.nfv.VnfInfoModifiableAttributes	84
6.2.33.1	Description	84
6.2.33.2	Properties	84
6.2.33.3	Definition	84
6.2.33.4	Examples	85
6.2.33.5	Additional Requirements	85
6.2.34	tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions	85
6.2.34.1	Description	85
6.2.34.2	Properties	85
6.2.34.3	Definition	85
6.2.34.4	Examples	86
6.2.34.5	Additional Requirements	86
6.2.35	tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata	86
6.2.35.1	Description	86
6.2.35.2	Properties	86
6.2.35.3	Definition	86
6.2.35.4	Examples	86
6.2.35.5	Additional Requirements	87
6.2.36	tosca.datatypes.nfv.Qos	87
6.2.36.1	Description	87
6.2.37	tosca.datatypes.nfv.LogicalNodeData	87
6.2.37.1	Description	87
6.2.37.2	Properties	87
6.2.37.3	Definition	88
6.2.37.4	Examples	88
6.2.37.5	Additional Requirements	88
6.2.38	Void	88
6.2.39	tosca.datatypes.nfv.VirtualBlockStorageData	88
6.2.39.1	Description	88
6.2.39.2	Properties	88
6.2.39.3	Definition	89
6.2.39.4	Examples	90
6.2.39.5	Additional Requirements	90
6.2.40	tosca.datatypes.nfv.VirtualObjectStorageData	90
6.2.40.1	Description	90
6.2.40.2	Properties	90
6.2.40.3	Definition	91
6.2.40.4	Examples	91
6.2.40.5	Additional Requirements	91
6.2.41	tosca.datatypes.nfv.VirtualFileStorageData	91
6.2.41.1	Description	91
6.2.41.2	Properties	91
6.2.41.3	Definition	92

6.2.41.4	Examples	92
6.2.41.5	Additional Requirements	93
6.2.42	tosca.datatypes.nfv.VirtualLinkBitrateLevel	93
6.2.42.1	Description	93
6.2.42.2	Properties	93
6.2.42.3	Definition	93
6.2.42.4	Examples	93
6.2.42.5	Additional Requirements	93
6.2.43	tosca.datatypes.nfv.VnfOperationAdditionalParameters	94
6.2.43.1	Description	94
6.2.43.2	Properties	94
6.2.43.3	Definition	94
6.2.43.4	Examples	94
6.2.43.5	Additional Requirements	95
6.2.44	tosca.datatypes.nfv.VnfChangeFlavourOperationConfiguration	95
6.2.44.1	Description	95
6.2.44.2	Properties	95
6.2.44.3	Definition	96
6.2.44.4	Examples	96
6.2.44.5	Additional Requirements	96
6.2.45	tosca.datatypes.nfv.VnfChangeExtConnectivityOperation Configuration	96
6.2.45.1	Description	96
6.2.45.2	Properties	96
6.2.45.3	Definition	96
6.2.45.4	Examples	97
6.2.45.5	Additional Requirements	97
6.2.46	tosca.datatypes.nfv.VnfMonitoringParameter	97
6.2.47	tosca.datatypes.nfv.VnfcMonitoringParameter	97
6.2.47.1	Description	97
6.2.47.2	Properties	97
6.2.47.3	Definition	98
6.2.47.4	Examples	99
6.2.47.5	Additional Requirements	99
6.2.48	tosca.datatypes.nfv.VirtualLinkMonitoringParameter	99
6.2.48.1	Description	99
6.2.48.2	Properties	99
6.2.48.3	Definition	100
6.2.48.4	Examples	100
6.2.48.5	Additional Requirements	100
6.2.49	tosca.datatypes.nfv.InterfaceDetails	100
6.2.49.1	Description	100
6.2.49.2	Properties	100
6.2.49.3	Definition	101
6.2.49.4	Examples	101
6.2.49.5	Additional Requirements	101
6.2.50	tosca.datatypes.nfv.UriComponents	101
6.2.50.1	Description	101
6.2.50.2	Properties	101
6.2.50.3	Definition	102
6.2.50.4	Examples	102
6.2.50.5	Additional Requirements	102
6.2.51	tosca.datatypes.nfv.UriAuthority	103
6.2.51.1	Description	103
6.2.51.2	Properties	103
6.2.51.3	Definition	103
6.2.51.4	Examples	104
6.2.51.5	Additional Requirements	104
6.2.52	tosca.datatypes.nfv.VnfProfile	104
6.2.52.1	Description	104
6.2.53	tosca.datatypes.nfv.ChecksumData	104
6.2.53.1	Description	104
6.2.53.2	Properties	104

6.2.53.3	Definition	105
6.2.53.4	Examples	105
6.2.53.5	Additional Requirements	105
6.2.54	tosca.datatypes.nfv.VnfmInterfaceInfo	105
6.2.54.1	Description	105
6.2.54.2	Properties	105
6.2.54.3	Definition	106
6.2.54.4	Examples	106
6.2.54.5	Additional Requirements	106
6.2.55	tosca.datatypes.nfv.OauthServerInfo	106
6.2.55.1	Description	106
6.2.55.2	Properties	107
6.2.55.3	Definition	107
6.2.55.4	Examples	107
6.2.55.5	Additional Requirements	107
6.2.56	tosca.datatypes.nfv.BootData	107
6.2.56.1	Description	107
6.2.56.2	Properties	107
6.2.56.3	Definition	108
6.2.56.4	Examples	108
6.2.56.5	Additional Requirements	108
6.2.57	tosca.datatypes.nfv.KvpData	109
6.2.57.1	Description	109
6.2.57.2	Properties	109
6.2.57.3	Definition	109
6.2.57.4	Examples	109
6.2.57.5	Additional Requirements	109
6.2.58	tosca.datatypes.nfv.ContentOrFileData	110
6.2.58.1	Description	110
6.2.58.2	Properties	110
6.2.58.3	Definition	110
6.2.58.4	Examples	111
6.2.58.5	Additional Requirements	111
6.2.59	tosca.datatypes.nfv.BootDataVimSpecificProperties	111
6.2.59.1	Description	111
6.2.59.2	Properties	111
6.2.59.3	Definition	112
6.2.59.4	Examples	112
6.2.59.5	Additional Requirements	112
6.2.60	tosca.datatypes.nfv.VnfPackageChangeSelector	112
6.2.60.1	Description	112
6.2.60.2	Properties	113
6.2.60.3	Definition	113
6.2.60.4	Examples	113
6.2.60.5	Additional Requirements	113
6.2.61	tosca.datatypes.nfv.VnfPackageChangeComponentMapping	114
6.2.61.1	Description	114
6.2.61.2	Properties	114
6.2.61.3	Definition	114
6.2.61.4	Examples	115
6.2.61.5	Additional Requirements	115
6.2.62	tosca.datatypes.nfv.VnfChangeCurrentPackageOperation Configuration	115
6.2.62.1	Description	115
6.2.62.2	Properties	115
6.2.62.3	Definition	115
6.2.62.4	Examples	116
6.2.62.5	Additional Requirements	116
6.2.63	tosca.datatypes.nfv.VnfCreateSnapshotOperationConfiguration	116
6.2.63.1	Description	116
6.2.63.2	Properties	116
6.2.63.3	Definition	116
6.2.63.4	Examples	116

6.2.63.5	Additional Requirements	116
6.2.64	tosca.datatypes.nfv.VnfRevertToSnapshotOperationConfiguration	116
6.2.64.1	Description	116
6.2.64.2	Properties	117
6.2.64.3	Definition	117
6.2.64.4	Examples	117
6.2.64.5	Additional Requirements	117
6.2.65	tosca.datatypes.nfv.ServicePortData	117
6.2.65.1	Description	117
6.2.65.2	Properties	117
6.2.65.3	Definition	118
6.2.65.4	Examples	118
6.2.65.5	Additional Requirements	118
6.2.66	tosca.datatypes.nfv.AdditionalServiceData	119
6.2.66.1	Description	119
6.2.66.2	Properties	119
6.2.66.3	Definition	119
6.2.66.4	Examples	119
6.2.66.5	Additional Requirements	120
6.2.67	tosca.datatypes.nfv.VnfLcmOpCoord	120
6.2.67.1	Description	120
6.2.67.2	Properties	120
6.2.67.3	Definition	122
6.2.67.4	Examples	122
6.2.67.5	Additional Requirements	122
6.2.68	tosca.datatypes.nfv.InputOpCoordParams	123
6.2.68.1	Description	123
6.2.68.2	Properties	123
6.2.68.3	Definition	123
6.2.69	tosca.datatypes.nfv.OutputOpCoordParams	123
6.2.69.1	Description	123
6.2.69.2	Properties	123
6.2.69.3	Definition	123
6.2.70	tosca.datatypes.nfv.ExtendedResourceData	124
6.2.70.1	Description	124
6.2.70.2	Properties	124
6.2.70.3	Definition	124
6.2.70.4	Examples	125
6.2.70.5	Additional Requirements	125
6.2.71	tosca.datatypes.nfv.Hugepages	126
6.2.71.1	Description	126
6.2.71.2	Properties	126
6.2.71.3	Definition	126
6.2.71.4	Examples	127
6.2.71.5	Additional Requirements	127
6.2.72	tosca.datatypes.nfv.MaxNumberOfImpactedInstances	127
6.2.72.1	Description	127
6.2.72.2	Properties	127
6.2.72.3	Definition	128
6.2.72.4	Examples	128
6.2.72.5	Additional Requirements	128
6.2.73	tosca.datatypes.nfv.MinNumberOPreservedInstances	128
6.2.73.1	Description	128
6.2.73.2	Properties	129
6.2.73.3	Definition	129
6.2.73.4	Examples	129
6.2.73.5	Additional Requirements	130
6.2.74	tosca.datatypes.nfv.NfviMaintenanceInfo	130
6.2.74.1	Description	130
6.2.74.2	Properties	130
6.2.74.3	Definition	131
6.2.74.4	Examples	132

6.2.74.5	Additional Requirements	132
6.2.75	tosca.datatypes.nfv.McioIdentificationData	132
6.2.75.1	Description	132
6.2.75.2	Properties	132
6.2.75.3	Definition	133
6.2.75.4	Examples	133
6.2.75.5	Additional Requirements	133
6.2.76	tosca.datatypes.nfv.VipCpLevel	133
6.2.76.1	Description	133
6.2.76.2	Properties	133
6.2.76.3	Definition	134
6.2.76.4	Examples	134
6.2.76.5	Additional Requirements	134
6.2.77	tosca.datatypes.nfv.ServiceData	134
6.2.77.1	Description	134
6.2.77.2	Properties	134
6.2.77.3	Definition	134
6.2.77.4	Examples	135
6.2.77.5	Additional Requirements	135
6.2.78	tosca.datatypes.nfv. SelectVnfDeployableModulesOperationConfiguration.....	135
6.2.78.1	Description	135
6.2.78.2	Properties	135
6.2.78.3	Definition	135
6.2.78.4	Examples	135
6.2.78.5	Additional Requirements	136
6.2.79	tosca.datatypes.nfv.CertificateBaseProfile	136
6.2.79.1	Description	136
6.2.79.2	Properties	136
6.2.79.3	Definition	136
6.2.79.4	Examples	137
6.2.79.5	Additional requirements	137
6.2.80	tosca.datatypes.nfv.CSRRequirements	137
6.2.80.1	Description	137
6.2.80.2	Properties	137
6.2.80.3	Definition	138
6.2.80.4	Examples	139
6.2.80.5	Additional Requirements	140
6.2.81	tosca.datatypes.nfv.CertSubjectData.....	140
6.2.81.1	Description	140
6.2.81.2	Properties	140
6.2.81.3	Definition	140
6.2.81.4	Examples	141
6.2.81.5	Additional Requirements	141
6.2.82	tosca.datatypes.nfv.SizeRange.....	141
6.2.82.1	Description	141
6.2.82.2	Properties	141
6.2.82.3	Definition	142
6.2.82.4	Examples	142
6.2.82.5	Additional Requirements	142
6.2.83	tosca.datatypes.nfv.IntegerRange	142
6.2.83.1	Description	142
6.2.83.2	Properties	142
6.2.83.3	Definition	142
6.2.83.4	Examples	143
6.2.83.5	Additional Requirements	143
6.2.84	tosca.datatypes.nfv.CpuPowerState	143
6.2.84.1	Description	143
6.2.84.2	Properties	143
6.2.84.4	Examples	144
6.2.84.5	Additional Requirements	144
6.2.85	tosca.datatypes.nfv.PowerProfile	144
6.2.85.1	Description	144

6.2.85.2	Properties	145
6.2.85.3	Definition	145
6.2.85.4	Examples	146
6.2.85.5	Additional Requirements	146
6.2.86	tosca.datatypes.nfv.PowerConsumptionInfo	146
6.2.86.1	Description	146
6.2.86.2	Properties	146
6.2.86.3	Definition	146
6.2.86.4	Examples	147
6.2.86.5	Additional requirements	147
6.3	Artifact Types	147
6.3.1	tosca.artifacts.nfv.SwImage	147
6.3.1.1	Description	147
6.3.1.2	Properties	147
6.3.1.3	Definition	148
6.3.2	tosca.artifacts.Implementation.nfv.Mistral	150
6.3.2.1	Description	150
6.3.2.2	Definition	150
6.3.3	tosca.artifacts.nfv.HelmChart	150
6.3.3.1	Description	150
6.3.3.2	Properties	150
6.3.3.3	Description	150
6.3.4	tosca.artifacts.nfv.HelmParamMappingScript	151
6.3.4.1	Description	151
6.3.4.2	Properties	151
6.3.4.3	Definition	151
6.3.5	tosca.artifacts.nfv.HelmParamMappingRule	152
6.3.5.1	Description	152
6.3.5.2	Properties	152
6.3.5.3	Definition	152
6.4	Capability Types	152
6.4.1	tosca.capabilities.nfv.VirtualBindable	152
6.4.1.1	Description	152
6.4.1.2	Properties	152
6.4.1.3	Definition	152
6.4.2	tosca.capabilities.nfv.VirtualLinkable	153
6.4.2.1	Description	153
6.4.3	tosca.capabilities.nfv.VirtualCompute	153
6.4.3.1	Description	153
6.4.3.2	Properties	153
6.4.3.3	Definition	154
6.4.4	tosca.capabilities.nfv.VirtualStorage	154
6.4.4.1	Description	154
6.4.4.2	Definition	155
6.4.5	tosca.capabilities.nfv.TrunkBindable	155
6.4.5.1	Description	155
6.4.5.2	Properties	155
6.4.5.3	Definition	155
6.4.6	tosca.capabilities.nfv.ContainerDeployable	155
6.4.6.1	Description	155
6.4.6.2	Properties	155
6.4.6.3	Definition	156
6.4.7	tosca.capabilities.nfv.AssociableVdu	156
6.4.7.1	Description	156
6.4.7.2	Properties	156
6.4.7.3	Definition	156
6.4.8	tosca.capabilities.nfv.DeployableModuleMember	156
6.4.8.1	Description	156
6.4.8.2	Properties	157
6.4.8.3	Definition	157
6.4.9	tosca.capabilities.nfv.PaaSServiceRequestable	157
6.4.9.1	Description	157

6.4.9.2	Properties	157
6.4.9.3	Definition	157
6.4.10	tosca.capabilities.nfv.InstallableCertificate	157
6.4.10.1	Description	157
6.4.10.2	Definition	158
6.4.11	tosca.capabilities.nfv.AssociablePaasService	158
6.4.11.1	Description	158
6.5	Requirement Types	158
6.6	Relationship Types	158
6.6.1	tosca.relationships.nfv.VirtualBindsTo	158
6.6.1.1	Description	158
6.6.1.2	Properties	158
6.6.1.3	Definition	158
6.6.2	tosca.relationships.nfv.VirtualLinksTo	159
6.6.2.1	Description	159
6.6.3	tosca.relationships.nfv.AttachesTo	159
6.6.3.1	Description	159
6.6.3.2	Properties	159
6.6.3.3	Definition	159
6.6.4	tosca.relationships.nfv.TrunkBindsTo	159
6.6.4.1	Description	159
6.6.4.2	Properties	159
6.6.4.3	Definition	160
6.6.5	tosca.relationships.nfv.DeploysTo	160
6.6.5.1	Description	160
6.6.5.2	Properties	160
6.6.5.3	Definition	160
6.6.6	tosca.relationships.nfv.MciopAssociates	160
6.6.6.1	Description	160
6.6.6.2	Properties	160
6.6.6.3	Definition	161
6.6.7	tosca.relationships.nfv.DeployableModuleAssociates	161
6.6.7.1	Description	161
6.6.7.2	Properties	161
6.6.7.3	Definition	161
6.6.8	tosca.relationships.nfv.PaasServiceRequestsTo	161
6.6.8.1	Description	161
6.6.8.2	Properties	161
6.6.8.3	Definition	162
6.6.9	tosca.relationships.nfv.InstallsTo	162
6.6.9.1	Description	162
6.6.9.2	Properties	162
6.6.9.3	Definition	162
6.6.10	tosca.relationships.nfv.PaasServiceAssociates	162
6.6.10.1	Description	162
6.7	Interface Types	162
6.7.1	tosca.interfaces.nfv.Vnflcm	162
6.7.1.1	Description	162
6.7.1.2	Definition	163
6.7.1.3	Additional Requirements	167
6.7.1.4	Support of LCM scripts	167
6.7.1.5	Examples	168
6.7.2	tosca.interfaces.nfv.VnfIndicator	171
6.7.2.1	Description	171
6.7.2.2	Definition	171
6.7.2.3	Examples	171
6.7.3	tosca.interfaces.nfv.ChangeCurrentVnfPackage	171
6.7.3.1	Description	171
6.7.3.2	Definition	171
6.7.3.3	Examples	172
6.8	Node Types	172
6.8.1	tosca.nodes.nfv.VNF	172

6.8.1.1	Description	172
6.8.1.2	Properties	172
6.8.1.3	Attributes	175
6.8.1.4	Requirements	175
6.8.1.5	Capabilities	176
6.8.1.6	Definition	176
6.8.1.7	Artifact	178
6.8.1.8	Additional Requirements	178
6.8.1.9	Example	180
6.8.2	tosca.nodes.nfv.VnfExtCp	185
6.8.2.1	Description	185
6.8.2.2	Properties	185
6.8.2.3	Attributes	185
6.8.2.4	Requirements	185
6.8.2.5	Capabilities	185
6.8.2.6	Definition	185
6.8.2.7	Additional Requirements	186
6.8.2.8	Example	186
6.8.3	tosca.nodes.nfv.Vdu.Compute	187
6.8.3.1	Description	187
6.8.3.2	Properties	187
6.8.3.3	Attributes	188
6.8.3.4	Requirements	188
6.8.3.5	Capabilities	188
6.8.3.6	Definition	189
6.8.3.7	Additional requirements	190
6.8.3.8	Example	191
6.8.4	tosca.nodes.nfv.Vdu.VirtualBlockStorage	196
6.8.4.1	Description	196
6.8.4.2	Properties	197
6.8.4.3	Attributes	197
6.8.4.4	Requirements	197
6.8.4.5	Capabilities	197
6.8.4.6	Definition	197
6.8.4.7	Additional requirements	198
6.8.5	tosca.nodes.nfv.Vdu.VirtualObjectStorage	198
6.8.5.1	Description	198
6.8.5.2	Properties	198
6.8.5.3	Attributes	199
6.8.5.4	Requirements	199
6.8.5.5	Capabilities	199
6.8.5.6	Definition	199
6.8.5.7	Additional requirements	200
6.8.6	tosca.nodes.nfv.Vdu.VirtualFileStorage	200
6.8.6.1	Description	200
6.8.6.2	Properties	200
6.8.6.3	Attributes	200
6.8.6.4	Requirements	200
6.8.6.5	Capabilities	200
6.8.6.6	Definition	201
6.8.6.7	Additional requirements	201
6.8.7	tosca.nodes.nfv.Cp	201
6.8.7.1	Description	201
6.8.8	tosca.nodes.nfv.VduCp	202
6.8.8.1	Description	202
6.8.8.2	Properties	202
6.8.8.3	Attributes	203
6.8.8.4	Requirements	203
6.8.8.5	Capabilities	203
6.8.8.6	Definition	204
6.8.8.7	Additional Requirements	204
6.8.9	tosca.nodes.nfv.VnfVirtualLink	205

6.8.9.1	Description	205
6.8.9.2	Properties	205
6.8.9.3	Requirements	206
6.8.9.4	Capabilities.....	206
6.8.9.5	Definition	206
6.8.10	tosca.nodes.nfv.VipCp	207
6.8.10.1	Description	207
6.8.10.2	Properties	207
6.8.10.3	Attributes.....	207
6.8.10.4	Requirements	208
6.8.10.5	Definition	208
6.8.10.6	Example	208
6.8.11	tosca.nodes.nfv.VduSubCp	209
6.8.11.1	Description	209
6.8.11.2	Properties	209
6.8.11.3	Attributes.....	209
6.8.11.4	Requirements	209
6.8.11.5	Definition	210
6.8.11.6	Example	210
6.8.11.7	Additional Requirements	210
6.8.12	tosca.nodes.nfv.Vdu.OsContainer.....	211
6.8.12.1	Description	211
6.8.12.2	Properties	211
6.8.12.3	Attributes.....	214
6.8.12.4	Capabilities.....	214
6.8.12.5	Definition	215
6.8.12.6	Additional requirements.....	217
6.8.13	tosca.nodes.nfv.Vdu.OsContainerDeployableUnit	217
6.8.13.1	Description	217
6.8.13.2	Properties	218
6.8.13.3	Attributes.....	219
6.8.13.4	Requirements	219
6.8.13.5	Capabilities.....	220
6.8.13.6	Definition	220
6.8.14	tosca.nodes.nfv.Mciop	222
6.8.14.1	Description	222
6.8.14.2	Properties	223
6.8.14.3	Attributes.....	223
6.8.14.4	Requirements	223
6.8.14.5	Capabilities.....	223
6.8.14.6	Definition	223
6.8.14.7	Additional Requirements	224
6.8.14.8	Example	224
6.8.15	tosca.nodes.nfv.VirtualCp.....	224
6.8.15.1	Description	224
6.8.15.2	Properties	224
6.8.15.3	Attributes.....	224
6.8.15.4	Requirements	225
6.8.15.5	Definition	225
6.8.15.5a	Additional requirements.....	225
6.8.15.6	Example	226
6.8.16	tosca.nodes.nfv.DeployableModule	226
6.8.16.1	Description	226
6.8.16.2	Properties	227
6.8.16.3	Attributes.....	227
6.8.16.4	Requirements	227
6.8.16.5	Capabilities.....	227
6.8.16.6	Definition	227
6.8.16.7	Example	227
6.8.17	tosca.nodes.nfv.PaaSServiceRequest	228
6.8.17.1	Description	228
6.8.17.2	Properties	228

6.8.17.3	Attributes.....	228
6.8.17.4	Requirements	228
6.8.17.5	Capabilities.....	229
6.8.17.6	Definition	229
6.8.17.7	Additional requirements.....	230
6.8.17.8	Example	230
6.8.18	tosca.nodes.nfv.PaaSServiceProfile	230
6.8.18.1	Description	230
6.8.18.2	Properties	230
6.8.18.3	Attributes.....	230
6.8.18.4	Requirements	230
6.8.18.5	Capabilities.....	230
6.8.18.6	Definition	231
6.8.18.7	Additional requirements.....	231
6.8.18.8	Example	231
6.8.19	tosca.nodes.nfv.Certificate.....	231
6.8.19.1	Description	231
6.8.19.2	Properties	232
6.8.19.3	Attributes.....	232
6.8.19.4	Requirements	232
6.8.19.5	Capabilities.....	232
6.8.19.6	Definition	232
6.8.19.7	Additional requirements.....	233
6.8.19.8	Example	233
6.9	Group Types.....	234
6.9.1	tosca.groups.nfv.PlacementGroup	234
6.9.1.1	Description	234
6.9.1.2	Properties	234
6.9.1.3	Definition	234
6.9.1.4	Additional Requirements	234
6.9.1.5	Examples.....	234
6.10	Policy Types.....	235
6.10.1	tosca.policies.nfv.InstantiationLevels	235
6.10.1.1	Description	235
6.10.1.2	Properties	235
6.10.1.3	Definition	235
6.10.2	tosca.policies.nfv.VduInstantiationLevels	236
6.10.2.1	Description	236
6.10.2.2	Properties	236
6.10.2.3	Definition	236
6.10.2.4	Additional Requirements	236
6.10.3	tosca.policies.nfv.VirtualLinkInstantiationLevels	237
6.10.3.1	Description	237
6.10.3.2	Properties	237
6.10.3.3	Definition	237
6.10.3.4	Additional Requirements	237
6.10.4	Void	237
6.10.5	tosca.policies.nfv.ScalingAspects.....	238
6.10.5.1	Description	238
6.10.5.2	Properties	238
6.10.5.3	Definition	238
6.10.5.4	Additional Requirements	238
6.10.5.5	Examples.....	238
6.10.6	tosca.policies.nfv.VduScalingAspectDeltas	239
6.10.6.1	Description	239
6.10.6.2	Properties	239
6.10.6.3	Definition	239
6.10.6.4	Additional Requirements	240
6.10.6.5	Examples.....	240
6.10.7	tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas	240
6.10.7.1	Description	240
6.10.7.2	Properties	240

6.10.7.3	Definition	240
6.10.7.4	Additional Requirements	241
6.10.7.5	Examples	241
6.10.8	tosca.policies.nfv.VduInitialDelta	241
6.10.8.1	Description	241
6.10.8.2	Properties	241
6.10.8.3	Definition	241
6.10.8.4	Examples	242
6.10.9	tosca.policies.nfv.VirtualLinkBitrateInitialDelta	242
6.10.9.1	Description	242
6.10.9.2	Properties	242
6.10.9.3	Definition	242
6.10.9.4	Examples	243
6.10.10	AffinityRule, AntiAffinityRule	243
6.10.10.1	Description	243
6.10.10.2	Properties	243
6.10.10.3	targets	244
6.10.10.4	Definition	245
6.10.10.5	Examples	245
6.10.11	tosca.policies.nfv.Abstract.SecurityGroupRule	246
6.10.11.1	Description	246
6.10.12	tosca.policies.nfv.SupportedVnfInterface	246
6.10.12.1	Description	246
6.10.12.2	Properties	246
6.10.12.3	Definition	247
6.10.12.4	Additional requirements	247
6.10.12.5	Example	247
6.10.13	tosca.policies.nfv.SecurityGroupRule	248
6.10.13.1	Description	248
6.10.13.2	Properties	248
6.10.13.3	targets	248
6.10.13.4	Definition	248
6.10.13.5	Additional Requirements	248
6.10.14	tosca.policies.nfv.VnfIndicator	249
6.10.14.1	Description	249
6.10.14.2	Properties	249
6.10.14.3	Definition	249
6.10.14.4	Additional requirements	249
6.10.15	tosca.policies.nfv.VnfPackageChange	250
6.10.15.1	Description	250
6.10.15.2	Properties	250
6.10.15.3	Definition	251
6.10.15.4	Additional Requirements	252
6.10.15.5	Example	253
6.10.16	tosca.policies.nfv.LcmCoordinationAction	261
6.10.16.1	Description	261
6.10.16.2	Properties	261
6.10.16.3	Definition	262
6.10.16.4	Additional Requirements	262
6.10.17	tosca.policies.nfv.LcmCoordinationsForLcmOperation	262
6.10.17.1	Description	262
6.10.17.2	Properties	263
6.10.17.3	Definition	264
6.10.17.4	Additional Requirements	264
6.10.18	tosca.policies.nfv.VipCpScalingAspectDeltas	264
6.10.18.1	Description	264
6.10.18.2	Properties	265
6.10.18.3	Definition	265
6.10.18.4	Additional Requirements	265
6.10.18.5	Examples	265
6.10.19	tosca.policies.nfv.VipCpInitialDelta	266
6.10.19.1	Description	266

6.10.19.2	Properties	266
6.10.19.3	Definition	266
6.10.19.4	Additional Requirements	266
6.10.19.5	Examples	266
6.10.20	tosca.policies.nfv.VipCpInstantiationLevels	266
6.10.20.1	Description	266
6.10.20.2	Properties	267
6.10.20.3	Definition	267
6.10.20.4	Additional Requirements	267
6.10.21	tosca.policies.nfv.PowerProfiles	267
6.10.21.1	Description	267
6.10.21.2	Properties	268
6.10.21.3	Definition	268
6.11	VNFD TOSCA service template design	268
6.11.1	General	268
6.11.2	Single or multiple deployment flavour design with two levels of service templates	268
6.11.3	Single deployment flavour design with one service template	270
6.11.4	Package change (handling the Change current VNF Package request)	271
7	NSD TOSCA model	271
7.1	Introduction	271
7.2	Data Types	272
7.2.1	Void	272
7.2.2	tosca.datatypes.nfv.VnfProfile	272
7.2.2.1	Description	272
7.2.3	tosca.datatype.nfv.NsVIProfile	272
7.2.3.1	Description	272
7.2.3.2	Properties	272
7.2.3.3	Definition	273
7.2.3.4	Examples	273
7.2.3.5	Additional Requirements	273
7.2.4	tosca.datatypes.nfv.ConnectivityType	274
7.2.4.1	Description	274
7.2.5	tosca.datatypes.nfv.NsVirtualLinkQos	274
7.2.5.1	Description	274
7.2.5.2	Properties	274
7.2.5.3	Definition	274
7.2.5.4	Examples	274
7.2.5.5	Additional Requirements	274
7.2.6	tosca.datatypes.nfv.LinkBitrateRequirements	275
7.2.6.1	Description	275
7.2.7	Void	275
7.2.8	Void	275
7.2.9	Void	275
7.2.10	Void	275
7.2.11	tosca.datatypes.nfv.CpProtocolData	275
7.2.11.1	Description	275
7.2.12	tosca.datatypes.nfv.AddressData	275
7.2.12.1	Description	275
7.2.13	tosca.datatypes.nfv.L2AddressData	275
7.2.13.1	Description	275
7.2.14	tosca.datatypes.nfv.L3AddressData	275
7.2.14.1	Description	275
7.2.15	tosca.datatypes.nfv.Qos	275
7.2.15.1	Description	275
7.2.16	tosca.datatypes.nfv.NsProfile	275
7.2.16.1	Description	275
7.2.16.2	Properties	276
7.2.16.3	Definition	276
7.2.16.4	Example	277
7.2.16.5	Additional Requirements	277
7.2.17	tosca.datatypes.nfv.Mask	278

7.2.17.1	Description	278
7.2.17.2	Properties	278
7.2.17.3	Definition	278
7.2.17.4	Examples	278
7.2.18	tosca.datatypes.nfv.NsOperationAdditionalParameters	279
7.2.18.1	Description	279
7.2.18.2	Properties	279
7.2.18.3	Definition	279
7.2.18.4	Examples	279
7.2.19	tosca.datatypes.nfv.NsMonitoringParameter	280
7.2.19.1	Description	280
7.2.19.2	Properties	280
7.2.19.3	Definition	281
7.2.19.4	Examples	281
7.2.19.5	Additional Requirements	281
7.2.20	tosca.datatypes.nfv.VnfMonitoringParameter	282
7.2.21	tosca.datatypes.nfv.NsVirtualLinkProtocolData	282
7.2.21.1	Description	282
7.2.21.2	Properties	282
7.2.21.3	Definition	282
7.2.21.4	Examples	283
7.2.21.5	Additional Requirements	283
7.2.22	tosca.datatypes.nfv.NsL2ProtocolData	283
7.2.22.1	Description	283
7.2.22.2	Properties	283
7.2.22.3	Definition	284
7.2.22.4	Examples	285
7.2.22.5	Additional Requirements	285
7.2.23	tosca.datatypes.nfv.NsL3ProtocolData	285
7.2.23.1	Description	285
7.2.23.2	Properties	285
7.2.23.3	Definition	285
7.2.23.4	Examples	286
7.2.23.5	Additional Requirements	286
7.2.24	tosca.datatypes.nfv.NsIpAllocationPool	286
7.2.24.1	Description	286
7.2.24.2	Properties	286
7.2.24.3	Definition	287
7.2.24.4	Examples	287
7.2.24.5	Additional Requirements	287
7.2.25	tosca.datatypes.nfv.NsScalingAspect	287
7.2.25.1	Description	287
7.2.25.2	Properties	287
7.2.25.3	Definition	288
7.2.25.4	Examples	288
7.2.25.5	Additional Requirements	288
7.2.26	tosca.datatypes.nfv.NsLevels	288
7.2.26.1	Description	288
7.2.26.2	Properties	288
7.2.26.3	Definition	289
7.2.26.4	Examples	289
7.2.26.5	Additional Requirements	289
7.2.27	tosca.datatypes.nfv.ScaleNsByStepsData	289
7.2.27.1	Description	289
7.2.27.2	Properties	289
7.2.27.3	Definition	289
7.2.27.4	Examples	290
7.2.27.5	Additional Requirements	290
7.2.28	tosca.datatypes.nfv.ScaleNsToLevelData	290
7.2.28.1	Description	290
7.2.28.2	Properties	290
7.2.28.3	Definition	291

7.2.28.4	Examples	291
7.2.28.5	Additional Requirements	291
7.2.29	tosca.datatypes.nfv.NsDataFlowMirroring	291
7.2.29.1	Description	291
7.2.29.2	Properties	291
7.2.29.3	Definition	292
7.2.29.4	Example	292
7.2.29.5	Additional Requirements	292
7.2.30	tosca.datatypes.nfv.NsScaleInfo	292
7.2.30.1	Description	292
7.2.30.2	Properties	292
7.2.30.3	Definition	293
7.2.30.4	Examples	293
7.2.30.5	Additional Requirements	293
7.3	Artifact Types	293
7.4	Capability Types	293
7.4.1	tosca.capabilities.nfv.VirtualLinkable	293
7.4.1.1	Description	293
7.4.2	tosca.capabilities.nfv.Forwarding	293
7.4.2.1	Description	293
7.4.2.2	Properties	294
7.4.2.3	Definition	294
7.4.3	tosca.capabilities.nfv.NsPaasServiceRequestable	294
7.4.3.1	Description	294
7.4.3.2	Properties	294
7.4.3.3	Definition	294
7.4.4	tosca.capabilities.nfv.AssociablePaasService	294
7.4.4.1	Description	294
7.4.5	tosca.capabilities.nfv.VirtualLinkCapabilityConfigurable	295
7.4.5.1	Description	295
7.4.5.2	Properties	295
7.4.5.3	Definition	295
7.5	Requirement Types	295
7.6	Relationship Types	295
7.6.1	tosca.relationships.nfv.VirtualLinksTo	295
7.6.1.1	Description	295
7.6.2	tosca.relationships.nfv.ForwardTo	295
7.6.2.1	Description	295
7.6.2.2	Properties	296
7.6.2.3	Definition	296
7.6.3	tosca.relationships.nfv.NsPaasServiceRequestsTo	296
7.6.3.1	Description	296
7.6.3.2	Properties	296
7.6.3.3	Definition	296
7.6.4	tosca.relationships.nfv.PaasServiceAssociates	296
7.6.4.1	Description	296
7.6.5	tosca.relationships.nfv.VirtualLinkCapabilityConfiguresTo	296
7.6.5.1	Description	296
7.6.5.2	Properties	297
7.6.5.3	Definition	297
7.7	Interface Types	297
7.7.1	tosca.interfaces.nfv.Nslcm	297
7.7.1.1	Description	297
7.7.1.2	Definition	298
7.7.1.3	Additional Requirements	299
7.7.1.4	Support of LCM scripts	299
7.7.1.5	Examples	299
7.7.2	tosca.interfaces.nfv.NsVnfIndicator	300
7.7.2.1	Description	300
7.7.2.2	Definition	301
7.7.2.3	Examples	301
7.8	Node Types	301

7.8.1	tosca.nodes.nfv.NS	301
7.8.1.1	Description	301
7.8.1.2	Properties	301
7.8.1.3	Attributes.....	302
7.8.1.4	Requirements	303
7.8.1.5	Capabilities.....	303
7.8.1.6	Definition	303
7.8.1.7	Artifact	305
7.8.1.8	Additional requirements.....	305
7.8.2	tosca.nodes.nfv.Sap.....	305
7.8.2.1	Description	305
7.8.2.2	Properties	305
7.8.2.3	Attributes.....	305
7.8.2.4	Requirements	305
7.8.2.5	Capabilities.....	306
7.8.2.6	Definition	306
7.8.2.7	Additional requirements.....	306
7.8.2.8	Example	306
7.8.3	tosca.nodes.nfv.NsVirtualLink	307
7.8.3.1	Description	307
7.8.3.2	Properties	307
7.8.3.3	Attributes.....	307
7.8.3.4	Requirements	307
7.8.3.5	Capabilities.....	308
7.8.3.6	Definition	308
7.8.3.7	Artifact	308
7.8.3.8	Additional Requirements	309
7.8.3.9	Example	309
7.8.4	tosca.nodes.nfv.Cp	309
7.8.4.1	Description	309
7.8.5	tosca.nodes.nfv.NfpPositionElement	309
7.8.5.1	Description	309
7.8.5.2	Properties	309
7.8.5.3	Attributes.....	309
7.8.5.4	Requirements	309
7.8.5.5	Capabilities.....	310
7.8.5.6	Definition	310
7.8.5.7	Artifact	310
7.8.5.8	Additional Requirements	310
7.8.5.9	Example	310
7.8.6	tosca.nodes.nfv.NFP	310
7.8.6.1	Description	310
7.8.6.2	Properties	311
7.8.6.3	Attributes.....	311
7.8.6.4	Requirements	311
7.8.6.5	Capabilities.....	311
7.8.6.6	Definition	311
7.8.7	tosca.nodes.nfv.NfpPosition	311
7.8.7.1	Description	311
7.8.7.2	Properties	311
7.8.7.3	Attributes.....	312
7.8.7.4	Requirements	312
7.8.7.5	Capabilities.....	312
7.8.7.6	Definition	313
7.8.7.7	Artifact	313
7.8.7.8	Additional Requirements	313
7.8.7.9	Example	313
7.8.8	tosca.nodes.nfv.Forwarding	313
7.8.8.1	Description	313
7.8.8.2	Properties	314
7.8.8.3	Attributes.....	314
7.8.8.4	Requirements	314

7.8.8.5	Capabilities.....	314
7.8.8.6	Definition	314
7.8.8.7	Artifact	314
7.8.8.8	Additional Requirements	315
7.8.8.9	Example	315
7.8.9	tosca.nodes.nfv.NsPaasServiceRequest	315
7.8.9.1	Description	315
7.8.9.2	Properties	315
7.8.9.3	Attributes.....	316
7.8.9.4	Requirements	316
7.8.9.5	Capabilities.....	316
7.8.9.6	Definition	316
7.8.9.7	Artifact	317
7.8.9.8	Additional requirements.....	317
7.8.9.9	Example	317
7.8.10	tosca.nodes.nfv.NsPaasServiceProfile	317
7.8.10.1	Description	317
7.8.10.2	Properties	318
7.8.10.3	Attributes.....	318
7.8.10.4	Requirements	318
7.8.10.5	Capabilities.....	318
7.8.10.6	Definition	319
7.8.10.7	Additional requirements.....	319
7.8.10.8	Example	319
7.8.11	tosca.nodes.nfv.NsVirtualLinkAdditionalCapability	319
7.8.11.1	Description	319
7.8.11.2	Properties	319
7.8.11.3	Attributes.....	320
7.8.11.4	Requirements	320
7.8.11.5	Capabilities.....	320
7.8.11.6	Definition	320
7.8.11.7	Artifact	321
7.8.11.8	Additional requirements.....	321
7.8.11.9	Example	321
7.9	Group Types.....	321
7.9.1	tosca.groups.nfv.NsPlacementGroup	321
7.9.1.1	Description	321
7.9.1.2	Properties	321
7.9.1.3	Definition	322
7.9.1.4	Additional Requirements	322
7.9.2	tosca.groups.nfv.VNFFG	322
7.9.2.1	Description	322
7.9.2.2	Properties	322
7.9.2.3	Definition	323
7.9.2.4	Additional Requirements	323
7.9.2.5	Example	323
7.10	Policy Types.....	323
7.10.1	NsAffinityRule, NsAntiAffinityRule	323
7.10.1.1	Description	323
7.10.1.2	Properties	324
7.10.1.3	Targets.....	324
7.10.1.4	Definition	325
7.10.1.5	Examples.....	325
7.10.2	tosca.policies.nfv.NsSecurityGroupRule	326
7.10.2.1	Description	326
7.10.2.2	Properties	326
7.10.2.3	targets	326
7.10.2.4	Definition	326
7.10.2.5	Additional Requirements	327
7.10.3	tosca.policies.nfv.NfpRule	327
7.10.3.1	Description	327
7.10.3.2	Properties	327

7.10.3.3	Targets.....	328
7.10.3.4	Definition	328
7.10.3.5	Example	330
7.10.4	tosca.policies.nfv.NsMonitoring	330
7.10.4.1	Description	330
7.10.4.2	Properties	330
7.10.4.3	targets	330
7.10.4.4	Definition	330
7.10.4.5	Additional Requirements	331
7.10.5	tosca.policies.nfv.VnfMonitoring	331
7.10.5.1	Description	331
7.10.5.2	Properties	331
7.10.5.3	targets	331
7.10.5.4	Definition	331
7.10.5.5	Additional Requirements	332
7.10.6	tosca.policies.nfv.Abstract.SecurityGroupRule	332
7.10.6.1	Description	332
7.10.7	tosca.policies.nfv.NsScalingAspects	332
7.10.7.1	Description	332
7.10.7.2	Properties	332
7.10.7.3	Definition	333
7.10.7.4	Examples	333
7.10.8	tosca.policies.nfv.VnfToLevelMapping	333
7.10.8.1	Description	333
7.10.8.2	Properties	333
7.10.8.3	Definition	334
7.10.8.4	Examples	334
7.10.9	tosca.policies.nfv.NsToLevelMapping	334
7.10.9.1	Description	334
7.10.9.2	Properties	334
7.10.9.3	Definition	335
7.10.9.4	Examples	335
7.10.10	tosca.policies.nfv.VirtualLinkToLevelMapping	335
7.10.10.1	Description	335
7.10.10.2	Properties	336
7.10.10.3	Definition	336
7.10.10.4	Examples	336
7.10.11	tosca.policies.nfv.NsInstantiationLevels.....	336
7.10.11.1	Description	336
7.10.11.2	Properties	337
7.10.11.3	Definition	337
7.10.12	tosca.policies.nfv.VnfToInstantiationLevelMapping.....	337
7.10.12.1	Description	337
7.10.12.2	Properties	338
7.10.12.3	Definition	338
7.10.12.4	Examples	338
7.10.12.5	Additional requirements	338
7.10.13	tosca.policies.nfv.NsToInstantiationLevelMapping	338
7.10.13.1	Description	338
7.10.13.2	Properties	339
7.10.13.3	Definition	339
7.10.13.4	Examples	339
7.10.13.5	Additional requirements	339
7.10.14	tosca.policies.nfv.VirtualLinkToInstantiationLevelMapping.....	340
7.10.14.1	Description	340
7.10.14.2	Properties	340
7.10.14.3	Definition	340
7.10.14.4	Examples	340
7.10.15	tosca.policies.nfv.NsAutoScale	341
7.10.15.1	Description	341
7.10.15.2	Properties	341
7.10.15.3	Definition	341

7.10.15.4	Additional requirements	341
7.10.16	tosca.policies.nfv.DataFlowInfo	341
7.10.16.1	Description	341
7.10.16.2	Properties	342
7.10.16.3	Definition	342
7.10.16.4	Additional requirements	343
7.10.17	tosca.policies.nfv.SapToDeployableModuleMapping	343
7.10.17.1	Description	343
7.10.17.2	Properties	343
7.10.17.3	Definition	344
7.10.17.4	Additional requirements	344
7.10.17.5	Example	344
7.11	NSD TOSCA service template design.....	345
7.11.1	General.....	345
7.11.2	Single or multiple deployment flavour design with two levels of service templates	345
7.11.3	Single deployment flavour design with one service template	347
8	PNFD TOSCA model.....	348
8.1	Introduction	348
8.2	Data Types.....	349
8.2.1	tosca.datatypes.nfv.CpProtocolData	349
8.2.1.1	Description	349
8.2.2	tosca.datatypes.nfv.AddressData	349
8.2.2.1	Description	349
8.2.3	tosca.datatypes.nfv.L2AddressData	349
8.2.3.1	Description	349
8.2.4	tosca.datatypes.nfv.L3AddressData	349
8.2.4.1	Description	349
8.2.5	tosca.datatypes.nfv.LocationInfo	350
8.2.5.1	Description	350
8.2.5.2	Properties	350
8.2.5.3	Definition	350
8.2.5.4	Examples	351
8.2.5.5	Additional Requirements	351
8.2.6	tosca.datatypes.nfv.CivicAddressElement	351
8.2.6.1	Description	351
8.2.6.2	Properties	351
8.2.6.3	Definition	351
8.2.6.4	Examples	352
8.2.6.5	Additional Requirements	352
8.2.7	tosca.datatypes.nfv.GeographicCoordinates	352
8.2.7.1	Description	352
8.2.7.2	Properties	352
8.2.7.3	Definition	353
8.2.7.4	Examples	353
8.2.7.5	Additional Requirements	353
8.3	Artifact Types.....	353
8.4	Capability Types	353
8.4.1	tosca.capabilities.nfv.VirtualLinkable	353
8.4.1.1	Description	353
8.5	Requirement Types.....	354
8.6	Relationship Types.....	354
8.6.1	tosca.relationships.nfv.VirtualLinksTo.....	354
8.6.1.1	Description	354
8.7	Interface Types	354
8.8	Node Types	354
8.8.1	tosca.nodes.nfv.PNF	354
8.8.1.1	Description	354
8.8.1.2	Properties	354
8.8.1.3	Attributes.....	355
8.8.1.4	Requirements	355
8.8.1.5	Capabilities.....	356

8.8.1.6	Definition	356
8.8.1.7	Artifact	357
8.8.1.8	Additional Requirements	357
8.8.1.9	Example	357
8.8.2	tosca.nodes.nfv.PnfExtCp	357
8.8.2.1	Description	357
8.8.2.2	Properties	357
8.8.2.3	Attributes	357
8.8.2.4	Requirements	358
8.8.2.5	Capabilities	358
8.8.2.6	Definition	358
8.8.3	tosca.nodes.nfv.Cp	358
8.8.3.1	Description	358
8.9	Group Types	358
8.10	Policy Types	358
8.10.1	tosca.policies.nfv.PnfSecurityGroupRule	358
8.10.1.1	Description	358
8.10.1.2	Properties	359
8.10.1.3	targets	359
8.10.1.4	Definition	359
8.10.1.5	Additional Requirements	359
8.10.2	tosca.policies.nfv.Abstract.SecurityGroupRule	359
8.10.2.1	Description	359
8.11	PNFD TOSCA service template design	359
8.11.1	General	359
9	Common Definitions	360
9.1	Introduction	360
9.2	Data Types	360
9.2.1	tosca.datatypes.nfv.L2AddressData	360
9.2.1.1	Description	360
9.2.1.2	Properties	360
9.2.1.3	Definition	361
9.2.1.4	Examples	361
9.2.1.5	Additional Requirements	361
9.2.2	tosca.datatypes.nfv.L3AddressData	361
9.2.2.1	Description	361
9.2.2.2	Properties	361
9.2.2.3	Definition	363
9.2.2.4	Examples	364
9.2.2.5	Additional Requirements	365
9.2.3	tosca.datatypes.nfv.AddressData	365
9.2.3.1	Description	365
9.2.3.2	Properties	365
9.2.3.3	Definition	366
9.2.3.4	Examples	366
9.2.3.5	Additional Requirements	366
9.2.4	tosca.datatypes.nfv.ConnectivityType	367
9.2.4.1	Description	367
9.2.4.2	Properties	367
9.2.4.3	Definition	367
9.2.4.4	Examples	368
9.2.4.5	Additional Requirements	368
9.2.5	tosca.datatypes.nfv.LinkBitrateRequirements	368
9.2.5.1	Description	368
9.2.5.2	Properties	368
9.2.5.3	Definition	369
9.2.5.4	Examples	369
9.2.5.5	Additional Requirements	369
9.2.6	tosca.datatypes.nfv.CpProtocolData	369
9.2.6.1	Description	369
9.2.6.2	Properties	369

9.2.6.3	Definition	370
9.2.6.4	Examples	370
9.2.6.5	Additional Requirements	370
9.2.7	tosca.datatypes.nfv.Qos	370
9.2.7.1	Description	370
9.2.7.2	Properties	370
9.2.7.3	Definition	371
9.2.7.4	Examples	371
9.2.7.5	Additional Requirements	371
9.2.8	tosca.datatypes.nfv.VnfProfile	371
9.2.8.1	Description	371
9.2.8.2	Properties	372
9.2.8.3	Definition	373
9.2.8.4	Example	374
9.2.8.5	Additional Requirements	374
9.2.9	tosca.datatypes.nfv.VnfMonitoringParameter	374
9.2.9.1	Description	374
9.2.9.2	Properties	374
9.2.9.3	Definition	375
9.2.9.4	Examples	375
9.2.9.5	Additional Requirements	376
9.2.10	tosca.datatypes.nfv.VersionDependency	376
9.2.10.1	Description	376
9.2.10.2	Properties	376
9.2.10.3	Definition	376
9.2.10.4	Examples	377
9.2.10.5	Additional Requirements	377
9.2.11	tosca.datatypes.nfv.VersionDependencyStatement	377
9.2.11.1	Description	377
9.2.11.2	Properties	378
9.2.11.3	Definition	378
9.2.11.4	Examples	378
9.2.11.5	Additional Requirements	378
9.2.12	tosca.datatypes.nfv.ScaleInfo	378
9.2.12.1	Description	378
9.2.12.2	Properties	379
9.2.12.3	Definition	379
9.2.12.4	Examples	379
9.2.12.5	Additional Requirements	379
9.2.13	tosca.datatypes.nfv.SelectedDeployableModules	379
9.2.13.1	Description	379
9.2.13.2	Properties	379
9.2.13.3	Definition	380
9.2.13.4	Examples	380
9.2.13.5	Additional Requirements	380
9.2.14	tosca.datatypes.nfv.DomainNameData	381
9.2.14.1	Description	381
9.2.14.2	Properties	381
9.2.14.3	Definition	382
9.2.14.4	Examples	382
9.3	Artifact Types	382
9.4	Capability Types	382
9.4.1	tosca.capabilities.nfv.VirtualLinkable	382
9.4.1.1	Description	382
9.4.1.2	Properties	383
9.4.1.3	Definition	383
9.4.2	Void	383
9.4.3	tosca.capabilities.nfv.AssociablePaasService	383
9.4.3.1	Description	383
9.4.3.2	Properties	383
9.4.3.3	Definition	383
9.5	Requirement Types	383

9.6	Relationship Types	384
9.6.1	tosca.relationships.nfv.VirtualLinksTo	384
9.6.1.1	Description	384
9.6.1.2	Properties	384
9.6.1.3	Definition	384
9.6.2	Void	384
9.6.3	tosca.relationships.nfv.VipVirtualLinksTo	384
9.6.3.1	Description	384
9.6.3.2	Properties	384
9.6.3.3	Definition	384
9.6.4	tosca.relationships.nfv.PaasServiceAssociates	385
9.6.4.1	Description	385
9.6.4.2	Properties	385
9.6.4.3	Definition	385
9.7	Interface Types	385
9.8	Node Types	385
9.8.1	tosca.nodes.nfv.Cp	385
9.8.1.1	Description	385
9.8.1.2	Properties	386
9.8.1.3	Attributes	387
9.8.1.4	Requirements	387
9.8.1.5	Capabilities	387
9.8.1.6	Definition	387
9.8.1.7	Additional requirements	388
9.9	Group Types	388
9.10	Policy Types	389
9.10.1	tosca.policies.nfv.Abstract.SecurityGroupRule	389
9.10.1.1	Description	389
9.10.1.2	Properties	389
9.10.1.3	Definition	390
9.10.1.4	Additional Requirements	391
Annex A (informative):	Examples	392
A.1	Deployment flavour design mapping	392
A.1.1	Introduction	392
A.1.2	Design principle for VNF deployment flavour	392
A.1.3	Design principle for NS deployment flavour	393
A.2	VNFD with deployment flavour modelling design example	393
A.3	VNF external connection point	403
A.3.1	General	403
A.3.2	External connection point re-exposing an internal connection point	403
A.3.3	External connection point connected to an internal virtual link	406
A.4	VNFD modelling design example by using TOSCA composition	408
A.5	VNFD with Single deployment flavour modelling design example	412
A.6	Scaling and Instantiation Level examples	416
A.6.1	ScalingAspect and InstantiationLevels policies with uniform delta	416
A.6.2	ScalingAspect and InstantiationLevels policies with non-uniform deltas	423
A.7	Service Access Point	427
A.7.1	General	427
A.7.2	VNF External connection point exposing as a SAP	427
A.7.3	SAP connected to an NS virtual link	429
A.8	NSD with Single deployment flavour modelling design example	431
A.9	Mapping between NFV IM and TOSCA concepts	435
A.9.1	Introduction	435
A.9.2	Mapping between ETSI GS NFV-IFA 011 IM and TOSCA concepts	435
A.9.3	Mapping between ETSI GS NFV-IFA 014 IM and TOSCA concepts	437

A.10	PNFD modelling design example.....	438
A.11	NSD with Multiple deployment flavour modelling design example.....	440
A.12	NSD with nested NS design example.....	446
A.13	Virtual IP address connection point	451
A.14	NSD VNF Forwarding Graph design example	458
A.15	Auto-scale and auto-heal design.....	467
A.15.1	Introduction	467
A.15.2	Auto-scale and auto-heal design with use of VNF indicator in VNFD	467
A.15.3	Auto-scale design with use of VNF indicator in NSD.....	471
A.16	VDU connection point in trunk mode	478
A.17	NS scaling	481
A.18	VNFD illustrating OsContainer modelling example.....	486
A.19	NSD with data flow mirroring information.....	492
A.20	NSD example with l2_network scope used in NsAffinityRule/NsAntiAffinityRule	495
A.21	VNFD with deployable modules design example.....	499
A.22	NSD with SAP to deployable modules mapping	502
A.22.1	Deployable modules in a constituent VNF of the NS.....	502
A.22.2	Deployable modules in a constituent nested NS of a composite NS	504
A.23	VNFD example with simplified design by using MCIOP.....	506
Annex B (normative):	etsi_nfv_sol001_type definitions	511
B.1	Purpose.....	511
B.2	VNFD type definitions file.....	511
B.3	NSD type definitions file.....	511
B.4	PNFD type definitions file	512
B.5	Common type definitions file.....	512
Annex C (normative):	Conformance	513
C.1	Purpose.....	513
C.2	NFV TOSCA YAML service template	513
C.3	NFV TOSCA processor.....	514
Annex D (informative):	Mapping between properties of TOSCA types and API attributes.....	515
D.1	Introduction	515
D.2	VNFD-related constructs.....	515
D.3	NSD-related constructs.....	526
Annex E (informative):	TOSCA Imperative workflows.....	532
E.1	Purpose.....	532
E.2	TOSCA Imperative workflows for the NSD	532
E.2.1	Introduction	532
E.2.2	Definition of an NS workflow	532
E.2.3	Examples	533
Annex F (informative):	Non-Backward Compatible Changes in the GS.....	538

F.1	Introduction	538
F.2	Non-Backward Compatible changes between version 2.6.1 and 2.7.1	538
F.3	Non-Backward Compatible changes between version 2.8.1 and 3.3.1	538
F.4	Non-Backward Compatible changes between version 3.3.1 and 3.5.1	539
F.5	Non-Backward Compatible changes between version 3.5.1 and 4.2.1	539
F.6	Non-Backward Compatible changes between version 4.2.1 and 4.3.1	539
Annex G (informative): Change History		541
History		550

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the [ETSI IPR online database](#).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the [ETSI Drafting Rules](#) (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document specifies a data model for NFV descriptors, using the TOSCA-Simple-Profile-YAML-v1.3 [20], fulfilling the requirements specified in ETSI GS NFV-IFA 011 [1] and ETSI GS NFV-IFA 014 [2] for a Virtualised Network Function Descriptor (VNFD), a Network Service Descriptor (NSD) and a Physical Network Function Descriptor (PNFD). The present document also specifies requirements on the VNFM and NFVO specific to the handling of NFV descriptors based on the TOSCA-Simple-Profile-YAML-v1.3 [20].

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at [ETSI docbox](#).

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

- [1] [ETSI GS NFV-IFA 011](#): "Network Functions Virtualisation (NFV) Release 5; Management and Orchestration; VNF Descriptor and Packaging Specification".
- [2] [ETSI GS NFV-IFA 014](#): "Network Functions Virtualisation (NFV) Release 5; Management and Orchestration; Network Service Templates Specification".
- [3] Void.
- [4] Void.
- [5] [Private Enterprise Numbers registry at IANA](#).
- [6] [IETF RFC 5234](#): "Augmented BNF for Syntax Specifications: ABNF".
- [7] [ETSI GS NFV-IFA 027](#): "Network Functions Virtualisation (NFV) Release 5; Management and Orchestration; Performance Measurements Specification".
- [8] [IETF RFC 3986](#): "Uniform Resource Identifier (URI): Generic Syntax".
- [9] [IETF RFC 4122](#): "A Universally Unique Identifier (UUID) URN Namespace".
- [10] [ISO 3166 \(all parts\)](#): "Codes for the representation of names of countries and their subdivisions".
- [11] [IETF RFC 4776](#): "Dynamic Host Configuration Protocol (DHCPv4 and DHCPv6) Option for Civic Addresses Configuration Information".
- [12] Void.
- [13] [IETF RFC 5646](#): "Tags for Identifying Languages".
- [14] [Hash Function Textual Names registry at IANA](#).
- [15] [IEEE™ 1003.1-2017](#): "IEEE Standard for Information Technology -- Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7", 2018 edition.

NOTE: Revision of IEEE™ Std 1003.1-2008.

- [16] [IEEE™ 802.1Q-2018](#): "IEEE Standard for Local and Metropolitan Area Networks - Bridges and Bridged Networks".
- [17] [IETF RFC 791](#): "Internet Protocol".
- [18] [IETF RFC 8200](#): "Internet Protocol, Version 6 (IPv6) Specification".
- [19] [IANA](#): "Assigned Internet Protocol Numbers".
- [20] [OASIS](#): "TOSCA Simple Profile in YAML Version 1.3".
- [21] [IETF RFC 6225](#): "Dynamic Host Configuration Protocol Options for Coordinate-Based Location Configuration Information".
- [22] [ETSI GS NFV-SOL 002](#): "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; RESTful protocols specification for the Ve-Vnfm Reference Point".
- [23] [Helm™ Charts V3.11.1](#).
- [24] [IETF RFC 9110](#): "HTTP Semantics".
- [25] [ETSI GS NFV-SOL 003](#): "Network Functions Virtualisation (NFV) Release 5; Protocols and Data Models; RESTful protocols specification for the Or-Vnfm Reference Point".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

- [i.1] ETSI GS NFV-IFA 007: "Network Functions Virtualisation (NFV) Release 5; Management and Orchestration; Or-Vnfm reference point - Interface and Information Model Specification".
- [i.2] ETSI GR NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".
- [i.3] Void.
- [i.4] Void.
- [i.5] Void.
- [i.6] ETSI GS NFV-SOL 004: "Network Functions Virtualisation (NFV) Release 5; Protocols and Data Models; VNF Package and PNFD Archive specification".
- [i.7] [Mistral Workflow Language v2 specification](#).
- [i.8] ETSI GS NFV-IFA 013: "Network Functions Virtualisation (NFV) Release 5; Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification".
- [i.9] Void.
- [i.10] ETSI GS NFV-SOL 005: "Network Functions Virtualisation (NFV) Release 5; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point".
- [i.11] ETSI GS NFV-SOL 007: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; Network Service Descriptor File Structure Specification".

[i.12] [OpenStack® documentation](#): "Disk and container formats for images".

NOTE: The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. ETSI is not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

[i.13] [Openstack](#): "Metadata service".

[i.14] [Openstack](#): "Provide user data to instances".

[i.15] [Openstack](#): "Personality Class".

[i.16] [ETSI NFV registry of VimConnectionInfo information](#).

[i.17] [IETF RFC 4090](#): "Fast Reroute Extensions to RSVP-TE for LSP Tunnels".

[i.18] TOSCA-Simple-Profile-yaml-v1.2: "TOSCA Simple Profile in YAML Version 1.2".

[i.19] ETSI GS NFV-SOL 001 (V3.3.1): "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; NFV descriptors based on TOSCA specification".

[i.20] ETSI GS NFV-SOL 001 (V2.8.1): "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on TOSCA specification".

[i.21] ETSI GS NFV-SOL 001 (V3.5.1): "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; NFV descriptors based on TOSCA specification".

[i.22] ETSI GS NFV-SOL 001 (V2.6.1): "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on TOSCA specification".

[i.23] ETSI GS NFV-SOL 001 (V2.7.1): "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on TOSCA specification".

[i.24] ETSI GS NFV-SOL 001 (V4.2.1): "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; NFV descriptors based on TOSCA specification".

[i.25] ETSI GS NFV-SOL 001 (V4.3.1): "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; NFV descriptors based on TOSCA specification".

[i.26] Void.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI GR NFV 003 [i.2] and the following apply:

TOSCA interface type: reusable entity that describes a set of TOSCA operations that can be included as part of a Node type or Relationship Type definition

NOTE: See TOSCA-Simple-Profile-YAML-v1.3 [20].

TOSCA operation: behavioural lifecycle procedure in a TOSCA node or relationship definition that can be invoked by an orchestration engine, whose implementation definition can be provided in the service template as part of a node template definition or a relationship template definition, or rely on an implementation of the operation built in the orchestration engine

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GR NFV 003 [i.2] and the following apply:

DNS	Domain Name System
FQDN	Fully Qualified Domain Name
iBGP	interior BGP
NVE	Network Virtualization Edge
NVGRE	Network Virtualization using Generic Routing Encapsulation

4 Overview of TOSCA model

TOSCA (Topology and Orchestration Specification for Cloud Applications) is a modelling language for describing the components of a cloud application and their relationships. TOSCA uses the concept of service templates to describe cloud workloads. TOSCA further provides means of associating standard or user-defined lifecycle operations to a cloud application component or to a relationship between components. The present document is based on TOSCA-Simple-Profile-YAML-v1.3 [20], which describes a YAML rendering for TOSCA.

5 General concept of using TOSCA to model NFV descriptors

5.1 Introduction

An NFV deployment template is modelled by using one or more TOSCA service template as defined in TOSCA-Simple-Profile-YAML-v1.3 [20].

Three main deployment templates are identified in the present document:

- The Virtualised Network Function Descriptor (VNFD).
- The Network Service Descriptor (NSD).
- The Physical Network Function Descriptor (PNFD).

When processing TOSCA service templates modelling all or part of an NFV descriptor, the consumer of the NFV descriptor shall comply with and implement the semantics of any of the keynames defined in clause 3.10 of TOSCA-Simple-Profile-YAML-v1.3 [20] and used in clauses 5, 6, 7 and 8 of the present document. The presence of other keynames defined in clause 3.10 of TOSCA-Simple-Profile-YAML-v1.3 [20] shall not cause the NFV descriptor to be rejected and the consumer of the NFV descriptor may choose to ignore them and their associated contents, unless they are part of the type definitions files referred to in Annex B of the present document.

5.2 Network Service Descriptor

The Network Service Descriptor (NSD) is a deployment template which consists of information used by the NFVO for lifecycle management of an NS as defined in ETSI GS NFV-IFA 014 [2]. The NSD:

- References zero, one or more Virtualised Network Function Descriptors (VNFD).
- References zero, one or more Physical Network Functions Descriptors (PNFD).
- References zero, one or more nested NSD.
- Includes zero, one or more Virtual Link Descriptors (VLD).
- Includes zero, one or more VNF Forwarding Graph Descriptors (VNFFGD).

A VNFFGD describes a topology of the Network Service or a portion of the Network Service.

A VLD describes the resource for deploying and managing the lifecycle of virtual links between the constituents of an NS.

A PNFD describes the connectivity requirements to integrate PNFs in an NS.

A nested NSD is an NSD from which a nested NS can be instantiated within a parent NS instance.

5.3 Virtualised Network Function Descriptor

The VNFD is a component of a VNF package. It is used by both the NFVO and the VNFM.

A VNFD is a deployment template which describes a VNF in terms of deployment and operational behaviour requirements. It also contains Virtualised Deployment Units (VDUs), internal virtual link descriptors, external connection point descriptors, software image descriptors, and deployment flavour descriptors, as defined in ETSI GS NFV-IFA 011 [1].

A VNFD contains the following main pieces of information, as shown in figure 5.3-1:

- Virtualisation Deployment Unit (VDU) is a construct supporting the description of the deployment and operational behaviour of a VNF Component (VNFC). A VNFC instance created based on the VDU maps to a single virtualisation container (e.g. a VM). A VDU describes the resources needed to deploy and manage the lifecycle of a VNFC. A VDU includes internal Connection Point Descriptors (CPDs) that describe internal connection points that can either be used to connect a VNFC to an internal virtual link or be re-exposed outside the VNF as external connection points.
- External CPD: describes an external connection point of a VNF, where either an internal connection point of a VDU is exposed as external connection point or the external connection point is directly connected to an internal virtual link.
- Internal VLD: describes the resource requirements for deploying and managing the lifecycle of virtual links between one or more VNFC instances created based on one or more VDUs.

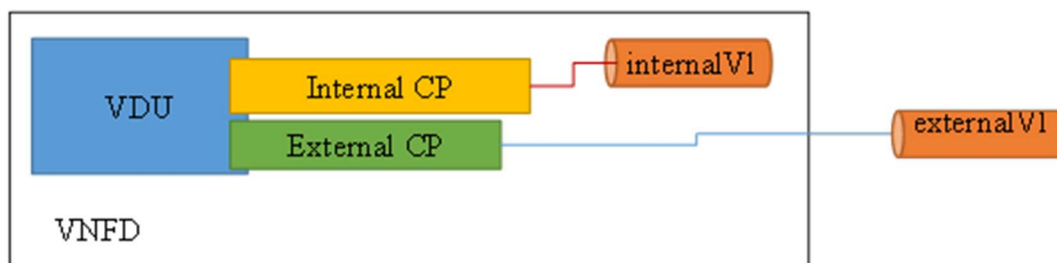


Figure 5.3-1: Overview of VNF descriptor

The information within a VNFD is structured according to one or more VNF deployment flavours (VnfDf) that specify different deployment configuration of a VNF, in terms of its internal topology and resource needs.

5.4 Physical Network Function Descriptor

The Physical Network Function Descriptor (PNFD) information element is a deployment template enabling on-boarding PNFs and referencing them from an NSD. It focuses on connectivity aspects only.

5.5 `tosca_definitions_version` and Namespace prefix

The "tosca_definitions_version" keyword when used in the present document shall comply with the definition as specified in section 3.1.2 of TOSCA-Simple-Profile-YAML-v1.3 [20] with the associated Namespace Alias value defined in the TOSCA-Simple-Profile-YAML-v1.3 [20].

NOTE 1: This implies that service templates complying to the present document can only import service templates that reference the same version - and thus use the same grammar - as the importing service template. This is a restriction compared to the TOSCA-Simple-Profile-YAML-v1.3 specification [20].

NOTE 2: As specified in TOSCA-Simple-Profile-YAML-v1.3 [20], the grammar used in TOSCA-Simple-Profile-YAML-v1.2 [i.18] is still supported with deprecation. The present document indicates the cases in which support with deprecation is applicable.

Table 5.5-1 defines the TOSCA Namespace prefix that shall be used to declare the namespace of all the TOSCA types as specified in the present document.

Table 5.5-1

Namespace Prefix	Specification Description
toscafv	The TOSCA namespace prefix of all the TOSCA types as specified in the present document.

5.6 Imports statement

5.6.1 VNFD TOSCA service template

A VNFD TOSCA service template as specified in clause 6.11 shall include a TOSCA import definition referencing the following files:

- The file defined in clause B.2 that includes all the type definitions from clause 6 of the present document.
- Others, as described in clause 6.11.

As specified in TOSCA-Simple-Profile-YAML-v1.3 [20], the import statement can use a single-line or multi-line grammar.

The single-line grammar supports the import of single or multiple uniquely named VNFD types and other template definition files.

EXAMPLE 1:

```
imports:
- https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_vnfd_types.yaml
- any_other_files.yaml
- custom_vnfd_datatypes_extension.yaml
```

The multi-line grammar also supports the import of single or multiple uniquely named VNFD types and other template definition files. The "file" keyword is a mandatory parameter in this grammar.

EXAMPLE 2:

```
imports:
- file:
https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_vnfd_types.yaml
- file: any_other_files.yaml
- file: custom_vnfd_datatypes_extension.yaml
```

5.6.2 NSD TOSCA service template

An NSD TOSCA service template as specified in clause 7.11 shall include a TOSCA import definition referencing the following files:

- The file defined in clause B.3 that includes all the type definitions from clause 7 of the present document.

- Others, as described in clause 7.11.

As specified in TOSCA-Simple-Profile-YAML-v1.3 [20], the import statement can use a single-line or multi-line grammar.

The single-line grammar supports the import of single or multiple uniquely named NSD types and other template definition files.

EXAMPLE 1:

```
imports:
- https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_nsd_types.yaml
- any_other_files.yaml
- custom_nsd_node_types_extension.yaml
```

The multi-line grammar also supports the import of single or multiple uniquely named NSD types and other template definition files. The "file" keyword is a mandatory parameter in this grammar.

EXAMPLE 2:

```
imports:
- file:
https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_nsd_types.yaml
- file: any_other_files.yaml
- file: custom_nsd_node_types_extension.yaml
```

5.6.3 PNFD TOSCA service template

A PNFD TOSCA service template as specified in clause 8.11 shall include a TOSCA import definition referencing the following files:

- The file defined in clause B.4 that includes all the type definitions from clause 8 of the present document.
- Others, as described in clause 8.11.

As specified in TOSCA-Simple-Profile-YAML-v1.3 [20], the import statement can use a single-line or multi-line grammar.

The single-line grammar supports the import of single or multiple uniquely named PNFD types and other template definition files.

EXAMPLE 1:

```
imports:
- https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_pnfd_types.yaml
- any_other_files.yaml
- custom_pnfd_node_types_extension.yaml
```

The multi-line grammar also supports the import of single or multiple uniquely named PNFD types and other template definition files. The "file" keyword is a mandatory parameter in this grammar.

EXAMPLE 2:

```
imports:
- file:
https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_pnfd_types.yaml
- file: any_other_files.yaml
- file: custom_pnfd_node_types_extension.yaml
```

5.7 Type extension

5.7.1 Introduction

Type extension is used when VNF-specific type information is introduced in the VNFD (e.g. modifiable attributes, configurable properties and additional parameters to LCM operations) or NSD (e.g. additional parameters to LCM operations).

5.7.2 Rules

Type extension may be applied to NFV types defined in the present document within the limits specified in table 5.7.3-1 and table 5.7.4-1, adhering to the following rule.

A derived type shall extend the base type in such a way that it remains substitutable for the base type with the following requirements:

New properties and attributes may be introduced with no restriction within the limits specified in table 5.7.3-1 and table 5.7.4-1.

Existing properties may be extended according to the following rules:

- a) A scalar property shall not be extended to another type (e.g. a string property shall not be replaced with an integer property or with a complex property of the same name).
- b) A complex property of data type "X" may only be extended to a property of type "Y" where "Y" is derived from "X" according to the present rules (recursive rule: present rules applied to each property of the derived data type).
- c) A property of type list with entry schema "X" may only be extended to a list with entry schema "Y" where "Y" is an extension of "X" according to the present rules (recursive rule: present rules applied to the elements of the list).
- d) A property of type map with entry schema "X" may only be extended to a map with entry schema "Y" where "Y" is an extension of "X" according to the present rules (recursive rule: present rules applied to the values of the map).

In general, the above rules apply to introducing/extending other elements beyond properties such as capabilities, requirements, interfaces, operations, inputs, etc. as well.

5.7.3 VNFD Types

Table 5.7.3-1 specifies the extension point where VNFD author may extend the pre-defined types.

Table 5.7.3-1: VNFD type extension points

Type	Keyname	Property name
tosca.nodes.nfv.VNF	properties	modifiable_attributes (as a new property) configurable_properties (as a new property). See note 4.
	requirements	New requirements with capability type VirtualLinkable (as new requirements).
	interfaces	Vnflcm.{operation_name}.inputs.additional_parameters (as a new property) Vnflcm.{operation_name}.inputs (as new properties). VnfIndicator.notifications (as new notifications). See note 4.
	attributes	One attribute of primitive type per VNF indicator may be added. One attribute of type integer per scaling aspect may be added. It holds the value of the current scale level.

Type	Keyname	Property name
tosca.nodes.nfv.Vdu.Compute	properties	configurable_properties (as a new property). See note 1 and note 4.
tosca.datatypes.nfv.VnfInfoModifiableAttributes	properties	extensions (as a new property) metadata (as a new property). See notes 2, 4 and 5.
tosca.datatypes.nfv.VnfConfigurableProperties	properties	additional_configurable_properties (as a new property). See notes 2, 4 and 5.
tosca.datatypes.nfv.VnfcConfigurableProperties	properties	additional_vnfc_configurable_properties (as a new property). See notes 2, 4 and 5.
tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions	properties	(new properties). See notes 2, 4 and 5.
tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata	properties	(new properties). See notes 2, 4 and 5.
tosca.datatypes.nfv.VnfAdditionalConfigurableProperties	properties	(new properties). See notes 2, 4 and 5.
tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties	properties	(new properties). See notes 2, 4 and 5.
tosca.datatypes.nfv.VnfOperationAdditionalParameters	properties	(new properties). See notes 2, 4 and 5.
tosca.datatypes.nfv.InputOpCoordParams	properties	(new properties). See notes 2, 4 and 5.
tosca.datatypes.nfv.OutputOpCoordParams	properties	(new properties). See notes 2, 4 and 5.
tosca.interfaces.nfv.VnfIndicator	notifications	one notification may be added per Vnf indicator. See notes 3 and 5.
tosca.interfaces.nfv.ChangeCurrentVnfPackage	operations	{change_current_package_script} (as new operation). {change_current_package_script}.inputs.additional_parameters (as new property). See notes 3 and 5.
tosca.policies.nfv.LcmCoordinationAction	policies	See clause 6.10.16.4.
NOTE 1: VNF specific Vdu.Compute node types should be given names starting by the provider name followed by a dot (".") in order to avoid collisions if these node types are imported in an NSD service template. The provider name may, but need not, be identical to the value of the provider property of the VNF node type. Furthermore, it is the VNF provider's responsibility to ensure the uniqueness of the names of its Vdu.Compute node types, i.e. the Vdu.Compute node type names starting with its provider name. NOTE 2: VNF specific extension datatypes should be given names starting by the provider name followed by a dot (".") in order to avoid collisions when importing these datatypes in an NSD service template. The provider name may, but need not, be identical to the value of the provider property of the VNF node type. Furthermore, it is the VNF provider's responsibility to ensure the uniqueness of the names of its datatypes, i.e. the datatype names starting with its provider name. NOTE 3: VNF specific interface types should be given names starting by the provider name followed by a dot (".") in order to avoid collisions when importing these types in an NSD service template. The provider name may, but need not, be identical to the value of the provider property of the VNF node type. Furthermore, it is the VNF provider's responsibility to ensure the uniqueness of the names of its interface types, i.e. the interface type names starting with its provider name. NOTE 4: If a property is defined with a required value equal to false, the default value shall not be present in VNFD. This also applies to any new datatypes introduced in the VNFD. NOTE 5: If a VNF provider, notwithstanding the provisions in notes 2 and 3, uses the same name for a data or an interface type definition in two or more VNFs because the definitions are identical, the common data or interface type definition should be placed in a separate file which is imported into the file containing the VNF node type definition. Furthermore, the file should contain a namespace definition before the type definitions, using the "namespace" keyword and a value, e.g.: namespace: MyCompany.MyInterface. The value shall be the same in all VNFs. This is in order to avoid a parser conflict if multiple VNFDs that use the same interface type definition are used in the same NSD. Furthermore, the complete content of the file shall be the same in all VNFs. If more than one file with type definitions is provided, each shall define its own namespace, unless it is imported in another file that already defines a namespace. However, this rule does not need to be followed when the two or more VNFDs are versions of the same VNF. Versions of the same VNF are likely to have some type definitions that are common to both, but it is unlikely that they are used in the same NSD. In case two versions of the same VNF that share some type definitions are used in the same NSD, the handling to avoid type conflicts is done in the NSD by following the provisions specified in clauses 7.11.2 and 7.11.3.		

5.7.4 NSD types

Table 5.7.4-1 specifies the extension points where NSD author may extend the pre-defined types.

Table 5.7.4-1: NSD type extension points

Type	Keyname	Property name
tosca.nodes.nfv.NS	requirements	New requirements with capability type VirtualLinkable (as new requirements).
	interfaces	Nslcm.{operation_name}.inputs.additional_parameters (as a new property).
tosca.datatypes.nfv.NsOperationAdditionalParameters	properties	(new properties). See note.
tosca.policies.nfv.NsAutoScale	policies	See clause 7.10.15.
NOTE: If a property is defined with a required value equal to false, the default value shall not be present in NSD. This also applies to any new datatypes introduced in the NSD.		

5.7.5 Security-sensitive properties in extended data types

The definition of the properties of some of the data types derived from the data types specified in the present document may include the following metadata:

sensitive: "true"

NOTE: Double quotes are needed to avoid that the parser interprets it as the Boolean value true.

This metadata indicates that the property holds security-sensitive information (e.g. passwords).

It is out of the scope of the present document to specify the exact behaviour of a functional block handling security-sensitive properties. The intent of this metadata is to signal not to expose the value of the property by means such as user interfaces, logging files, programmatic interfaces, etc.

Specific handling of these properties when they are used as parameters in the APIs is defined in the affected specifications, e.g. ETSI GS NFV-SOL 003 [25].

Extension of the types listed in table 5.7.5-1 may include properties with this metadata.

Table 5.7.5-1: VNFD and NSD extensible data types

Data type
tosca.datatypes.nfv.VnflInfoModifiableAttributesExtensions
tosca.datatypes.nfv.VnfAdditionalConfigurableProperties
tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties
tosca.datatypes.nfv.VnfOperationAdditionalParameters
tosca.datatypes.nfv.NsOperationAdditionalParameters

In this version of the present document the use of the security-sensitive tagging is limited to properties defined in the extended types. The use of the security-sensitive tagging in properties defined in the present document is not supported.

The security-sensitive tagging is foreseen for properties whose values are expected to be dynamically set from the APIs. Tagging a property that has a value assigned in the VNFD as security-sensitive does not prevent its exposure when the complete VNFD is exposed. Therefore, assigning a value in the VNFD should be avoided.

5.8 Non-Backward Compatible changes

Annex F provides the list of non-backward compatible changes during the development of the present document.

5.9 Use of TOSCA functions

The TOSCA service templates complying with the present document may use the TOSCA functions listed in table 5.9-1. Use of these TOSCA functions shall comply with the provisions in section 4 of TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 5.9-1: Supported TOSCA functions

TOSCA function
get_property
get_artifact
get_input (see note 1)
get_attribute (see note 2)
TOSCA intrinsic functions (see note 3)
NOTE 1: The get_input function is used to retrieve the values of parameters declared in the input section of a service template and assign them properties. Service templates complying with the present document may only use the get_input function to assigning values to the properties listed in table 5.9-2.
NOTE 2: Service templates complying with the present document may only use the get_attribute function to retrieve the value of the following attributes: a) In a VNF node template: scale_status attribute. b) In an NS node template: scale_status attribute and any VNF node attribute holding the value of a VNF indicator.
NOTE 3: TOSCA intrinsic functions (concat, join and token) are defined in section 4.3 of TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 5.9-2: Applicable properties for get_input

Type	Property
tosca.nodes.nfv.VNF	flavour_id modifiable_attributes configurable_properties
tosca.nodes.nfv.Vdu.Compute	configurable_properties
NOTE: Input values are either assigned in the service templates or received from the APIs. The mapping between TOSCA properties and API attributes is described in Annex D of the present document.	

6 VNFD TOSCA model

6.1 Introduction

The VNFD information model specified by ETSI GS NFV-IFA 011 [1] is mapped to the TOSCA concepts. The VNFD is represented as one or more TOSCA service templates to be used by the VNFM for deploying and managing the lifecycle of a VNF instance.

Table 6.1-1 describes the mapping of the main information elements defined in ETSI GS NFV-IFA 011 [1] applicable to a VNFD and the corresponding NFV-specific TOSCA Types, as well the basic TOSCA types defined in TOSCA-Simple-Profile-YAML-v1.3 [20] from which they are derived from. The full definition of all types can be found in the following clauses.

NOTE 1: The autoScale rule with use of VNF monitoring parameters specified in ETSI GS NFV-IFA 011 [1] is not supported in this version of the present document.

NOTE 2: The monitoring parameters for Oscontainer based VNF specified in ETSI GS NFV-IFA 011 [1] is not supported in this version of the present document.

NOTE 3: The VirtualCpd for VM-based VNF as specified in ETSI GS NFV-IFA 011 [1] is not supported in this version of the present document.

NOTE 4: The monitoringType specified in ETSI GS NFV-IFA 011 [1] is not supported in this version of the present document.

Table 6.1-1: Mapping of ETSI GS NFV-IFA 011 [1] information elements with TOSCA types

ETSI GS NFV-IFA 011 [1] Elements	VNFD TOSCA types	Derived from
VNFD	tosca.nodes.nfv.VNF	tosca.nodes.Root
Vdu	n/a (see note 1)	n/a
Cpd (Connection Point)	tosca.nodes.nfv.Cp	tosca.nodes.Root
VduCpd (internal connection point)	tosca.nodes.nfv.VduCp	tosca.nodes.nfv.Cp
VirtualCpd	tosca.nodes.nfv.VirtualCp	tosca.nodes.nfv.Cp
VipCpd	tosca.nodes.nfv.VipCp	tosca.nodes.nfv.Cp
VnfVirtualLinkDesc (Virtual Link)	tosca.nodes.nfv.VnfVirtualLink	tosca.nodes.Root
VnfExtCpd (External Connection Point)	tosca.nodes.nfv.VnfExtCp tosca.nodes.nfv.VduCp	tosca.nodes.nfv.Cp
VirtualStorageDesc	tosca.nodes.nfv.Vdu.VirtualBlockStorage tosca.nodes.nfv.Vdu.VirtualObjectStorage tosca.nodes.nfv.Vdu.VirtualFileStorage	tosca.nodes.Root
VirtualComputeDesc	tosca.nodes.nfv.Vdu.Compute	tosca.nodes.Root
OsContainerDesc	tosca.nodes.nfv.Vdu.OsContainer	tosca.nodes.Root
SwImageDesc	tosca.artifacts.nfv.SwImage	tosca.artifacts.Deployment .Image
CertificateDesc	tosca.nodes.nfv.Certificate (see note 4)	tosca.nodes.Root
VnfDf	n/a (see note 2)	n/a
SecurityGroupRule	tosca.policies.nfv.SecurityGroupRule	tosca.policies.Root
VnfConfigurableProperties	tosca.datatypes.nfv.VnfConfigurableProperties	tosca.datatypes.Root
VnfInfoModifiableAttributes	tosca.datatypes.nfv.VnfInfoModifiableAttributes	tosca.datatypes.Root
n/a (see note 3)	tosca.nodes.nfv.Mciop	tosca.nodes.Root
MciopProfile	n/a (see note 3)	n/a
DeployableModule	tosca.nodes.nfv.DeployableModule	tosca.nodes.Root
PaasServiceProfile	tosca.nodes.nfv.PaasServiceProfile	tosca.nodes.Root
PaasServiceRequest	tosca.nodes.nfv.PaasServiceRequest	tosca.nodes.Root
NOTE 1: The Vdu information element is represented as a collection of toscanodes.nfv.VduCp, toscanodes.nfv.Vdu.Compute (for VM based VNFs), toscanodes.nfv.Vdu.OsContainerDeployableUnit (for Oscontainer based VNFs), toscanodes.nfv.Vdu.OsContainer (for Oscontainer based VNFs), toscanodes.nfv.Vdu.VirtualBlockStorage, toscanodes.nfv.Vdu.VirtualObjectStorage and toscanodes.nfv.Vdu.VirtualFileStorage types. NOTE 2: The VnfDf information element is represented as a TOSCA service template. NOTE 3: There is not a direct mapping between the MciopProfile in ETSI GS NFV-IFA 011 [1] and the toscanodes.nfv.Mciop. However, the 'deploymentOrder' and 'associatedVdu' attributes defined in the MciopProfile element are mapped to toscanodes.nfv.Mciop. The 'affinityOrAntiAffinityGroupID' is mapped to toscapolicies.nfv.AffinityRule or toscapolicies.nfv.AntiAffinityRule. NOTE 4: The use of certificate management in direct-mode does not require any information in VNFD.		

Figure 6.1-1 provides an overview of the TOSCA node types used to build a service template representing a VNFD for a specific deployment flavour, and of the relationship between them for VNFs when all its virtualisation containers are realized as VMs. The figure shows one of the three types of virtual storage. A detailed description is provided in clause 6.11.

6.2 Data Types

6.2.1 `tosca.datatypes.nfv.CpProtocolData`

6.2.1.1 Description

The `CpProtocolData` data type is defined in clause 9.2.6 of the present document.

6.2.2 `tosca.datatypes.nfv.AddressData`

6.2.2.1 Description

The `AddressData` data type is defined in clause 9.2.3 of the present document.

6.2.3 `tosca.datatypes.nfv.L2AddressData`

6.2.3.1 Description

The `L2AddressData` data type is defined in clause 9.2.1 of the present document.

6.2.4 `tosca.datatypes.nfv.VirtualNetworkInterfaceRequirements`

6.2.4.1 Description

The `VirtualNetworkInterfaceRequirements` data type describes requirements on a virtual network interface, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.4.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.4.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VirtualNetworkInterfaceRequirements</code>
Type Qualified Name	<code>toscanfv:VirtualNetworkInterfaceRequirements</code>
Type URI	<code>tosca.datatypes.nfv.VirtualNetworkInterfaceRequirements</code>

6.2.4.2 Properties

The properties of the `VirtualNetworkInterfaceRequirements` data type shall comply with the provisions set out in table 6.2.4.2-1.

Table 6.2.4.2-1: Properties

Name	Required	Type	Constraints	Description
<code>name</code>	no	string		Provides a human readable name for the requirement.
<code>description</code>	no	string		Provides a human readable description of the requirement.
<code>network_interface_requirements</code>	yes	map of string		The network interface requirements. A map of strings that contain a set of key-value pairs that describes the hardware platform specific network interface deployment requirements. More information regarding the usage of this property is available at: https://register.etsi.org.

Name	Required	Type	Constraints	Description
nic_io_requirements	no	tosca.datatypes.nfv.LogicalNodeData		This references (couples) the CP with any logical node I/O requirements (for network devices) that may have been created. Linking these attributes is necessary so that so that I/O requirements that need to be articulated at the logical node level can be associated with the network interface requirements associated with the CP.

6.2.4.3 Definition

The syntax of the VirtualNetworkInterfaceRequirements data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualNetworkInterfaceRequirements:
  derived_from: toska.datatypes.Root
  description: Describes requirements on a virtual network interface
  properties:
    name:
      type: string
      description: Provides a human readable name for the requirement.
      required: false
    description:
      type: string
      description: Provides a human readable description of the requirement.
      required: false
    network_interface_requirements:
      type: map
      description: The network interface requirements. A map of strings that contain
a set of key-value pairs that describes the hardware platform specific network
interface deployment requirements.
      required: true
      entry_schema:
        type: string
    nic_io_requirements:
      type: toska.datatypes.nfv.LogicalNodeData
      description: references (couples) the CP with any logical node I/O
requirements (for network devices) that may have been created. Linking these
attributes is necessary so that so that I/O requirements that need to be articulated
at the logical node level can be associated with the network interface requirements
associated with the CP.
      required: false

```

6.2.4.4 Examples

None.

6.2.4.5 Additional Requirements

None.

6.2.5 toska.datatypes.nfv.L3AddressData

6.2.5.1 Description

The L3AddressData data type is defined in clause 9.2.2 of the present document.

6.2.6 `tosca.datatypes.nfv.RequestedAdditionalCapability`

6.2.6.1 Description

The RequestedAdditionalCapability data type describes requested additional capability for a particular VDU, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.6.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.6.1-1: Type name, shorthand, and URI

Shorthand Name	RequestedAdditionalCapability
Type Qualified Name	toscanfv:RequestedAdditionalCapability
Type URI	tosca.datatypes.nfv.RequestedAdditionalCapability

6.2.6.2 Properties

The properties of the RequestedAdditionalCapability data type shall comply with the provisions set out in table 6.2.6.2-1.

Table 6.2.6.2-1: Properties

Name	Required	Type	Constraints	Description
requested_additional_capability_name	yes	string		Identifies a requested additional capability for the VDU.
support_mandatory	yes	boolean		Indicates whether the requested additional capability is mandatory for successful operation.
min_requested_additional_capability_version	no	string		Identifies the minimum version of the requested additional capability.
preferred_requested_additional_capability_version	no	string		Identifies the preferred version of the requested additional capability.
target_performance_parameters	yes	map of string		Identifies specific attributes, dependent on the requested additional capability type.

6.2.6.3 Definition

The syntax of the RequestedAdditionalCapability data type shall comply with the following definition:

```

tosca.datatypes.nfv.RequestedAdditionalCapability:
  derived_from: toska.datatypes.Root
  description: describes requested additional capability for a particular VDU
  properties:
    requested_additional_capability_name:
      type: string
      description: Identifies a requested additional capability for the VDU.
      required: true
    support_mandatory:
      type: boolean
      description: Indicates whether the requested additional capability is
mandatory for successful operation.
      required: true
    min_requested_additional_capability_version:
      type: string
      description: Identifies the minimum version of the requested additional
capability.
      required: false
    preferred_requested_additional_capability_version:
      type: string
      description: Identifies the preferred version of the requested additional
capability.
      required: false
    target_performance_parameters:
      type: map
      description: Identifies specific attributes, dependent on the requested
additional capability type.
      required: true
    entry_schema:
      type: string

```

6.2.6.4 Examples

None.

6.2.6.5 Additional Requirements

None.

6.2.7 toska.datatypes.nfv.VirtualMemory

6.2.7.1 Description

The VirtualMemory data type supports the specification of requirements related to virtual memory of a virtual compute resource, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.7.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.7.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualMemory
Type Qualified Name	toscanfv:VirtualMemory
Type URI	tosca.datatypes.nfv.VirtualMemory

6.2.7.2 Properties

The properties of the VirtualMemory data type shall comply with the provisions set out in table 6.2.7.2-1.

Table 6.2.7.2-1: Properties

Name	Required	Type	Constraints	Description
virtual_mem_size	yes	scalar-unit.size		Amount of virtual memory.
virtual_mem_size_valid_values	no	tosca.datatypes.nfv.SizeRange		Indicates valid values for the amount of virtual memory. If this property is present the amount of virtual memory can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the virtual compute resource is instantiated with the value indicated in the virtual_mem_size property. If this property is not present the amount of virtual memory is not configurable via the VNF LCM interface and is always equal to the value indicated in the virtual_mem_size property.
virtual_mem_oversubscription_policy	no	string		The memory core oversubscription policy in terms of virtual memory to physical memory on the platform.
vdu_mem_requirements	no	map of string		The hardware platform specific VDU memory requirements. A map of strings that contains a set of key-value pairs that describes hardware platform specific VDU memory requirements. More information regarding the usage of this property is available at: https://register.etsi.org.
numa_enabled	yes	boolean	default: false	It specifies the memory allocation to be cognisant of the relevant process/core allocation.
huge_pages_requirements	no	list of tosca.datatypes.nfv.Hugepages		The requirement for huge pages resources. Each element in the list indicates a hugepage size and the total memory requested for hugepages of that size. Each element may also optionally indicate a range of valid values for the total memory requested for that page size if the amount required can be configurable via the VNF LCM interface.

6.2.7.3 Definitions

The syntax of the VirtualMemory data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualMemory:
  derived_from: tosca.datatypes.Root
  description: supports the specification of requirements related to virtual memory
of a virtual compute resource
  properties:
    virtual_mem_size:
      type: scalar-unit.size
      description: Amount of virtual memory.
      required: true
    virtual_mem_size_valid_values:
      type: tosca.datatypes.nfv.SizeRange
      description: Indicates valid values for the amount of virtual memory. If this
property is present the amount of virtual memory can be indicated in a VNF LCM
operation. If no value is indicated in the VNF LCM operation, the virtual compute

```

resource is instantiated with the value indicated in the `virtual_mem_size` property. If this property is not present the amount of virtual memory is not configurable via the VNF LCM interface and is always equal to the value indicated in the `virtual_mem_size` property.

```

    required: false
    virtual_mem_oversubscription_policy:
      type: string
      description: The memory core oversubscription policy in terms of virtual
memory to physical memory on the platform.
      required: false
    vdu_mem_requirements:
      type: map
      description: The hardware platform specific VDU memory requirements. A map of
strings that contains a set of key-value pairs that describes hardware platform
specific VDU memory requirements.
      required: false
      entry_schema:
        type: string
    numa_enabled:
      type: boolean
      description: It specifies the memory allocation to be cognisant of the
relevant process/core allocation.
      required: true
      default: false
    huge_pages_requirements:
      type: list
      description: The requirement for huge pages resources. Each element in the
list indicates a hugepage size and the total memory requested for hugepages of that
size. Each element may also optionally indicate a range of valid values for the total
memory requested for that page size if the amount required can be configurable via the
VNF LCM interface.
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.Hugepages

```

6.2.7.4 Examples

None.

6.2.7.5 Additional Requirements

None.

6.2.8 `tosca.datatypes.nfv.VirtualCpu`

6.2.8.1 Description

The `VirtualCpu` data type supports the specification of requirements related to virtual CPU(s) of a virtual compute resource, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.8.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.8.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VirtualCpu</code>
Type Qualified Name	<code>toscanfv:VirtualCpu</code>
Type URI	<code>tosca.datatypes.nfv.VirtualCpu</code>

6.2.8.2 Properties

The properties of the VirtualCpu data type shall comply with the provisions set out in table 6.2.8.2-1.

Table 6.2.8.2-1: Properties

Name	Required	Type	Constraints	Description
cpu_architecture	no	string		CPU architecture type. Examples are x86, ARM.
num_virtual_cpu	yes	integer	greater_than: 0	Number of virtual CPUs.
num_virtual_cpu_valid_values	no	tosca.datatypes.nfv.IntegerRange		Indicates valid values for the number of virtual CPUs. If this property is present the number of virtual CPUs can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the virtual compute resource is instantiated with the value indicated in the num_virtual_cpu property. If this property is not present the number of virtual CPUs is not configurable via the VNF LCM interface and is always equal to the value indicated in the num_virtual_cpu property.
virtual_cpu_clock	no	scalar-unit.frequency		Minimum virtual CPU clock rate.
virtual_cpu_oversubscription_policy	no	string		CPU core oversubscription policy e.g. the relation of virtual CPU cores to physical CPU cores/threads.
vdu_cpu_requirements	no	map of string		The hardware platform specific VDU CPU requirements. A map of strings that contains a set of key-value pairs describing VDU CPU specific hardware platform requirements. More information regarding the usage of this property is available at: https://register.etsi.org .
virtual_cpu_pinning	no	tosca.datatypes.nfv.VirtualCpuPinning		The virtual CPU pinning configuration for the virtualised compute resource.
virtual_cpu_power_state_requirements	no	tosca.datatypes.nfv.CpuPowerState		Requirements for virtual CPU power state configuration for the virtualised compute resource.

6.2.8.3 Definition

The syntax of the VirtualCpu data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualCpu:
  derived_from: tosc.datatypes.Root
  description: Supports the specification of requirements related to virtual CPU(s)
of a virtual compute resource
  properties:
    cpu_architecture:
      type: string
      description: CPU architecture type. Examples are x86, ARM
      required: false
    num_virtual_cpu:
      type: integer
      description: Number of virtual CPUs
      required: true

```

```

    constraints:
      - greater_than: 0
    num_virtual_cpu_valid_values:
      type: toska.datatypes.nfv.IntegerRange
      description: Indicates valid values for the number of virtual CPUs. If this
property is present the number of virtual CPUs can be indicated in a VNF LCM
operation. If no value is indicated in the VNF LCM operation, the virtual compute
resource is instantiated with the value indicated in the num_virtual_cpu property. If
this property is not present the number of virtual CPUs is not configurable via the
VNF LCM interface and is always equal to the value indicated in the num_virtual_cpu
property.
      required: false
    virtual_cpu_clock:
      type: scalar-unit.frequency
      description: Minimum virtual CPU clock rate
      required: false
    virtual_cpu_oversubscription_policy:
      type: string
      description: CPU core oversubscription policy e.g. the relation of virtual CPU
cores to physical CPU cores/threads.
      required: false
    vdu_cpu_requirements:
      type: map
      description: The hardware platform specific VDU CPU requirements. A map of
strings that contains a set of key-value pairs describing VDU CPU specific hardware
platform requirements.
      required: false
      entry_schema:
        type: string
    virtual_cpu_pinning:
      type: toska.datatypes.nfv.VirtualCpuPinning
      description: The virtual CPU pinning configuration for the virtualised compute
resource.
      required: false
    virtual_cpu_power_state_requirements:
      type: toska.datatypes.nfv.CpuPowerState
      description: Requirements for virtual CPU power state configuration for the
virtualised compute resource.
      required: false

```

6.2.8.4 Examples

None.

6.2.8.5 Additional Requirements

None.

6.2.9 toska.datatypes.nfv.VirtualCpuPinning

6.2.9.1 Description

The VirtualCpuPinning data type supports the specification of requirements related to the virtual CPU pinning configuration of a virtual compute resource, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.9.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.9.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualCpuPinning
Type Qualified Name	toscanfv:VirtualCpuPinning
Type URI	tosca.datatypes.nfv.VirtualCpuPinning

6.2.9.2 Properties

The properties of the VirtualCpuPinning data type shall comply with the provisions set out in table 6.2.9.2-1.

Table 6.2.9.2-1: Properties

Name	Required	Type	Constraints	Description
virtual_cpu_pinning_policy	no	string	Valid values: See YAML definition constraints	Indicates the policy for CPU pinning. The policy can take values of "static" or "dynamic". In case of "dynamic" the allocation of virtual CPU cores to logical CPU cores is decided by the VIM when the virtualisation container is realized as a VM or by the CISM if it is realized as one or more OS containers (e.g. SMT (Simultaneous Multi-Threading) requirements). In case of "static" the allocation is requested to be according to the <code>virtual_cpu_pinning_rule</code> .
virtual_cpu_pinning_rule	no	list of string		Provides the list of rules for allocating virtual CPU cores to logical CPU cores/threads.

6.2.9.3 Definition

The syntax of the VirtualCpuPinning data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualCpuPinning:
  derived_from: toscanfv.Root
  description: Supports the specification of requirements related to the virtual CPU
pinning configuration of a virtual compute resource
  properties:
    virtual_cpu_pinning_policy:
      type: string
      description: Indicates the policy for CPU pinning. The policy can take values
of "static" or "dynamic". In case of "dynamic" the allocation of virtual CPU cores to
logical CPU cores is decided by the VIM. (e.g. SMT (Simultaneous Multi-Threading)
requirements). In case of "static" the allocation is requested to be according to the
virtual_cpu_pinning_rule.
      required: false
      constraints:
        - valid_values: [ static, dynamic ]
    virtual_cpu_pinning_rule:
      type: list
      description: Provides the list of rules for allocating virtual CPU cores to
logical CPU cores/threads
      required: false
      entry_schema:
        type: string

```

6.2.9.4 Examples

None.

6.2.9.5 Additional Requirements

The `virtual_cpu_pinning_rule` shall be included if the `virtual_cpu_pinning_policy` property is set to "static" and shall be absent otherwise.

6.2.10 `tosca.datatypes.nfv.VnfcConfigurableProperties`

6.2.10.1 Description

The `VnfcConfigurableProperties` data type defines the configurable properties of a VNFC, as defined in ETSI GS NFV-IFA 011 [1].

Table 6.2.10.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.10.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfcConfigurableProperties</code>
Type Qualified Name	<code>toscanfv:VnfcConfigurableProperties</code>
Type URI	<code>tosca.datatypes.nfv.VnfcConfigurableProperties</code>

6.2.10.2 Properties

The properties of the `VnfcconfigurableProperties` shall comply with the provisions set out in table 6.2.10.2-1.

Table 6.2.10.2-1: Properties

Name	Required	Type	Constraints	Description
<code>additional_vnfc_configurable_properties</code>	no	<code>tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties</code>		Describes additional configuration for VNFC that can be modified using the <code>ModifyVnfInfo</code> operation.

6.2.10.3 Definition

The syntax of the `VnfcConfigurableProperties` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfcConfigurableProperties:
  derived_from: toska.datatypes.Root
  description: Defines the configurable properties of a VNFC
  properties:
    additional_vnfc_configurable_properties:
      type: toska.datatypes.nfv.VnfcAdditionalConfigurableProperties
      description: Describes additional configuration for VNFC that can be modified
using the ModifyVnfInfo operation
      required: false
      # derived types are expected to introduce
      # additional_vnfc_configurable_properties with its type derived from
      # toska.datatypes.nfv.VnfcAdditionalConfigurableProperties
```

6.2.10.4 Examples

Example definition of configurable properties without properties assignment value.

```
tosca_definitions_version: tosca_simple_yaml_1_3

node_types:
  MyCompany.nodes.nfv.Vdu.Aux:
    derived_from: tosca.nodes.nfv.Vdu.Compute
    properties:
      configurable_properties:
        type: MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties

data_types:
  MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcConfigurableProperties
    properties:
      additional_vnfc_configurable_properties:
        type: MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties
        required: true

  MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties
    properties:
      name_prefix_in_vim:
        type: string
        required: false
      dns_server:
        type: string
        required: true

topology_template:
  ..

  node_templates:
    aux:
      type: MyCompany.nodes.nfv.Vdu.Aux
      properties:
        ..
```

Example definition of configurable properties with properties assignment value.

```
tosca_definitions_version: tosca_simple_yaml_1_3

...

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      ...
    interfaces:
      Vnflcm:
        type: tosca.interfaces.nfv.Vnflcm

  MyCompany.nodes.nfv.Vdu.Aux:
    derived_from: tosca.nodes.nfv.Vdu.Compute
    properties:
      configurable_properties:
        type: MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties
```

```

        required: false

data_types:
  MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties:
    derived_from: toska.datatypes.nfv.VnfcConfigurableProperties
    properties:
      additional_vnfc_configurable_properties:
        type: MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties
        required: true

  MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties:
    derived_from: toska.datatypes.nfv.VnfcAdditionalConfigurableProperties
    properties:
      name_prefix_in_vim:
        type: string
        required: true
        default: "MyCustomer"
      dns_server:
        type: string
        required: true
        default: "90.200.250.57"

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    requirements:
      virtual_link: [ dbBackendIpv4, virtual_link ] # IPv4 for SQL

inputs:
  name_prefix_in_vim:
    type: string
  dns_server:
    type: string

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0

  dbBackend:
    type: MyCompany.nodes.nfv.Vdu.Aux
    properties:
      ...
      configurable_properties:
        additional_vnfc_configurable_properties:
          name_prefix_in_vim: { get_input: name_prefix_in_vim }
          dns_server: { get_input: dns_server }

```

In the above example, default values are provided in the node type definition, properties assignment by using TOSCA get_input function is described in the node template. The properties values from the API will override the default values.

6.2.10.5 Additional Requirements

None.

6.2.11 `tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties`

6.2.11.1 Description

The `VnfcAdditionalConfigurableProperties` type is an empty base type for deriving data types for describing additional configurable properties for a given VNFC. Table 6.2.11.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.11.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfcAdditionalConfigurableProperties</code>
Type Qualified Name	<code>toscanfv:VnfcAdditionalConfigurableProperties</code>
Type URI	<code>tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties</code>

6.2.11.2 Properties

None.

6.2.11.3 Definition

The syntax of the `VnfcAdditionalConfigurableProperties` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties:
  derived_from: toska.datatypes.Root
  description: VnfcAdditionalConfigurableProperties type is an empty base type
for deriving data types for describing additional configurable properties for a
given VNFC.
```

6.2.11.4 Examples

See clause 6.2.10.4.

6.2.12 `tosca.datatypes.nfv.VduProfile`

6.2.12.1 Description

The `VduProfile` data type describes additional instantiation data for a given `Vdu.Compute` (for VM based VDU) or `Vdu.OsContainerDeployableUnit` (for Oscontainer based VDU) used in a specific deployment flavour. Table 6.2.12.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.12.1-1: Type name, shorthand and URI

Shorthand Name	<code>VduProfile</code>
Type Qualified Name	<code>toscanfv:VduProfile</code>
Type URI	<code>tosca.datatypes.nfv.VduProfile</code>

6.2.12.2 Properties

The properties of the `VduProfile` data type shall comply with the provisions set out in table 6.2.12.2-1.

Table 6.2.12.2-1: Properties

Name	Required	Type	Constraints	Description
min_number_of_instances	yes	integer	greater_or_equal: 0	Minimum number of instances of the VNFC based on this Vdu.Compute (for VM based VDU) or Vdu.OsContainerDeployableUnit node (for Oscontainer based VDU) that is permitted to exist for a particular VNF deployment flavour.
max_number_of_instances	yes	integer	greater_than: 0	Maximum number of instances of the VNFC based on this Vdu.Compute (for VM based VDU) or Vdu.OsContainerDeployableUnit node (for Oscontainer based VDU) that is permitted to exist for a particular VNF deployment flavour.
nfvi_maintenance_info	no	tosca.datatypes.nfv.NfviMaintenanceInfo		Provides information on the impact tolerance and rules to be observed when instance(s) of the Vdu.Compute (for VM based VDU) are impacted during NFVI operation and maintenance (e.g. NFVI resource upgrades). See notes 2 and 3.
modify_capacity_attributes_op	no	list of String	Valid values: See YAML definition constraints	Indicates in which VNF LCM operations the change of values in capacity related attributes is supported for VNFCs created from this VDU. When change VNF DF or change current VNF package is indicated, it refers to change of DF or VNF package, respectively, to the one where the attribute is indicated.
NOTE 1: A vduld property, which exists in ETSI GS NFV-IFA 011 [1] is not needed, as the VduProfile is contained in the Vdu.Compute node (for VM based VDU) or Vdu.OsContainerDeployableUnit node (for Oscontainer based VDU). NOTE 2: The impact tolerance and rules also apply to the VirtualBlockStorage, VirtualObjectStorage and VirtualFileStorage nodes connected to the Vdu.Compute (for VM based VDU) via one particular occurrence of the virtual_storage requirement and to each VduCp node connected to the Vdu.Compute (for VM based VDU) via one particular occurrence of the virtual_binding capability. NOTE 3: An NFVI level operation (e.g. restart of a virtual machine) can impact a VNF and the VNF may be able to tolerate only a limited number of such impacts simultaneously. The nfvi_maintenance_info provides constraints related to detection and tolerance so that negative impact on VNF functionality can be avoided during NFVI maintenance operations.				

6.2.12.3 Definition

The syntax of the VduProfile data type shall comply with the following definition:

```

tosca.datatypes.nfv.VduProfile:
  derived_from: tosca.datatypes.Root
  description: describes additional instantiation data for a given Vdu.Compute (for
VM based VDU) or Vdu.OsContainerDeployableUnit node (for Oscontainer based VDU) used
in a specific deployment flavour.
  properties:
    min_number_of_instances:
      type: integer
      description: Minimum number of instances of the VNFC based on this Vdu.Compute
(for VM based VDU) or Vdu.OsContainerDeployableUnit node (for Oscontainer based VDU)
that is permitted to exist for a particular VNF deployment flavour.
      required: true
      constraints:
        - greater_or_equal: 0
    max_number_of_instances:
      type: integer
      description: Maximum number of instances of the VNFC based on this Vdu.Compute
(for VM based VDU) or Vdu.OsContainerDeployableUnit node (for Oscontainer based VDU)
that is permitted to exist for a particular VNF deployment flavour.
      required: true
      constraints:

```



```
- greater_than: 0
nfvi_maintenance_info:
  type: tosa.datatypes.nfv.NfviMaintenanceInfo
  description: Provides information on the impact tolerance and rules to be
observed when instance(s) of the Vdu.Compute (for VM based VDU) are impacted during
NFVI operation and maintenance (e.g. NFVI resource upgrades).
  required: false
  modify_capacity_attributes_op:
    type: list
    description: Indicates in which VNF LCM operations the change of values in
capacity related attributes is supported for VNFCs created from this VDU. When
CHANGE_VNF_DF or CHANGE_CURRENT_VNF_PACKAGE is indicated, it refers to change of DF or
VNF package, respectively, to the one where the attribute is indicated.
    required: false
    entry_schema:
      type: string
      constraints:
        - valid_values: [ CHANGE_VNF_DF, CHANGE_CURRENT_VNF_PACKAGE, SCALE_VNF,
SELECT_DEPL_MODS ]
```

6.2.12.4 Examples

```
tosca_definitions_version: tosca_simple_yaml_1_3

topology template:
...

node_templates:
  VDU_A:
    type: tosca.nodes.nfv.Vdu.Compute
    properties:
      vdu_profile:
        min_number_of_instances: 2
        max_number_of_instances: 6
      # other properties omitted for brevity
    requirements:
      - virtual_storage: VirtualStorage_A1
      - virtual_storage: VirtualStorage_A2
    capabilities:
      virtual_binding:

  VirtualStorage_A1:
    type: tosca.nodes.nfv.Vdu.VirtualBlockStorage
    properties:
      # omitted for brevity
      # per_vnfc_instance property not present or set to true
    capabilities:
      virtual_storage

  VirtualStorage_A2:
    type: tosca.nodes.nfv.Vdu.VirtualBlockStorage
    properties:
      # omitted for brevity
      # per_vnfc_instance property not present or set to true
    capabilities:
      virtual_storage

  VduCp_A1:
    type: tosca.nodes.nfv.VduCp
    properties:
      # omitted for brevity
    requirements:
      - virtual_binding: VDU_A
      - virtual_link

  VduCp_A2:
    type: tosca.nodes.nfv.VduCp
    properties:
      # omitted for brevity
    requirements:
      - virtual_binding: VDU_A
      - virtual_link
```

Above snippet shows part of a topology template. The VDU_A node template is a Vdu.Compute node that is connected to two VirtualBlockStorage nodes: VirtualStorage_A1 and VirtualStorage_A2. It also has two VduCps: VduCp_A1 and VduCp_A2.

The minimum number of instances of VDU_A that are permitted to exist is 2. Likewise, the minimum number of instances of VirtualStorage_A1, VirtualStorage_A2, VduCp_A1 and VduCp_A2 that are permitted to exist is 2.

The maximum number of instances of VDU_A that are permitted to exist is 6. Likewise, the maximum number of instances of VirtualStorage_A1, VirtualStorage_A2, VduCp_A1 and VduCp_A2 that are permitted to exist is 6.

6.2.12.5 Additional requirements

The properties of the vdu_profile indicate the maximum and minimum number of Vdu.Compute instances that are permitted to exist, created from a given Vdu.Compute node template during its lifecycle, as well as:

- If the 'per_vnfc_instance' property of the VirtualBlockStorage, VirtualObjectStorage or VirtualFileStorage nodes connected to the Vdu.Compute node is set to 'true' or absent: the maximum and minimum number of instances of each VirtualBlockStorage, VirtualObjectStorage and VirtualFileStorage nodes connected to the Vdu.Compute via one particular occurrence of the virtual_storage requirement.
- If 'per_vnfc_instance' property is set to 'false' only one instance of the storage node shall exist.
- The maximum and minimum number instances of each VduCp node connected to the Vdu.Compute via one particular occurrence of the virtual_binding capability.

6.2.13 tosa.datatypes.nfv.VIProfile

6.2.13.1 Description

The VIProfile data type describes additional instantiation data for a given VL used in a specific VNF deployment flavour. Table 6.2.13.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.13.1-1: Type name, shorthand, and URI

Shorthand Name	VIProfile
Type Qualified Name	toscanfv:VIProfile
Type URI	tosca.datatypes.nfv.VIProfile

6.2.13.2 Properties

The properties of the VIProfile data type shall comply with the provisions set out in table 6.2.13.2-1.

Table 6.2.13.2-1: Properties

Name	Required	Type	Constraints	Description
max_bitrate_requirements	yes	tosca.datatypes.nfv.LinkBitrateRequirements		Specifies the maximum bitrate requirements for a VL instantiated according to this profile.
min_bitrate_requirements	yes	tosca.datatypes.nfv.LinkBitrateRequirements		Specifies the minimum bitrate requirements for a VL instantiated according to this profile.
qos	no	tosca.datatypes.nfv.Qos		Specifies the QoS requirements of a VL instantiated according to this profile.
virtual_link_protocol_data	no	list of tosca.datatypes.nfv.VirtualLinkProtocolData		Specifies the protocol data for a virtual link. If more than 1 values are present, the order shall be the same as the order of the layer_protocols occurrences in the connectivity_type property of the same VnfVirtualLink node, i.e. the first occurrence of the virtual_link_protocol_data represents the highest layer protocol data, and the last occurrence represents the lowest layer protocol data.
NOTE: A vnfVirtualLinkDescId property, which exists in ETSI GS NFV-IFA 011 [1] is not needed, as the VLProfile is contained in the VL node.				

6.2.13.3 Definition

The syntax of the VLProfile data type shall comply with the following definition:

```

tosca.datatypes.nfv.VLProfile:
  derived_from: tosca.datatypes.Root
  description: Describes additional instantiation data for a given VL used in a
specific VNF deployment flavour.
  properties:
    max_bitrate_requirements:
      type: tosca.datatypes.nfv.LinkBitrateRequirements
      description: Specifies the maximum bitrate requirements for a VL instantiated
according to this profile.
      required: true
    min_bitrate_requirements:
      type: tosca.datatypes.nfv.LinkBitrateRequirements
      description: Specifies the minimum bitrate requirements for a VL instantiated
according to this profile.
      required: true
    qos:
      type: tosca.datatypes.nfv.Qos
      description: Specifies the QoS requirements of a VL instantiated according to
this profile.
      required: false
    virtual_link_protocol_data:
      type: list
      description: Specifies the protocol data for a virtual link.
      required: false
    entry_schema:
      type: tosca.datatypes.nfv.VirtualLinkProtocolData

```

6.2.13.4 Examples

None.

6.2.13.5 Additional Requirements

None.

6.2.14 toska.datatypes.nfv.VirtualLinkProtocolData

6.2.14.1 Description

The VirtualLinkProtocolData data type describes one protocol layer and associated protocol data for a given virtual link used in a specific VNF deployment flavour. Table 6.2.14.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.14.1-1: Type name, shorthand and URI

Shorthand Name	VirtualLinkProtocolData
Type Qualified Name	toscanfv:VirtualLinkProtocolData
Type URI	tosca.datatypes.nfv.VirtualLinkProtocolData

6.2.14.2 Properties

The properties of the VirtualLinkProtocolData data type shall comply with the provisions set out in table 6.2.14.2-1.

Table 6.2.14.2-1: Properties

Name	Required	Type	Constraints	Description
associated_layer_protocol	yes	string	Valid values: See YAML definition constraints	Identifies one of the protocols a virtualLink gives access to (ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire) as specified by the connectivity_type property.
l2_protocol_data	no	tosca.datatypes.nfv.L2ProtocolData		Specifies the L2 protocol data for a virtual link. Shall be present when the associatedLayerProtocol attribute indicates a L2 protocol and shall be absent otherwise.
l3_protocol_data	no	tosca.datatypes.nfv.L3ProtocolData		Specifies the L3 protocol data for this virtual link. Shall be present when the associatedLayerProtocol attribute indicates a L3 protocol and shall be absent otherwise.

6.2.14.3 Definition

The syntax of the VirtualLinkProtocolData data type shall comply with the following definition:

```
tosca.datatypes.nfv.VirtualLinkProtocolData:
  derived_from: toska.datatypes.Root
  description: describes one protocol layer and associated protocol data for a
given virtual link used in a specific VNF deployment flavour
  properties:
    associated_layer_protocol:
      type: string
      description: Identifies one of the protocols a virtualLink gives access
to (ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire) as specified by the
connectivity_type property.
      required: true
      constraints:
        - valid_values: [ ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire ]
    l2_protocol_data:
      type: toska.datatypes.nfv.L2ProtocolData
```

```

    description: Specifies the L2 protocol data for a virtual link. Shall be
    present when the associatedLayerProtocol attribute indicates a L2 protocol and
    shall be absent otherwise.
    required: false
    l3_protocol_data:
      type: toska.datatypes.nfv.L3ProtocolData
      description: Specifies the L3 protocol data for this virtual link. Shall
      be present when the associatedLayerProtocol attribute indicates a L3 protocol and
      shall be absent otherwise.
      required: false

```

6.2.14.4 Examples

None.

6.2.14.5 Additional Requirements

None.

6.2.15 toska.datatypes.nfv.L2ProtocolData

6.2.15.1 Description

The L2ProtocolData data type describes L2 protocol data for a given virtual link used in a specific VNF deployment flavour. Table 6.2.15.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.15.1-1: Type name, shorthand and URI

Shorthand Name	L2ProtocolData
Type Qualified Name	toscanfv:L2ProtocolData
Type URI	tosca.datatypes.nfv.L2ProtocolData

6.2.15.2 Properties

The properties of the L2ProtocolData data type shall comply with the provisions set out in table 6.2.15.2-1.

Table 6.2.15.2-1: Properties

Name	Required	Type	Constraints	Description
name	no	string		Identifies the network name associated with this L2 protocol.
network_type	no	string	Valid values: See YAML definition constraints	Specifies the network type for this L2 protocol. The value may be overridden at run-time.
vlan_transparent	yes	boolean	default: false	Specifies whether to support VLAN transparency for this L2 protocol or not.
mtu	no	integer	greater_than: 0	Specifies the Maximum Transmission Unit (MTU) value for this L2 protocol.
segmentation_id	no	string		If present, specifies a specific virtualised network segment, which depends on the network type. For e.g. VLAN ID for VLAN network type and tunnel ID for GRE/VXLAN network types. See note.
NOTE: If this property is included in the VNFD, the property value shall be provided at run-time, unless a default value is provided at design time in the VNFD. If a default value is provided at design-time, this value may be overridden at run-time.				

6.2.15.3 Definition

The syntax of the L2ProtocolData data type shall comply with the following definition:

```

tosca.datatypes.nfv.L2ProtocolData:
  derived_from: toska.datatypes.Root
  description: describes L2 protocol data for a given virtual link used in a
specific VNF deployment flavour.
  properties:
    name:
      type: string
      description: Identifies the network name associated with this L2 protocol.
      required: false
    network_type:
      type: string
      description: Specifies the network type for this L2 protocol. The value
may be overridden at run-time.
      required: false
      constraints:
        - valid_values: [ flat, vlan, vxlan, gre ]
    vlan_transparent:
      type: boolean
      description: Specifies whether to support VLAN transparency for this L2
protocol or not.
      required: true
      default: false
    mtu:
      type: integer
      description: Specifies the maximum transmission unit (MTU) value for this
L2 protocol.
      required: false
      constraints:
        - greater_than: 0
    segmentation_id:
      type: string
      description: Specifies a specific virtualised network segment, which
depends on the network type. For e.g. VLAN ID for VLAN network type and tunnel ID
for GRE/VXLAN network types
      required: false

```

6.2.15.4 Examples

See example in clause A.5.

6.2.15.5 Additional Requirements

None.

6.2.16 `tosca.datatypes.nfv.L3ProtocolData`

6.2.16.1 Description

The `L3ProtocolData` data type describes L3 protocol data for a given virtual link used in a specific VNF deployment flavour. Table 6.2.16.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.16.1-1: Type name, shorthand and URI

Shorthand Name	<code>L3ProtocolData</code>
Type Qualified Name	<code>toscanfv:L3ProtocolData</code>
Type URI	<code>tosca.datatypes.nfv.L3ProtocolData</code>

6.2.16.2 Properties

The properties of the `L3ProtocolData` data type shall comply with the provisions set out in table 6.2.16.2-1.

Table 6.2.16.2-1: Properties

Name	Required	Type	Constraints	Description
<code>name</code>	no	string		Identifies the network name associated with this L3 protocol.
<code>ip_version</code>	yes	string	Valid values: See YAML definition constraints	Specifies IP version of this L3 protocol. The value of the <code>ip_version</code> property shall be consistent with the value of the <code>layer_protocol</code> in the <code>connectivity_type</code> property of the virtual link node.
<code>cidr</code>	yes	string		Specifies the CIDR (Classless Inter-Domain Routing) of this L3 protocol. The value may be overridden at run-time.
<code>ip_allocation_pools</code>	no	list of <code>tosca.datatypes.nfv.IpAllocationPool</code>		Specifies the allocation pools with start and end IP addresses for this L3 protocol. The value may be overridden at run-time.
<code>gateway_ip</code>	no	string		Specifies the gateway IP address for this L3 protocol. The value may be overridden at run-time.
<code>dhcp_enabled</code>	no	boolean		Indicates whether DHCP (Dynamic Host Configuration Protocol) is enabled or disabled for this L3 protocol. The value may be overridden at run-time.
<code>ipv6_address_mode</code>	no	string	Valid values: See YAML definition constraints	Specifies IPv6 address mode. May be present when the value of the <code>ipVersion</code> attribute is "ipv6" and shall be absent otherwise. The value may be overridden at run-time.

6.2.16.3 Definition

The syntax of the L3ProtocolData data type shall comply with the following definition:

```

tosca.datatypes.nfv.L3ProtocolData:
  derived_from: tosca.datatypes.Root
  description: describes L3 protocol data for a given virtual link used in a
specific VNF deployment flavour.
  properties:
    name:
      type: string
      description: Identifies the network name associated with this L3 protocol.
      required: false
    ip_version:
      type: string
      description: Specifies IP version of this L3 protocol. The value of the
ip_version property shall be consistent with the value of the layer_protocol in
the connectivity_type property of the virtual link node.
      required: true
      constraints:
        - valid_values: [ ipv4, ipv6 ]
    cidr:
      type: string
      description: Specifies the CIDR (Classless Inter-Domain Routing) of this
L3 protocol. The value may be overridden at run-time.
      required: true
    ip_allocation_pools:
      type: list
      description: Specifies the allocation pools with start and end IP
addresses for this L3 protocol. The value may be overridden at run-time.
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.IpAllocationPool
    gateway_ip:
      type: string
      description: Specifies the gateway IP address for this L3 protocol. The
value may be overridden at run-time.
      required: false
    dhcp_enabled:
      type: boolean
      description: Indicates whether DHCP (Dynamic Host Configuration Protocol)
is enabled or disabled for this L3 protocol. The value may be overridden at run-
time.
      required: false
    ipv6_address_mode:
      type: string
      description: Specifies IPv6 address mode. May be present when the value of
the ipVersion attribute is "ipv6" and shall be absent otherwise. The value may be
overridden at run-time.
      required: false
      constraints:
        - valid_values: [ slaac, dhcpv6-stateful, dhcpv6-stateless ]

```

6.2.16.4 Examples

None.

6.2.16.5 Additional Requirements

None.

6.2.17 toska.datatypes.nfv.IpAllocationPool

6.2.17.1 Description

The IpAllocationPool data type specifies a range of IP addresses. Table 6.2.17.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.17.1-1: Type name, shorthand, and URI

Shorthand Name	IpAllocationPool
Type Qualified Name	toscanfv:IpAllocationPool
Type URI	tosca.datatypes.nfv.IpAllocationPool

6.2.17.2 Properties

The properties of the IpAllocationPool data type shall comply with the provisions set out in table 6.2.17.2-1.

Table 6.2.17.2-1: Properties

Name	Required	Type	Constraints	Description
start_ip_address	yes	string		The IP address to be used as the first one in a pool of addresses derived from the cidr block full IP range
end_ip_address	yes	string		The IP address to be used as the last one in a pool of addresses derived from the cidr block full IP range

6.2.17.3 Definition

The syntax of the IpAllocationPool data type shall comply with the following definition:

```
tosca.datatypes.nfv.IpAllocationPool:
  derived_from: toska.datatypes.Root
  description: Specifies a range of IP addresses
  properties:
    start_ip_address:
      type: string
      description: The IP address to be used as the first one in a pool of
addresses derived from the cidr block full IP range
      required: true
    end_ip_address:
      type: string
      description: The IP address to be used as the last one in a pool of
addresses derived from the cidr block full IP range
      required: true
```

6.2.17.4 Examples

None.

6.2.17.5 Additional Requirements

None.

6.2.18 toska.datatypes.nfv.InstantiationLevel

6.2.18.1 Description

The InstantiationLevel data type describes the scale level for each aspect that corresponds to a given level of resources to be instantiated within a deployment flavour in term of the number VNFC instances. Table 6.2.18.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.18.1-1: Type name, shorthand, and URI

Shorthand Name	InstantiationLevel
Type Qualified Name	toscanfv:InstantiationLevel
Type URI	tosca.datatypes.nfv.InstantiationLevel

6.2.18.2 Properties

The properties of the InstantiationLevel data type shall comply with the provisions set out in table 6.2.18.2-1.

Table 6.2.18.2-1: Properties

Name	Required	Type	Constraints	Description
description	yes	string		Human readable description of the level.
scale_info	no	map of tosca.datatypes .nfv.ScaleInfo		Represents for each aspect the scale level that corresponds to this instantiation level. scale_info shall be present if the VNF supports scaling.

6.2.18.3 Definition

The syntax of the InstantiationLevel data type shall comply with the following definition:

```
tosca.datatypes.nfv.InstantiationLevel:
  derived_from: toska.datatypes.Root
  description: Describes the scale level for each aspect that corresponds to a given
level of resources to be instantiated within a deployment flavour in term of the
number VNFC instances
  properties:
    description:
      type: string
      description: Human readable description of the level
      required: true
    scale_info:
      type: map # key: aspectId
      description: Represents for each aspect the scale level that corresponds to
this instantiation level. scale_info shall be present if the VNF supports scaling.
      required: false
      entry_schema:
        type: toska.datatypes.nfv.ScaleInfo
```

6.2.18.4 Examples

See clause A.6.

6.2.18.5 Additional Requirements

None.

6.2.19 toska.datatypes.nfv.VduLevel

6.2.19.1 Description

The VduLevel data type indicates for a given Vdu.Compute in a given level the number of instances to deploy. Table 6.2.19.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.19.1-1: Type name, shorthand, and URI

Shorthand Name	VduLevel
Type Qualified Name	toscanfv:VduLevel
Type URI	tosca.datatypes.nfv.VduLevel

6.2.19.2 Properties

The properties of the VduLevel data type shall comply with the provisions set out in table 6.2.19.2-1.

Table 6.2.19.2-1: Properties

Name	Required	Type	Constraints	Description
number_of_instances	yes	integer	greater_or_equal: 0	Number of instances of VNFC based on this VDU to deploy for this level.

6.2.19.3 Definition

The syntax of the VduLevel data type shall comply with the following definition:

```
tosca.datatypes.nfv.VduLevel:
  derived_from: toska.datatypes.Root
  description: Indicates for a given Vdu.Compute in a given level the number of
instances to deploy
  properties:
    number_of_instances:
      type: integer
      description: Number of instances of VNFC based on this VDU to deploy for this
level.
      required: true
      constraints:
        - greater_or_equal: 0
```

6.2.19.4 Examples

See clause A.6.

6.2.19.5 Additional Requirements

None.

6.2.20 toska.datatypes.nfv.VnfLcmOperationsConfiguration

6.2.20.1 Description

The VnfLcmOperationsConfiguration data type represents information to configure lifecycle management operations as specified in ETSI GS NFV-IFA 007 [i.1]. Each VNF LCM operations configuration property represents a container for all attributes that affect the invocation of the corresponding VNF Lifecycle Management operation. Table 6.2.20.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.20.1-1: Type name, shorthand, and URI

Shorthand Name	VnfLcmOperationsConfiguration
Type Qualified Name	toscanfv:VnfLcmOperationsConfiguration
Type URI	tosca.datatypes.nfv.VnfLcmOperationsConfiguration

6.2.20.2 Properties

The properties of the VnfLcmOperationsConfiguration data type shall comply with the provisions set out in table 6.2.20.2-1.

Table 6.2.20.2-1: Properties

Name	Required	Type	Constraints	Description
instantiate	no	tosca.datatypes.nfv.VnfInstantiateOperationConfiguration		Configuration parameters for the InstantiateVnf operation.
scale	no	tosca.datatypes.nfv.VnfScaleOperationConfiguration		Configuration parameters for the ScaleVnf operation.
scale_to_level	no	tosca.datatypes.nfv.VnfScaleToLevelOperationConfiguration		Configuration parameters for the ScaleVnfToLevel operation.
change_flavour	no	tosca.datatypes.nfv.VnfChangeFlavourOperationConfiguration		Configuration parameters for the changeVnfFlavourOpConfig operation.
heal	no	tosca.datatypes.nfv.VnfHealOperationConfiguration		Configuration parameters for the HealVnf operation.
terminate	no	tosca.datatypes.nfv.VnfTerminateOperationConfiguration		Configuration parameters for the TerminateVnf operation.
operate	no	tosca.datatypes.nfv.VnfOperateOperationConfiguration		Configuration parameters for the OperateVnf operation.
change_ext_connectivity	no	tosca.datatypes.nfv.VnfChangeExtConnectivityOperationConfiguration		Configuration parameters for the changeExtVnfConnectivityOpConfig operation.
change_current_package	no	tosca.datatypes.nfv.VnfChangeCurrentPackageOperationConfiguration		Configuration parameters for the ChangeCurrentVnfPackage operation.

Name	Required	Type	Constraints	Description
create_snapshot	no	tosca.datatypes.nfv.VnfCreateSnapshotOperationConfiguration		Configuration parameters for the CreateVnfSnapshot operation.
revert_to_snapshot	no	tosca.datatypes.nfv.VnfRevertToSnapshotOperationConfiguration		Configuration parameters for the RevertToVnfSnapshot operation.
select_vnf_deployable_modules	no	tosca.datatypes.nfv.SelectVnfDeployableModuleOperationConfiguration		Configuration parameters for the SelectVnfDeployableModules operation.

6.2.20.3 Definition

The syntax of the VnfLcmOperationsConfiguration data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfLcmOperationsConfiguration:
  derived_from: toska.datatypes.Root
  description: Represents information to configure lifecycle management operations
  properties:
    instantiate:
      type: toska.datatypes.nfv.VnfInstantiateOperationConfiguration
      description: Configuration parameters for the InstantiateVnf operation
      required: false
    scale:
      type: toska.datatypes.nfv.VnfScaleOperationConfiguration
      description: Configuration parameters for the ScaleVnf operation
      required: false
    scale_to_level:
      type: toska.datatypes.nfv.VnfScaleToLevelOperationConfiguration
      description: Configuration parameters for the ScaleVnfToLevel operation
      required: false
    change_flavour:
      type: toska.datatypes.nfv.VnfChangeFlavourOperationConfiguration
      description: Configuration parameters for the changeVnfFlavourOpConfig
operation
      required: false
    heal:
      type: toska.datatypes.nfv.VnfHealOperationConfiguration
      description: Configuration parameters for the HealVnf operation
      required: false
    terminate:
      type: toska.datatypes.nfv.VnfTerminateOperationConfiguration
      description: Configuration parameters for the TerminateVnf operation
      required: false
    operate:
      type: toska.datatypes.nfv.VnfOperateOperationConfiguration
      description: Configuration parameters for the OperateVnf operation
      required: false
    change_ext_connectivity:
      type: toska.datatypes.nfv.VnfChangeExtConnectivityOperationConfiguration
      description: Configuration parameters for the changeExtVnfConnectivityOpConfig
operation
      required: false
    change_current_package:
      type: toska.datatypes.nfv.VnfChangeCurrentPackageOperationConfiguration
      description: Configuration parameters for the ChangeCurrentVnfPackage
operation

```

```

    required: false
    # derived types are expected to introduce new properties, with their type
derived from
    # tosca.datatypes.nfv.VnfChangeCurrentPackageOperationConfiguration,
    # with the same name as the operation designated to the
ChangeCurrentVnfPackage request
    create_snapshot:
      type: tosca.datatypes.nfv.VnfCreateSnapshotOperationConfiguration
      description: Configuration parameters for the CreateVnfSnapshot operation
      required: false
    revert_to_snapshot:
      type: tosca.datatypes.nfv.VnfRevertToSnapshotOperationConfiguration
      description: Configuration parameters for the RevertToVnfSnapshot operation
      required: false
    select_vnf_deployable_modules:
      type: tosca.datatypes.nfv.SelectVnfDeployableModulesOperationConfiguration
      description: Configuration parameters for the SelectVnfDeployableModules
operation
      required: false

```

6.2.20.4 Examples

None.

6.2.20.5 Additional Requirements

None.

6.2.21 tosca.datatypes.nfv.VnfInstantiateOperationConfiguration

6.2.21.1 Description

The VnfInstantiateOperationConfiguration data type represents information that affect the invocation of the InstantiateVnf operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.21.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.21.1-1: Type name, shorthand, and URI

Shorthand Name	VnfInstantiateOperationConfiguration
Type Qualified Name	toscanfv:VnfInstantiateOperationConfiguration
Type URI	tosca.datatypes.nfv.VnfInstantiateOperationConfiguration

6.2.21.2 Properties

The properties of the VnfInstantiateOperationConfiguration data type shall comply with the provisions set out in table 6.2.21.2-1.

Table 6.2.21.2-1: Properties

Name	Required	Type	Constraints	Description
target_scale_levels_supported	yes	boolean		Signals whether target scale levels are supported by this VNF during instantiation. Default is FALSE, i.e. "not supported".

6.2.21.3 Definition

The syntax of the VnfInstantiateOperationConfiguration data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfInstantiateOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the
InstantiateVnf operation.
  properties:
    target_scale_levels_supported:
      type: boolean
      description: Signals whether passing a value larger than one in the
numScalingSteps parameter of the ScaleVnf operation is supported by this VNF.
      required: true
      default: false

```

6.2.21.4 Examples

None.

6.2.21.5 Additional Requirements

None.

6.2.22 toska.datatypes.nfv.VnfScaleOperationConfiguration

6.2.22.1 Description

VnfScaleOperationConfiguration represents information that affect the invocation of the ScaleVnf operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.22.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.22.1-1: Type name, shorthand, and URI

Shorthand Name	VnfScaleOperationConfiguration
Type Qualified Name	toscanfv:VnfScaleOperationConfiguration
Type URI	tosca.datatypes.nfv.VnfScaleOperationConfiguration

6.2.22.2 Properties

The properties of the VnfScaleOperationConfiguration data type shall comply with the provisions set out in table 6.2.22.2-1.

Table 6.2.22.2-1: Properties

Name	Required	Type	Constraints	Description
scaling_by_more_than_one_step_supported	yes	boolean		Signals whether passing a value larger than one in the numScalingSteps parameter of the ScaleVnf operation is supported by this VNF. Default is FALSE, i.e. "not supported".

6.2.22.3 Definition

The syntax of the VnfScaleOperationConfiguration data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfScaleOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: Represents information that affect the invocation of the ScaleVnf
operation
  properties:
    scaling_by_more_than_one_step_supported:
      type: boolean
      description: Signals whether passing a value larger than one in the
numScalingSteps parameter of the ScaleVnf operation is supported by this VNF.
      required: true
      default: false
```

6.2.22.4 Examples

See clause 6.8.1.9.

6.2.22.5 Additional Requirements

None.

6.2.23 toska.datatypes.nfv.VnfScaleToLevelOperationConfiguration

6.2.23.1 Description

The VnfScaleToLevelOperationConfiguration data type represents information that affect the invocation of the ScaleVnfToLevel operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.23.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.23.1-1: Type name, shorthand, and URI

Shorthand Name	VnfScaleToLevelOperationConfiguration
Type Qualified Name	toscanfv:VnfScaleToLevelOperationConfiguration
Type URI	tosca.datatypes.nfv.VnfScaleToLevelOperationConfiguration

6.2.23.2 Properties

The properties of the VnfScaleToLevelOperationConfiguration data type shall comply with the provisions set out in table 6.2.23.2-1.

Table 6.2.23.2-1: Properties

Name	Required	Type	Constraints	Description
arbitrary_target_levels_supported	yes	boolean		Signals whether scaling according to the parameter "scaleInfo" is supported by this VNF.

6.2.23.3 Definition

The syntax of the VnfScaleToLevelOperationConfiguration data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfScaleToLevelOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the
ScaleVnfToLevel operation
```

```

properties:
  arbitrary_target_levels_supported:
    type: boolean
    description: Signals whether scaling according to the parameter "scaleInfo" is
supported by this VNF
    required: true

```

6.2.23.4 Examples

See clause 6.8.1.9.

6.2.23.5 Additional Requirements

None.

6.2.24 `tosca.datatypes.nfv.VnfHealOperationConfiguration`

6.2.24.1 Description

The `VnfHealOperationConfiguration` data type represents information that affect the invocation of the `HealVnf` operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.24.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.24.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfHealOperationConfiguration</code>
Type Qualified Name	<code>toscanfv:VnfHealOperationConfiguration</code>
Type URI	<code>tosca.datatypes.nfv.VnfHealOperationConfiguration</code>

6.2.24.2 Properties

The properties of the `VnfHealOperationConfiguration` data type shall comply with the provisions set out in table 6.2.24.2-1.

Table 6.2.24.2-1: Properties

Name	Required	Type	Constraints	Description
<code>causes</code>	no	list of string		Supported "cause" parameter values.

6.2.24.3 Definition

The syntax of the `VnfHealOperationConfiguration` data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfHealOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the HealVnf
operation
  properties:
    causes:
      type: list
      description: Supported "cause" parameter values
      required: false
      entry_schema:
        type: string

```

6.2.24.4 Examples

See clause 6.8.1.9.

6.2.24.5 Additional Requirements

None.

6.2.25 `tosca.datatypes.nfv.VnfTerminateOperationConfiguration`

6.2.25.1 Description

The `VnfTerminateOperationConfiguration` data type represents information that affect the invocation of the `TerminateVnf`, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.25.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.25.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfTerminateOperationConfiguration</code>
Type Qualified Name	<code>toscanfv:VnfTerminateOperationConfiguration</code>
Type URI	<code>tosca.datatypes.nfv.VnfTerminateOperationConfiguration</code>

6.2.25.2 Properties

The properties of the `VnfTerminateOperationConfiguration` data type shall comply with the provisions set out in table 6.2.25.2-1.

Table 6.2.25.2-1: Properties

Name	Required	Type	Constraints	Description
<code>min_graceful_termination_timeout</code>	yes	scalar-unit.time		Minimum timeout value for graceful termination of a VNF instance.
<code>max_recommended_graceful_termination_timeout</code>	no	scalar-unit.time		Maximum recommended timeout value that can be needed to gracefully terminate a VNF instance of a particular type under certain conditions, such as maximum load condition. This is provided by VNF provider as information for the operator facilitating the selection of optimal timeout value. This value is not used as constraint.

6.2.25.3 Definition

The syntax of the `VnfTerminateOperationConfiguration` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfTerminateOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the TerminateVnf
  properties:
    min_graceful_termination_timeout:
      type: scalar-unit.time
      description: Minimum timeout value for graceful termination of a VNF instance
      required: true
    max_recommended_graceful_termination_timeout:
      type: scalar-unit.time
```

```

    description: Maximum recommended timeout value that can be needed to
    gracefully terminate a VNF instance of a particular type under certain conditions,
    such as maximum load condition. This is provided by VNF provider as information for
    the operator facilitating the selection of optimal timeout value. This value is not
    used as constraint
    required: false

```

6.2.25.4 Examples

None.

6.2.25.5 Additional Requirements

None.

6.2.26 `tosca.datatypes.nfv.VnfOperateOperationConfiguration`

6.2.26.1 Description

The `VnfOperateOperationConfiguration` data type represents information that affect the invocation of the `OperateVnf` operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.26.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.26.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfOperateOperationConfiguration</code>
Type Qualified Name	<code>tosca.nfv.VnfOperateOperationConfiguration</code>
Type URI	<code>tosca.datatypes.nfv.VnfOperateOperationConfiguration</code>

6.2.26.2 Properties

The properties of the `VnfOperateOperationConfiguration` data type shall comply with the provisions set out in table 6.2.26.2-1.

Table 6.2.26.2-1: Properties

Name	Required	Type	Constraints	Description
<code>min_graceful_stop_timeout</code>	yes	<code>scalar-unit.time</code>		Minimum timeout value for graceful stop of a VNF instance.
<code>max_recommended_graceful_stop_timeout</code>	no	<code>scalar-unit.time</code>		Maximum recommended timeout value that can be needed to gracefully stop a VNF instance of a particular type under certain conditions, such as maximum load condition. This is provided by VNF provider as information for the operator facilitating the selection of optimal timeout value. This value is not used as constraint.

6.2.26.3 Definition

The syntax of the `VnfOperateOperationConfiguration` data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfOperateOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the OperateVnf
  operation
  properties:
    min_graceful_stop_timeout:
      type: scalar-unit.time

```

```

    description: Minimum timeout value for graceful stop of a VNF instance
    required: true
  max_recommended_graceful_stop_timeout:
    type: scalar-unit.time
    description: Maximum recommended timeout value that can be needed to
    gracefully stop a VNF instance of a particular type under certain conditions, such as
    maximum load condition. This is provided by VNF provider as information for the
    operator facilitating the selection of optimal timeout value. This value is not used
    as constraint
    required: false

```

6.2.26.4 Examples

None.

6.2.26.5 Additional Requirements

None.

6.2.27 `tosca.datatypes.nfv.ScaleInfo`

6.2.27.1 Description

The ScaleInfo data type is defined in clause 9.2.12 of the present document.

6.2.28 `tosca.datatypes.nfv.ScalingAspect`

6.2.28.1 Description

The ScalingAspect data type describes the details of an aspect used for horizontal scaling. Table 6.2.28.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.28.1-1: Type name, shorthand, and URI

Shorthand Name	ScalingAspect
Type Qualified Name	toscanfv:ScalingAspect
Type URI	tosca.datatypes.nfv.ScalingAspect

6.2.28.2 Properties

The properties of the ScalingAspect data type shall comply with the provisions set out in table 6.2.28.2-1.

Table 6.2.28.2-1: Properties

Name	Required	Type	Constraints	Description
name	yes	string		Human readable name of the aspect.
description	yes	string		Human readable description of the aspect.
max_scale_level	yes	integer	positiveInteger	Total number of scaling steps that can be applied with regards to this aspect. The value of this property corresponds to the number of scaling steps can be applied to this aspect when scaling it from the minimum scale level (i.e. 0) to the maximum scale level defined by this property.
step_deltas	no	list of string		List of scaling deltas to be applied for the different subsequent scaling steps of this aspect. The first entry in the array shall correspond to the first scaling step (between scale levels 0 to 1) and the last entry in the array shall correspond to the last scaling step (between maxScaleLevel-1 and maxScaleLevel).

6.2.28.3 Definition

The syntax of the ScalingAspect data type shall comply with the following definition:

```

tosca.datatypes.nfv.ScalingAspect:
  derived_from: toska.datatypes.Root
  description: describes the details of an aspect used for horizontal scaling
  properties:
    name:
      type: string
      description: Human readable name of the aspect
      required: true
    description:
      type: string
      description: Human readable description of the aspect
      required: true
    max_scale_level:
      type: integer # positiveInteger
      description: Total number of scaling steps that can be applied w.r.t. this
aspect. The value of this property corresponds to the number of scaling steps can be
applied to this aspect when scaling it from the minimum scale level (i.e. 0) to the
maximum scale level defined by this property
      required: true
      constraints:
        - greater_or_equal: 0
    step_deltas:
      type: list
      description: List of scaling deltas to be applied for the different subsequent
scaling steps of this aspect. The first entry in the array shall correspond to the
first scaling step (between scale levels 0 to 1) and the last entry in the array shall
correspond to the last scaling step (between maxScaleLevel-1 and maxScaleLevel)
      required: false
      entry_schema:
        type: string # Identifier

```

6.2.28.4 Examples

See clause A.6.

6.2.28.5 Additional Requirements

None.

6.2.29 `tosca.datatypes.nfv.LinkBitrateRequirements`

6.2.29.1 Description

The `LinkBitrateRequirements` data type is defined in clause 9.2.5 of the present document.

6.2.30 `tosca.datatypes.nfv.ConnectivityType`

6.2.30.1 Description

The `ConnectivityType` data type is defined in clause 9.2.4 of the present document.

6.2.31 `tosca.datatypes.nfv.VnfConfigurableProperties`

6.2.31.1 Description

The `VnfConfigurableProperties` data type describes configurable properties for a given VNF. Configurable properties can be standardized as listed below (e.g. related to auto scaling, auto healing and interface configuration) or can be VNF-specific as defined by the VNF provider.

The value of all VNF configurable properties listed in table 6.2.31.2-1 shall be modifiable anytime (including after instantiation of the VNF) via the Modify VNF information operation, unless stated otherwise in the description of the specific VNF configurable property.

Table 6.2.31.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.31.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfConfigurableProperties</code>
Type Qualified Name	<code>toscanfv:VnfConfigurableProperties</code>
Type URI	<code>tosca.datatypes.nfv.VnfConfigurableProperties</code>

6.2.31.2 Properties

The properties of the `VnfConfigurableProperties` data type shall comply with the provisions set out in table 6.2.31.2-1.

Table 6.2.31.2-1: Properties

Name	Required	Type	Constraints	Description
is_autoscale_enabled	no	boolean		It permits to enable (TRUE)/disable (FALSE) the auto-scaling functionality. If the property is not present, then configuring this VNF property is not supported.
is_autoheal_enabled	no	boolean		It permits to enable (TRUE)/disable (FALSE) the auto-healing functionality. If the property is not present, then configuring this VNF property is not supported.
vnfm_interface_info	no	list of tosca.datatypes .nfv.VnfmInterfa ceInfo		Contains information enabling the VNF instance to access to the NFV-MANO interfaces produced by the VNFM (e.g. URIs and credentials). If the property is not present, then configuring this VNF property is not supported. If this attribute is declared for a VNF, its initial value shall be set prior to or at instantiation time (as initial value in the VNFD or via the VNF LCM interface). Its value shall be further modifiable after instantiation via the Modify VNF information operation.
vnfm_oauth_server_info	no	tosca.datatypes .nfv.OauthServ erInfo		Contains information to enable discovery of the authorization server protecting access to VNFM interfaces. If the property is not present, then configuring this VNF property is not supported. If this attribute is declared for a VNF, its initial value shall be set prior to or at instantiation time (as initial value in the VNFD or via the VNF LCM interface). Its value shall be further modifiable after instantiation via the Modify VNF information operation.
vnf_oauth_server_info	no	tosca.datatypes .nfv.OauthServ erInfo		Contains information to enable discovery of the authorization server to validate the access tokens provided by the VNFM when the VNFM accesses the VNF interfaces, if that functionality (token introspection) is supported by the authorization server. If the property is not present, then configuring this VNF property is not supported. If this attribute is declared for a VNF, its initial value shall be set prior to or at instantiation time (as initial value in the VNFD or via the VNF LCM interface). Its value shall be further modifiable after instantiation via the Modify VNF information operation.
additional_configurable_property	no	tosca.datatypes .nfv.VnfAddition alConfigurable Properties		It provides VNF specific configurable properties that can be modified using the ModifyVnfInfo operation. If some of these properties are declared as required, their values shall be set prior to or at instantiation time (as initial value in

Name	Required	Type	Constraints	Description
				the VNFD or via the VNF LCM interface). Their values may be modifiable after instantiation via the Modify VNF information operation if such modification of individual attributes is supported by the VNF and declared per attribute in the VNFD.
mgmt_entity_interface_info	no	map of string		Contains information enabling the peering (connectivity) to the external management entity (e.g. URIs, credentials, and address information) managing the VNF. If the property is not present, then configuring this VNF property is not supported. If this attribute is declared for a VNF, its initial value shall be set prior to or at instantiation time (as initial value in the VNFD or via the VNF LCM interface). Its value shall be further modifiable after instantiation via the Modify VNF information operation.

6.2.31.3 Definition

The syntax of the VnfConfigurableProperties data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfConfigurableProperties:
  derived_from: tosca.datatypes.Root
  description: indicates configuration properties for a given VNF (e.g. related to
auto scaling and auto healing).
  properties:
    is_autoscale_enabled:
      type: boolean
      description: It permits to enable (TRUE)/disable (FALSE) the auto-scaling
functionality. If the property is not present, then configuring this VNF property is
not supported.
      required: false
    is_autoheal_enabled:
      type: boolean
      description: It permits to enable (TRUE)/disable (FALSE) the auto-healing
functionality. If the property is not present, then configuring this VNF property is
not supported.
      required: false
    vnfm_interface_info:
      type: list
      description: Contains information enabling access to the NFV-MANO interfaces
produced by the VNFM (e.g. URIs and credentials). If the property is not present, then
configuring this VNF property is not supported.
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.VnfmInterfaceInfo
    vnfm_oauth_server_info:
      type: tosca.datatypes.nfv.OauthServerInfo
      description: Contains information to enable discovery of the authorization
server protecting access to VNFM interfaces. If the property is not present, then
configuring this VNF property is not supported.
      required: false
    vnf_oauth_server_info:
      type: tosca.datatypes.nfv.OauthServerInfo

```

```

    description: Contains information to enable discovery of the authorization
server to validate the access tokens provided by the VNFM when the VNFM accesses the
VNF interfaces, if that functionality (token introspection) is supported by the
authorization server. If the property is not present, then configuring this VNF
property is not supported.
    required: false
    additional_configurable_properties:
    description: It provides VNF specific configurable properties that can be
modified using the ModifyVnfInfo operation
    required: false
    type: tosca.datatypes.nfv.VnfAdditionalConfigurableProperties
    # derived types are expected to introduce
    # additional_configurable_properties with its type derived from
    # tosca.datatypes.nfv.VnfAdditionalConfigurableProperties
    mgmt_entity_interface_info:
    type: map
    description: Contains information enabling the peering (connectivity) to the
external management entity (e.g. URIs, credentials, and address information) managing
the VNF. If the property is not present, then configuring this VNF property is not
supported.
    required: false
    entry_schema:
    type: string

```

6.2.31.4 Examples

Example definition of configurable properties without properties assignment value.

```

tosca_definitions_version: tosca_simple_yaml_1_3

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      flavour_id:
        constraints:
          - valid_values: [ simple, complex ]
    configurable_properties:
      type: MyCompany.datatypes.nfv.VnfConfigurableProperties

data_types:
  Mycompany.datatypes.nfv.VnfConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfConfigurableProperties
    properties:
      additional_configurable_properties:
        type: MyCompany.datatypes.nfv.VnfAdditionalConfigurableProperties

  MyCompany.datatypes.nfv.VnfAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfAdditionalConfigurableProperties
    properties:
      name_prefix_in_vim:
        type: string
        required: false
      dns_server:
        type: string
        required: true

```

In the above example, properties definitions are provided and properties assignment values are not necessary. The properties values are available in the API.

6.2.31.5 Additional Requirements

None.

6.2.32 `tosca.datatypes.nfv.VnfAdditionalConfigurableProperties`

6.2.32.1 Description

The `VnfAdditionalConfigurableProperties` data type is an empty base type for deriving data types for describing additional configurable properties for a given VNF. Table 6.2.32.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.32.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfAdditionalConfigurableProperties</code>
Type Qualified Name	<code>toscanfv:VnfAdditionalConfigurableProperties</code>
Type URI	<code>tosca.datatypes.nfv.VnfAdditionalConfigurableProperties</code>

6.2.32.2 Properties

The properties of the `VnfAdditionalConfigurableProperties` data type shall comply with the provisions set out in table 6.2.32.2-1.

Table 6.2.32.2-1: Properties

Name	Required	Type	Constraints	Description
<code>is_writable_anytime</code>	yes	boolean		It specifies whether these additional configurable properties are writeable (TRUE) at anytime (i.e. prior to / at instantiation time as well as after instantiation).or (FALSE) only prior to / at instantiation time. If this property is not present, the additional configurable properties are writable anytime.

6.2.32.3 Definition

The syntax of the `VnfAdditionalConfigurableProperties` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfAdditionalConfigurableProperties:
  derived_from: toska.datatypes.Root
  description: is an empty base type for deriving data types for describing
additional configurable properties for a given VNF
  properties:
    is_writable_anytime:
      type: boolean
      description: It specifies whether these additional configurable properties are
writeable (TRUE) at any time (i.e. prior to / at instantiation time as well as after
instantiation).or (FALSE) only prior to / at instantiation time. If this property is
not present, the additional configurable properties are writable anytime.
      required: true
      default: true
```

6.2.32.4 Examples

See clause 6.2.31.4.

6.2.32.5 Additional Requirements

None.

6.2.33 `tosca.datatypes.nfv.VnfInfoModifiableAttributes`

6.2.33.1 Description

The `VnfInfoModifiableAttributes` data type describes VNF-specific extension and metadata for a given VNF. Table 6.2.33.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.33.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfInfoModifiableAttributes</code>
Type Qualified Name	<code>toscanfv:VnfInfoModifiableAttributes</code>
Type URI	<code>tosca.datatypes.nfv.VnfInfoModifiableAttributes</code>

6.2.33.2 Properties

The properties of the `VnfInfoModifiableAttributes` data type shall comply with the provisions set out in table 6.2.33.2-1.

Table 6.2.33.2-1: Properties

Name	Required	Type	Constraints	Description
extensions	no	<code>tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions</code>		"Extension" properties of <code>VnfInfo</code> that are writeable.
metadata	no	<code>tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata</code>		"Metadata" properties of <code>VnfInfo</code> that are writeable.

6.2.33.3 Definition

The syntax of the `VnfInfoModifiableAttributes` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfInfoModifiableAttributes:
  derived_from: toska.datatypes.Root
  description: Describes VNF-specific extension and metadata for a given VNF
  properties:
    extensions:
      type: toska.datatypes.nfv.VnfInfoModifiableAttributesExtensions
      description: Extension properties of VnfInfo that are writeable
      required: false
      # derived types are expected to introduce
      # extensions with its type derived from
      # toska.datatypes.nfv.VnfInfoModifiableAttributesExtensions
    metadata:
      type: toska.datatypes.nfv.VnfInfoModifiableAttributesMetadata
      description: Metadata properties of VnfInfo that are writeable
      required: false
      # derived types are expected to introduce
      # metadata with its type derived from
      # toska.datatypes.nfv.VnfInfoModifiableAttributesMetadata
```

6.2.33.4 Examples

The following example shows an example of a derived VnfInfoModifiableAttributesExtensions data type that contains one security-sensitive property.

```
tosca_definitions_version: tosca_simple_yaml_1_3

data_types:
  MyCompany.datatypes.nfv.VnfInfoModifiableAttributes:
    derived_from: tosca.datatypes.nfv.VnfInfoModifiableAttributes
    properties:
      extensions:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions

  mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions:
    derived_from: tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions
    properties:
      password:
        type: string
      metadata:
        sensitive: "true"
```

See clause 6.8.1.9 for other examples.

6.2.33.5 Additional Requirements

None.

6.2.34 tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions

6.2.34.1 Description

The VnfInfoModifiableAttributesExtensions data type is an empty base type for deriving data types for describing VNF-specific extension. Table 6.2.34.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.34.1-1: Type name, shorthand, and URI

Shorthand Name	VnfInfoModifiableAttributesExtensions
Type Qualified Name	toscanfv:VnfInfoModifiableAttributesExtensions
Type URI	tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions

6.2.34.2 Properties

None.

6.2.34.3 Definition

The syntax of the VnfInfoModifiableAttributesExtensions data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions:
  derived_from: tosca.datatypes.Root
  description: is an empty base type for deriving data types for describing VNF-
specific extension
```

6.2.34.4 Examples

See clause 6.8.1.9.

6.2.34.5 Additional Requirements

None.

6.2.35 `tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata`

6.2.35.1 Description

The `VnfInfoModifiableAttributesMetadata` data type is an empty base type for deriving data types for describing VNF-specific metadata, like information about supported protocols and data models for configuring the VNF. Table 6.2.35.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.35.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfInfoModifiableAttributesMetadata</code>
Type Qualified Name	<code>tosca.nfv.VnfInfoModifiableAttributesMetadata</code>
Type URI	<code>tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata</code>

6.2.35.2 Properties

None.

6.2.35.3 Definition

The syntax of the `VnfInfoModifiableAttributesMetadata` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata:
  derived_from: toska.datatypes.Root
  description: is an empty base type for deriving data types for describing VNF-
specific metadata, like information about supported protocols and data models for
configuring the VNF.
```

6.2.35.4 Examples

Example metadata definition:

```
tosca_definitions_version: toska_simple_yaml_1_3

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: toska.nodes.nfv.VNF
    properties:
      flavour_id:
        constraints:
          - valid_values: [ simple, complex ]
      modifiable_attributes:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributes

data_types:
  mycompany.datatypes.nfv.VnfInfoModifiableAttributes:
    derived_from: toska.datatypes.nfv.VnfInfoModifiableAttributes
    properties:
      metadata:
```

```

    type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesMetadata

mycompany.datatypes.nfv.VnfInfoModifiableAttributesMetadata:
  derived_from: tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata
  properties:
    supported_configuration_protocols:
      type: string
      required: false
    supported_configuration_data_models:
      type: string
      required: false

```

6.2.35.5 Additional Requirements

None.

6.2.36 tosca.datatypes.nfv.Qos

6.2.36.1 Description

The Qos data type is defined in clause 9.2.7 of the present document.

6.2.37 tosca.datatypes.nfv.LogicalNodeData

6.2.37.1 Description

The LogicalNodeData data type describes compute, memory and I/O requirements associated with a particular VDU. Table 6.2.37.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.37.1-1: Type name, shorthand, and URI

Shorthand Name	LogicalNodeData
Type Qualified Name	toscanfv:LogicalNodeData
Type URI	tosca.datatypes.nfv.LogicalNodeData

6.2.37.2 Properties

The properties of the LogicalNodeData data type shall comply with the provisions set out in table 6.2.37.2-1.

Table 6.2.37.2-1: Properties

Name	Required	Type	Constraints	Description
logical_node_requirements	no	map of string		The logical node-level compute, memory and I/O requirements. A map of strings that contains a set of key-value pairs that describes hardware platform specific deployment requirements, including the number of CPU cores on this logical node, a memory configuration specific to a logical node or a requirement related to the association of an I/O device with the logical node. More information regarding the usage of this property is available at: https://register.etsi.org.

6.2.37.3 Definition

The syntax of the LogicalNodeData data type shall comply with the following definition:

```
tosca.datatypes.nfv.LogicalNodeData:
  derived_from: toska.datatypes.Root
  description: Describes compute, memory and I/O requirements associated with a
particular VDU.
  properties:
    logical_node_requirements:
      type: map
      description: The logical node-level compute, memory and I/O requirements. A
map of strings that contains a set of key-value pairs that describes hardware
platform specific deployment requirements, including the number of CPU cores on this
logical node, a memory configuration specific to a logical node or a requirement
related to the association of an I/O device with the logical node.
      required: false
      entry_schema:
        type: string
```

6.2.37.4 Examples

None.

6.2.37.5 Additional Requirements

None.

6.2.38 Void

6.2.39 toska.datatypes.nfv.VirtualBlockStorageData

6.2.39.1 Description

The VirtualBlockStorageData data type describes block storage requirements associated with compute resources in a particular VDU, either as a local disk or as virtual attached storage. Table 6.2.39.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.39.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualBlockStorageData
Type Qualified Name	toscanfv:VirtualBlockStorageData
Type URI	tosca.datatypes.nfv.VirtualBlockStorageData

6.2.39.2 Properties

The properties of the VirtualBlockStorageData data type shall comply with the provisions set out in table 6.2.39.2-1.

Table 6.2.39.2-1: Properties

Name	Required	Type	Constraints	Description
size_of_storage	yes	scalar-unit.size	greater_or_equal: 0 B	Size of virtualised storage resource.

Name	Required	Type	Constraints	Description
size_of_storage_valid-values	no	tosca.datatypes.nfv.SizeRange		Indicates valid values for the size of the virtualized storage resource. If this property is present the size of the virtualized storage resource can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the block storage resource is instantiated with the value indicated in the size_of_storage property. If this property is not present the size of the virtualized storage resource is not configurable via the VNF LCM interface and is always equal to the value indicated in the size_of_storage property.
vdu_storage_requirements	no	map of string		The hardware platform specific storage requirements. A map of strings that contains a set of key-value pairs that represents the hardware platform specific storage deployment requirements. More information regarding the usage of this property is available at: https://register.etsi.org .
rdma_enabled	yes	boolean	default: false	Indicate if the storage support RDMA.

6.2.39.3 Definition

The syntax of the VirtualBlockStorageData data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualBlockStorageData:
  derived_from: toska.datatypes.Root
  description: VirtualBlockStorageData describes block storage requirements
  associated with compute resources in a particular VDU, either as a local disk or as
  virtual attached storage
  properties:
    size_of_storage:
      type: scalar-unit.size
      description: Size of virtualised storage resource
      required: true
      constraints:
        - greater_or_equal: 0 B
    size_of_storage_valid_values:
      type: toska.datatypes.nfv.SizeRange
      description: Indicates valid values for the size of the virtualized storage
      resource. If this property is present the size of the virtualized storage resource can
      be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM
      operation, the block storage resource is instantiated with the value indicated in the
      size_of_storage property. If this property is not present the size of the virtualized
      storage resource is not configurable via the VNF LCM interface and is always equal to
      the value indicated in the size_of_storage property.
      required: false
    vdu_storage_requirements:
      type: map
      description: The hardware platform specific storage requirements. A map of
      strings that contains a set of key-value pairs that represents the hardware platform
      specific storage deployment requirements
      required: false
    entry_schema:
      type: string

```

```

rdma_enabled:
  type: boolean
  description: Indicates if the storage support RDMA
  required: true
  default: false

```

6.2.39.4 Examples

None.

6.2.39.5 Additional Requirements

None.

6.2.40 `tosca.datatypes.nfv.VirtualObjectStorageData`

6.2.40.1 Description

The `VirtualObjectStorageData` data type describes object storage requirements associated with compute resources in a particular VDU. Table 6.2.40.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.40.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VirtualObjectStorageData</code>
Type Qualified Name	<code>toscanfv:VirtualObjectStorageData</code>
Type URI	<code>tosca.datatypes.nfv.VirtualObjectStorageData</code>

6.2.40.2 Properties

The properties of the `VirtualObjectStorageData` data type shall comply with the provisions set out in table 6.2.40.2-1.

Table 6.2.40.2-1: Properties

Name	Required	Type	Constraints	Description
<code>max_size_of_storage</code>	no	scalar-unit.size	greater_or_equal: 0 B	Maximum size of virtualised storage resource.
<code>max_size_of_storage_valid_values</code>	no	<code>tosca.datatypes.nfv.SizeRange</code>		Indicates valid values for the maximum size of the virtualized storage resource. If this property is present the maximum size of the virtualized storage resource can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the object storage resource is instantiated with the value indicated in the <code>max_size_of_storage</code> property. If this property is not present the maximum size of the virtualized storage resource is not configurable via the VNF LCM interface and it is set to the value indicated in the <code>max_size_of_storage</code> property, if this property is present. This property shall not be present if the <code>max_size_of_storage</code> is not present.

6.2.40.3 Definition

The syntax of the VirtualObjectStorageData data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualObjectStorageData:
  derived_from: tosca.datatypes.Root
  description: VirtualObjectStorageData describes object storage requirements
  associated with compute resources in a particular VDU
  properties:
    max_size_of_storage:
      type: scalar-unit.size
      description: Maximum size of virtualised storage resource
      required: false
      constraints:
        - greater_or_equal: 0 B
    max_size_of_storage_valid_values:
      type: tosca.datatypes.nfv.SizeRange
      description: Indicates valid values for the maximum size of the
  virtualized storage resource. If this property is present the maximum size of the
  virtualized storage resource can be indicated in a VNF LCM operation. If no value
  is indicated in the VNF LCM operation, the object storage resource is instantiated
  with the value indicated in the max_size_of_storage property. If this property is
  not present the maximum size of the virtualized storage resource is not
  configurable via the VNF LCM interface and it is set to the value indicated in the
  in the max_size_of_storage property, if this property is present.
      required: false

```

6.2.40.4 Examples

None.

6.2.40.5 Additional Requirements

None.

6.2.41 tosca.datatypes.nfv.VirtualFileStorageData

6.2.41.1 Description

The VirtualFileStorageData data type describes file storage requirements associated with compute resources in a particular VDU. Table 6.2.41.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.41.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualFileStorageData
Type Qualified Name	toscanfv:VirtualFileStorageData
Type URI	tosca.datatypes.nfv.VirtualFileStorageData

6.2.41.2 Properties

The properties of the VirtualFileStorageData data type shall comply with the provisions set out in table 6.2.41.2-1.

Table 6.2.41.2-1: Properties

Name	Required	Type	Constraints	Description
size_of_storage	yes	scalar-unit.size	greater_or_equal: 0 B	Size of virtualised storage resource.

Name	Required	Type	Constraints	Description
size_of_storage_valid_values	no	tosca.datatypes.nfv.SizeRange		Indicates valid values for the size of the virtualized storage resource. If this property is present the size of the virtualized storage resource can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the file storage resource is instantiated with the value indicated in the size_of_storage property. If this property is not present the size of the virtualized storage resource is not configurable via the VNF LCM interface and is always equal to the value indicated in the size_of_storage property.
file_system_protocol	yes	string	Valid values: See YAML definition constraints	The shared file system protocol.

6.2.41.3 Definition

The syntax of the VirtualFileStorageData data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualFileStorageData:
  derived_from: toska.datatypes.Root
  description: VirtualFileStorageData describes file storage requirements
  associated with compute resources in a particular VDU
  properties:
    size_of_storage:
      type: scalar-unit.size
      description: Size of virtualised storage resource
      required: true
      constraints:
        - greater_or_equal: 0 B
    size_of_storage_valid_values:
      type: toska.datatypes.nfv.SizeRange
      description: Indicates valid values for the size of the virtualized
storage resource. If this property is present the size of the virtualized storage
resource can be indicated in a VNF LCM operation. If no value is indicated in the
VNF LCM operation, the file storage resource is instantiated with the value
indicated in the size_of_storage property. If this property is not present the
size of the virtualized storage resource is not configurable via the VNF LCM
interface and is always equal to the value indicated in the size_of_storage
property.
      required: false
    file_system_protocol:
      type: string
      description: The shared file system protocol
      required: true
      constraints:
        - valid_values: [ nfs, cifs ]

```

6.2.41.4 Examples

None.

6.2.41.5 Additional Requirements

None.

6.2.42 `tosca.datatypes.nfv.VirtualLinkBitrateLevel`

6.2.42.1 Description

The `VirtualLinkBitrateLevel` data type describes bitrate requirements applicable to the virtual link instantiated from a particular `VnfVirtualLink`. Table 6.2.42.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.42.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VirtualLinkBitrateLevel</code>
Type Qualified Name	<code>tosca.nfv.VirtualLinkBitrateLevel</code>
Type URI	<code>tosca.datatypes.nfv.VirtualLinkBitrateLevel</code>

6.2.42.2 Properties

The properties of the `VirtualLinkBitrateLevel` data type shall comply with the provisions set out in table 6.2.42.2-1.

Table 6.2.42.2-1: Properties

Name	Required	Type	Constraints	Description
<code>bitrate_requirements</code>	yes	<code>tosca.datatypes.nfv.LinkBitrateRequirements</code>		Virtual link bitrate requirements for an instantiation level or bitrate delta for a scaling step.

6.2.42.3 Definition

The syntax of the `VirtualLinkBitrateLevel` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VirtualLinkBitrateLevel:
  derived_from: tosca.datatypes.Root
  description: Describes bitrate requirements applicable to the virtual link
instantiated from a particular VnfVirtualLink
  properties:
    bitrate_requirements:
      type: tosca.datatypes.nfv.LinkBitrateRequirements
      description: Virtual link bitrate requirements for an instantiation level or
bitrate delta for a scaling step
      required: true
```

6.2.42.4 Examples

See clause A.6.

6.2.42.5 Additional Requirements

None.

6.2.43 tosca.datatypes.nfv.VnfOperationAdditionalParameters

6.2.43.1 Description

The VnfOperationAdditionalParameters data type is an empty base type for deriving data type for describing VNF-specific parameters to be passed when invoking lifecycle management operations as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.43.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.43.1-1: Type name, shorthand, and URI

Shorthand Name	VnfOperationAdditionalParameters
Type Qualified Name	toscanfv:VnfOperationAdditionalParameters
Type URI	tosca.datatypes.nfv.VnfOperationAdditionalParameters

6.2.43.2 Properties

None.

6.2.43.3 Definition

The syntax of the VnfOperationAdditionalParameters data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfOperationAdditionalParameters:
  derived_from: tosca.datatypes.Root
  description: Is an empty base type for deriving data type for describing VNF-
specific parameters to be passed when invoking lifecycle management operations
  #properties:
```

6.2.43.4 Examples

```
tosca_definitions_version: tosca_simple_yaml_1_3

node_types:
  MyCompany.nodes.nfv.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      ..
    interfaces:
      Vnflcm:
        operations:
          instantiate:
            inputs:
              additional_parameters:
                type: MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters

data_types:
  MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters:
    derived_from: tosca.datatypes.nfv.VnfOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
        required: true
      parameter_2:
        type: string
        required: true
        default: value_2
```

The following example shows the declaration of an additional parameter to receive a password in the instantiate operation. The metadata sensitive: "true" indicates that this property holds security-sensitive information.

```
tosca_definitions_version: tosca_simple_yaml_1_3

node_types:
  MyCompany.nodes.nfv.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      ..
    interfaces:
      Vnflcm:
        operations:
          instantiate:
            inputs:
              additional_parameters:
                type: MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters

data_types:
  MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters:
    derived_from: tosca.datatypes.nfv.VnfOperationAdditionalParameters
    properties:
      inst_password:
        type: string
        required: true
        metadata:
          sensitive: "true"
```

6.2.43.5 Additional Requirements

None.

6.2.44 tosca.datatypes.nfv.VnfChangeFlavourOperationConfiguration

6.2.44.1 Description

The VnfChangeFlavourOperationConfiguration data type represents information that affect the invocation of the ChangeVnfFlavour operation, as specified in ETSI GS NFV-IFA 011 [1]. This data type definition is reserved for future use in the present document. Table 6.2.44.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.44.1-1: Type name, shorthand, and URI

Shorthand Name	VnfChangeFlavourOperationConfiguration
Type Qualified Name	toscanfv:VnfChangeFlavourOperationConfiguration
Type URI	tosca.datatypes.nfv.VnfChangeFlavourOperationConfiguration

6.2.44.2 Properties

None.

6.2.44.3 Definition

The syntax of the VnfChangeFlavourOperationConfiguration data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfChangeFlavourOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the
ChangeVnfFlavour operation
  # This data type definition is reserved for future use in the present document.
  # properties:
```

6.2.44.4 Examples

None.

6.2.44.5 Additional Requirements

None.

6.2.45 toska.datatypes.nfv.VnfChangeExtConnectivityOperation Configuration

6.2.45.1 Description

The VnfChangeExtConnectivityOperationConfiguration data type represents information that affect the invocation of the ChangeExtVnfConnectivity operation, as specified in ETSI GS NFV-IFA 011 [1]. This data type definition is reserved for future use in the present document. Table 6.2.45.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.45.1-1: Type name, shorthand, and URI

Shorthand Name	VnfChangeExtConnectivityOperationConfiguration
Type Qualified Name	toscanfv:VnfChangeExtConnectivityOperationConfiguration
Type URI	tosca.datatypes.nfv.VnfChangeExtConnectivityOperationConfiguration

6.2.45.2 Properties

None.

6.2.45.3 Definition

The syntax of the VnfChangeExtConnectivityOperationConfiguration data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfChangeExtConnectivityOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the
ChangeExtVnfConnectivity operation
  # This data type definition is reserved for future use in the present document.
  # properties:
```


6.2.45.4 Examples

None.

6.2.45.5 Additional Requirements

None.

6.2.46 `tosca.datatypes.nfv.VnfMonitoringParameter`

The `VnfMonitoringParameter` data type is defined in clause 9.2.9 of the present document.

6.2.47 `tosca.datatypes.nfv.VnfcMonitoringParameter`

6.2.47.1 Description

This data type provides information on virtualised resource related performance metrics applicable to a VNFC. Table 6.2.47.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.47.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfcMonitoringParameter</code>
Type Qualified Name	<code>toscanfv:VnfcMonitoringParameter</code>
Type URI	<code>tosca.datatypes.nfv.VnfcMonitoringParameter</code>

6.2.47.2 Properties

The properties of the `VnfcMonitoringParameter` data type shall comply with the provisions set out in table 6.2.47.2-1.

Table 6.2.47.2-1: Properties

Name	Required	Type	Constraints	Description
name	no	string		Human readable name of the monitoring parameter.
performance_metric	yes	string	Valid values: See YAML definition constraints	Identifies a performance metric to be monitored. Performance metric values shall be either set to: • A corresponding measurement name defined in clause 7.2 of ETSI GS NFV-IFA 027 [7], without appending a sub-counter. In this case the VNFM computes these measurements from lower-level metrics collected from the VIM. See note. • A corresponding measurement name defined in clause 7.1 of ETSI GS NFV-IFA 027 [7], without appending a sub-counter. In this case the VNFM collects these metrics from the VIM for all compute, storage and network resources allocated to the VNFC instance.
collection_period	no	scalar-unit.time		Describes the periodicity at which to collect the performance information.
NOTE: The measured object type for _cpu_usage_mean_vnf, v_cpu_usage_peak_vnf, v_memory_usage_mean_vnf, v_memory_usage_peak_vnf, v_disk_usage_mean_vnf and v_disk_usage_peak_vnf is the VNFC.				

6.2.47.3 Definition

The syntax of the VnfcMonitoringParameter data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfcMonitoringParameter:
  derived_from: toska.datatypes.Root
  description: Represents information on virtualised resource related performance
metrics applicable to the VNF.
  properties:
    name:
      type: string
      description: Human readable name of the monitoring parameter
      required: true
    performance_metric:
      type: string
      description: Identifies a performance metric to be monitored, according to
ETSI GS NFV-IFA 027.
      required: true
      constraints:
        - valid_values: [ v_cpu_usage_mean_vnf, v_cpu_usage_peak_vnf,
v_memory_usage_mean_vnf, v_memory_usage_peak_vnf, v_disk_usage_mean_vnf,
v_disk_usage_peak_vnf, byte_incoming_vnf_int_cp, byte_outgoing_vnf_int_cp,
packet_incoming_vnf_int_cp, packet_outgoing_vnf_int_cp, v_cpu_usage_mean,
v_cpu_usage_peak, v_memory_usage_mean, v_memory_usage_peak, v_disk_usage_mean,
v_disk_usage_peak, v_net_byte_incoming, v_net_byte_outgoing, v_net_packet_incoming,
v_net_packet_outgoing, usage_mean_vStorage, usage_peak_vStorage ]
    collection_period:
      type: scalar-unit.time

```

```

    description: Describes the periodicity at which to collect the performance
    information.
    required: false
    constraints:
      - greater_than: 0 s

```

6.2.47.4 Examples

None.

6.2.47.5 Additional Requirements

None.

6.2.48 `tosca.datatypes.nfv.VirtualLinkMonitoringParameter`

6.2.48.1 Description

This data type provides information on virtualised resource related performance metrics. Table 6.2.48.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.48.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VirtualLinkMonitoringParameter</code>
Type Qualified Name	<code>tosca.nfv:VirtualLinkMonitoringParameter</code>
Type URI	<code>tosca.datatypes.nfv.VirtualLinkMonitoringParameter</code>

6.2.48.2 Properties

The properties of the `VirtualLinkMonitoringParameter` data type shall comply with the provisions set out in table 6.2.48.2-1.

Table 6.2.48.2-1: Properties

Name	Required	Type	Constraints	Description
<code>name</code>	no	string		Human readable name of the monitoring parameter.
<code>performance_metric</code>	yes	string	Valid values: See YAML definition constraints	Identifies a performance metric to be monitored. Performance metric values shall be set to following measurement names defined in clause 7.1 of ETSI GS NFV-IFA 027 [7], without appending a sub-counter: • <code>ByteIncoming</code> • <code>ByteOutgoing</code> • <code>PacketIncoming</code> • <code>PacketOutgoing</code> The VNFM collects these metrics from the VIM by aggregating the sub-counters of all virtual link ports attached to the virtual link to which the metrics apply.
<code>collection_period</code>	no	scalar-unit.time		Describes the periodicity at which to collect the performance information.

6.2.48.3 Definition

The syntax of the VirtualLinkMonitoringParameter data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualLinkMonitoringParameter:
  derived_from: toska.datatypes.Root
  description: Represents information on virtualised resource related performance
metrics applicable to the VNF.
  properties:
    name:
      type: string
      description: Human readable name of the monitoring parameter
      required: true
    performance_metric:
      type: string
      description: Identifies a performance metric to be monitored.
      required: true
      constraints:
        - valid_values: [ byte_incoming, byte_outgoing, packet_incoming,
packet_outgoing ]
    collection_period:
      type: scalar-unit.time
      description: Describes the periodicity at which to collect the performance
information.
      required: false
      constraints:
        - greater_than: 0 s

```

6.2.48.4 Examples

None.

6.2.48.5 Additional Requirements

None.

6.2.49 toska.datatypes.nfv.InterfaceDetails

6.2.49.1 Description

The InterfaceDetails data type describes information used to access an interface exposed by a VNF. It corresponds to the interfaceDetails attribute of the VnfInterfaceDetails information element defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.49.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.49.1-1: Type name, shorthand, and URI

Shorthand Name	InterfaceDetails
Type Qualified Name	toscanfv:InterfaceDetails
Type URI	tosca.datatypes.nfv.InterfaceDetails

6.2.49.2 Properties

The properties of the InterfaceDetails data type shall comply with the provisions set out in table 6.2.49.2-1.

Table 6.2.49.2-1: Properties

Name	Required	Type	Constraints	Description
uri_components	no	tosca.datatypes.nfv.UriComponents		Provides components to build a Uniform Resource Identifier (URI) where to access the interface end point.
interface_specific_data	no	map of string		Provides additional details that are specific to the type of interface considered.

6.2.49.3 Definition

The syntax of the InterfaceDetails data type shall comply with the following definition:

```

tosca.datatypes.nfv.InterfaceDetails:
  derived_from: toska.datatypes.Root
  description: information used to access an interface exposed by a VNF
  properties:
    uri_components:
      type: toska.datatypes.nfv.UriComponents
      description: Provides components to build a Uniform Resource Identifier (URI)
where to access the interface end point.
      required: false
    interface_specific_data:
      type: map
      description: Provides additional details that are specific to the type of
interface considered.
      required: false
    entry_schema:
      type: string

```

6.2.49.4 Examples

See clause 6.10.12.

6.2.49.5 Additional Requirements

None.

6.2.50 toska.datatypes.nfv.UriComponents

6.2.50.1 Description

The UriComponents data type describes information used to build a URI that complies with IETF RFC 3986 [8]. Table 6.2.50.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.50.1-1: Type name, shorthand, and URI

Shorthand Name	UriComponents
Type Qualified Name	toscanfv:UriComponents
Type URI	tosca.datatypes.nfv.UriComponents

6.2.50.2 Properties

The properties of the UriComponents data type shall comply with the provisions set out in table 6.2.50.2-1.

Table 6.2.50.2-1: Properties

Name	Required	Type	Constraints	Description
scheme	yes	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the scheme component of a URI, as per IETF RFC 3986 [8]
authority	no	tosca.datatypes.nfv.UriAuthority		Corresponds to the authority component of a URI, as per IETF RFC 3986 [8] See note
path	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the path component of a URI, as per IETF RFC 3986 [8]
query	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the query component of a URI, as per IETF RFC 3986 [8]
fragment	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the fragment component of a URI, as per IETF RFC 3986 [8]
NOTE: If this property is not included while the URI scheme requires it, the VNFM is expected to generate it, based on knowledge of the network configuration of the external CP instance that provides the connectivity for this interface.				

6.2.50.3 Definition

The syntax of the UriComponents data type shall comply with the following definition:

```

tosca.datatypes.nfv.UriComponents:
  derived_from: toska.datatypes.Root
  description: information used to build a URI that complies with IETF RFC 3986.
  properties:
    scheme:
      type: string # shall comply with IETF RFC 3986
      description: scheme component of a URI.
      required: true
    authority:
      type: toska.datatypes.nfv.UriAuthority
      description: Authority component of a URI
      required: false
    path:
      type: string # shall comply with IETF RFC 3986
      description: path component of a URI.
      required: false
    query:
      type: string # shall comply with IETF RFC 3986
      description: query component of a URI.
      required: false
    fragment:
      type: string # shall comply with IETF RFC 3986
      description: fragment component of a URI.
      required: false

```

6.2.50.4 Examples

See clause 6.10.12.

6.2.50.5 Additional Requirements

When this datatype is used to provide information for accessing APIs defined in ETSI GS NFV-SOL 002 [22], the path property may be included and the query and fragment properties shall be absent. The values of the scheme, authority and path properties form the {apiRoot} of the URI prefix.

6.2.51 toska.datatypes.nfv.UriAuthority

6.2.51.1 Description

The UriAuthority data type corresponds to the authority component of a URI as specified in IETF RFC 3986 [8]. Table 6.2.51.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.51.1-1: Type name, shorthand, and URI

Shorthand Name	UriAuthority
Type Qualified Name	toscanfv:UriAuthority
Type URI	tosca.datatypes.nfv.UriAuthority

6.2.51.2 Properties

The properties of the UriAuthority data type shall comply with the provisions set out in table 6.2.51.2-1.

Table 6.2.51.2-1: Properties

Name	Required	Type	Constraints	Description
user_info	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the user_info field of the authority component of a URI, as per IETF RFC 3986 [8]. For HTTP and HTTPS URIs, the provisions in sections 4.2.1, 4.2.2 and 4.2.4 of IETF RFC 9110 [24] apply, respectively.
host	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the host field of the authority component of a URI, as per IETF RFC 3986 [8]. See note 1.
port	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the port field of the authority component of a URI, as per IETF RFC 3986 [8]. See note 2.
NOTE 1: If this property is not included the VNFM is expected to generate it, based on knowledge of the network configuration of the external CP instance that provides the connectivity for this interface.				
NOTE 2: If this property is not included the default port for the protocol declared by the scheme property of the parent UriComponents structure shall be used unless there are configuration mechanisms applied that are outside the scope of the present document. If no default port exists for the URI scheme, the port property shall be included unless there are configuration mechanisms applied that are outside the scope of the present document.				

6.2.51.3 Definition

The syntax of the UriAuthority data type shall comply with the following definition:

```
tosca.datatypes.nfv.UriAuthority:
  derived_from: toska.datatypes.Root
  description: information that corresponds to the authority component of a URI as
specified in IETF RFC 3986
  properties:
    user_info:
      type: string # shall comply with IETF RFC 3986
      description: user_info field of the authority component of a URI
      required: false
    host:
      type: string # shall comply with IETF RFC 3986
      description: host field of the authority component of a URI
      required: false
```

```

port:
  type: string # shall comply with IETF RFC 3986
  description: port field of the authority component of a URI
  required: false

```

6.2.51.4 Examples

See clause 6.10.12.

6.2.51.5 Additional Requirements

When this datatype is used to provide information for accessing APIs defined in ETSI GS NFV-SOL 002 [22], the host property and port properties may be included and the user_info property shall not be included. If the host property is included and the value is a registered name, it is assumed that means are in place to resolve the host name to the correct IP address. If the host property is not included, it is assumed that the VNFM will use the IP address associated to one of the connection point instances created from the VnfExpCp and VduCp node types declared as a target of the SupportedVnfInterface policy.

NOTE: This means that if multiple CP instances exist that were created from a particular VnfExtCp or VduCp node template, the VNFM may use any of them to attempt accessing the interface. If no reply is received because the selected CP instance is out of service or is not reachable, the VNFM is expected to try reaching the interface through another CP instance.

6.2.52 `tosca.datatypes.nfv.VnfProfile`

6.2.52.1 Description

The VnfProfile data type is defined in clause 9.2.8 of the present document.

6.2.53 `tosca.datatypes.nfv.ChecksumData`

6.2.53.1 Description

The ChecksumData data type describes information about the result of performing a checksum operation over some arbitrary data. Table 6.2.53.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.53.1-1: Type name, shorthand, and URI

Shorthand Name	ChecksumData
Type Qualified Name	toscanfv:ChecksumData
Type URI	tosca.datatypes.nfv.ChecksumData

6.2.53.2 Properties

The properties of the ChecksumData data type shall comply with the provisions set out in table 6.2.53.2-1.

Table 6.2.53.2-1: Properties

Name	Required	Type	Constraints	Description
algorithm	yes	string	Valid values: See YAML definition constraints	Describes the algorithm used to obtain the checksum value, as described in [14].
hash	yes	string		Contains the result of applying the algorithm indicated by the algorithm property to the data to which this ChecksumData refers.

6.2.53.3 Definition

The syntax of the ChecksumData data type shall comply with the following definition:

```
tosca.datatypes.nfv.ChecksumData:
  derived_from: toska.datatypes.Root
  description: Describes information about the result of performing a checksum
operation over some arbitrary data
  properties:
    algorithm:
      type: string
      description: Describes the algorithm used to obtain the checksum value
      required: true
      constraints:
        - valid_values: [sha-224, sha-256, sha-384, sha-512 ]
    hash:
      type: string
      description: Contains the result of applying the algorithm indicated by the
algorithm property to the data to which this ChecksumData refers
      required: true
```

6.2.53.4 Examples

```
<some_tosca_entity>:
  properties:
    checksum:
      algorithm: sha-256
      hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d
```

6.2.53.5 Additional Requirements

None.

6.2.54 toska.datatypes.nfv.VnfmInterfaceInfo

6.2.54.1 Description

The VnfmInterfaceInfo data type describes information enabling the VNF instance to access the NFV-MANO interfaces produced by the VNFM (e.g. URIs and credentials). Table 6.2.54.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.54.1-1: Type name, shorthand, and URI

Shorthand Name	VnfmInterfaceInfo
Type Qualified Name	toscanfv.VnfmInterfaceInfo
Type URI	tosca.datatypes.nfv.VnfmInterfaceInfo

6.2.54.2 Properties

The properties of the VnfmInterfaceInfo data type shall comply with the provisions set out in table 6.2.54.2-1.

Table 6.2.54.2-1: Properties

Name	Required	Type	Constraints	Description
interface_name	yes	string	Valid values vnf_lcm vnf_pm vnf_fm	Identifies an interface produced by the VNFM.
details	no	tosca.datatypes.nfv.InterfaceDetails		Provide additional data to access the interface endpoint (e.g. API URI prefix).
credentials	no	map of string		Provides credential enabling access to the interface. This property is reserved for future use in the present document.

6.2.54.3 Definition

The syntax of the VnfmInterfaceInfo data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfmInterfaceInfo:
  derived_from: toska.datatypes.Root
  description: Describes information enabling the VNF instance to access the NFV-
MANO interfaces produced by the VNFM
  properties:
    interface_name:
      type: string
      description: Identifies an interface produced by the VNFM.
      required: true
      constraints:
        - valid_values: [ vnf_lcm, vnf_pm, vnf_fm ]
    details:
      type: toska.datatypes.nfv.InterfaceDetails
      description: Provide additional data to access the interface endpoint
      required: false
    credentials:
      type: map
      description: Provides credential enabling access to the interface
      required: false
  entry_schema:
    type: string

```

6.2.54.4 Examples

None.

6.2.54.5 Additional Requirements

None.

6.2.55 toska.datatypes.nfv.OauthServerInfo

6.2.55.1 Description

The OauthServerInfo data type describes information to enable discovery of the authorization server. This data type definition is reserved for future use in the present document. Table 6.2.55.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.55.1-1: Type name, shorthand, and URI

Shorthand Name	OauthServerInfo
Type Qualified Name	toscanfv:OauthServerInfo
Type URI	tosca.datatypes.nfv.OauthServerInfo

6.2.55.2 Properties

None.

6.2.55.3 Definition

The syntax of the OauthServerInfo data type shall comply with the following definition:

```
tosca.datatypes.nfv.OauthServerInfo:
  derived_from: toscanfv.Root
  description: information to enable discovery of the authorization server
  #properties:
  #This data type definition is reserved for future use in the present document
```

6.2.55.4 Examples

None.

6.2.55.5 Additional Requirements

None.

6.2.56 toscanfv.BootData

6.2.56.1 Description

The BootData data type describes the information used to customize a virtualised compute resource at boot time. Table 6.2.56.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.56.1-1: Type name, shorthand, and URI

Shorthand Name	BootData
Type Qualified Name	toscanfv:BootData
Type URI	tosca.datatypes.nfv.BootData

6.2.56.2 Properties

The properties of the BootData data type shall comply with the provisions set out in table 6.2.56.2-1.

Table 6.2.56.2-1: Properties

Name	Required	Type	Constraints	Description
vim_specific_properties	no	tosca.datatypes.nfv.BootDataVimSpecificProperties		Properties used for selecting VIM specific capabilities when setting the boot data.
kvp_data	no	tosca.datatypes.nfv.KvpData		A set of key-value pairs for configuring a virtual compute resource. The mechanisms for conveying these key-value pairs to the virtual compute resource are out of the scope of the present document. An example of such mechanisms is the OpenStack metadata service defined in [i.13].
content_or_file_data	no	tosca.datatypes.nfv.ContentOrFileData		A string content or a file for configuring a virtual compute resource. The mechanisms for conveying the string content or the file to the virtual compute resource are out of the scope of the present document. An example of such mechanisms is the OpenStack User-data service defined in [i.14].

6.2.56.3 Definition

The syntax of the BootData data type shall comply with the following definition:

```

tosca.datatypes.nfv.BootData:
  derived_from: tosca.datatypes.Root
  description: describes the information used to customize a virtualised compute
resource at boot time.
  properties:
    vim_specific_properties:
      type: tosca.datatypes.nfv.BootDataVimSpecificProperties
      description: Properties used for selecting VIM specific capabilities when
setting the boot data.
      required: false
    kvp_data:
      type: tosca.datatypes.nfv.KvpData
      description: A set of key-value pairs for configuring a virtual compute
resource.
      required: false
    content_or_file_data:
      type: tosca.datatypes.nfv.ContentOrFileData
      description: A string content or a file for configuring a virtual compute
resource.
      required: false

```

6.2.56.4 Examples

See clause 6.8.3.8.

6.2.56.5 Additional Requirements

None.

6.2.57 tosca.datatypes.nfv.KvpData

6.2.57.1 Description

The KvpData data type describes a set of key-value pairs information used to customize a virtualised compute resource by using only key-value pairs data. Table 6.2.57.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.57.1-1: Type name, shorthand, and URI

Shorthand Name	KvpData
Type Qualified Name	toscanfv:KvpData
Type URI	tosca.datatypes.nfv.KvpData

6.2.57.2 Properties

The properties of the MetaData data type shall comply with the provisions set out in table 6.2.57.2-1.

Table 6.2.57.2-1: Properties

Name	Required	Type	Constraints	Description
data	no	map of string		A map of strings that contains a set of key-value pairs that describes the information for configuring the virtualised compute resource.

6.2.57.3 Definition

The syntax of the KvpData data type shall comply with the following definition:

```

tosca.datatypes.nfv.KvpData:
  derived_from: tosca.datatypes.Root
  description: describes a set of key-value pairs information used to customize
a virtualised compute resource at boot time by using only key-value pairs data.
  properties:
    data:
      type: map
      description: A map of strings that contains a set of key-value pairs that
describes the information for configuring the virtualised compute resource.
      required: false
      entry_schema:
        type: string

```

6.2.57.4 Examples

See clause 6.8.3.8.

6.2.57.5 Additional Requirements

None.

6.2.58 tosca.datatypes.nfv.ContentOrFileData

6.2.58.1 Description

The ContentOrFileData data type describes a string content or a file information used to customize a virtualised compute resource by using string content or file. Table 6.2.58.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.58.1-1: Type name, shorthand, and URI

Shorthand Name	ContentOrFileData
Type Qualified Name	toscanfv:ContentOrFileData
Type URI	tosca.datatypes.nfv.ContentOrFileData

6.2.58.2 Properties

The properties of the ContentOrFileData data type shall comply with the provisions set out in table 6.2.58.2-1.

Table 6.2.58.2-1: Properties

Name	Required	Type	Constraints	Description
data	no	map of string		A map of strings that contains a set of key-value pairs that carries the dynamic deployment values which used to replace the corresponding variable parts in the file as identify by a URL as described in source_path. Shall be present if "source_path" is present and shall be absent otherwise. See note 1.
content	no	string		The string information used to customize a virtualised compute resource at boot time. See note 2.
source_path	no	string		The URL to a file contained in the VNF package used to customize a virtualised compute resource. The content shall comply with IETF RFC 3986 [8]. See note 2.
destination_path	no	string		The URL where to inject a file indicated by the source_path property into the virtualised compute resource. The content shall comply with IETF RFC 3986 [8]. See note 3.
NOTE 1: It is the file processor (e.g. in the VNFM) responsibility to replace the corresponding variable parts in the file with the value carried in the data property, the variable parts in the file are start with \$ and end with \$, its content is the same character with one of the keys in the data property, for example, if one of the keys in 'data' is "https_proxy", somewhere in the file content there is \$https_proxy\$. NOTE 2: One and only one of the following properties shall be present: contents or source_path. NOTE 3: It is only present when a particular method is used for transferring boot information into a virtualised compute resource and source_path is also present. For example, such method can be the personality method as described in [i.15], and it has been deprecated since Openstack 12.0.0 (Stein).				

6.2.58.3 Definition

The syntax of the ContentOrFileData data type shall comply with the following definition:

```
tosca.datatypes.nfv.ContentOrFileData:
  derived_from: tosca.datatypes.Root
  description: describes a string content or a file information used to
  customize a virtualised compute resource at boot time by using string content or
  file.
```

```

properties:
  data:
    type: map
    description: A map of strings that contains a set of key-value pairs that
carries the dynamic deployment values which used to replace the corresponding
variable parts in the file as identify by a URL as described in source_path. Shall
be present if "source_path" is present and shall be absent otherwise..
    required: false
    entry_schema:
      type: string
    content:
      type: string
      description: The string information used to customize a virtualised
compute resource at boot time.
      required: false
    source_path:
      type: string
      description: The URL to a file contained in the VNF package used to
customize a virtualised compute resource. The content shall comply with IETF
RFC 3986.
      required: false
    destination_path:
      type: string
      description: The URL address when inject a file into the virtualised
compute resource. The content shall comply with IETF RFC 3986.
      required: false

```

6.2.58.4 Examples

See clause 6.8.3.8.

6.2.58.5 Additional Requirements

None.

6.2.59 `tosca.datatypes.nfv.BootDataVimSpecificProperties`

6.2.59.1 Description

The `BootDataVimSpecificProperties` data type describes the VIM related information used for selecting VIM specific capabilities when setting the boot data when setting the boot data. Table 6.2.59.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.59.1-1: Type name, shorthand, and URI

Shorthand Name	<code>BootDataVimSpecificProperties</code>
Type Qualified Name	<code>toscanfv: BootDataVimSpecificProperties</code>
Type URI	<code>tosca.datatypes.nfv.BootDataVimSpecificProperties</code>

6.2.59.2 Properties

The properties of the `BootDataVimSpecificProperties` data type shall comply with the provisions set out in table 6.2.59.2-1.

Table 6.2.59.2-1: Properties

Name	Required	Type	Constraints	Description
vim_type	no	string		Discriminator for the different types of the VIM information. The set of permitted values is expected to change over time as new types or versions of VIMs become available. The ETSI NFV registry of VIM-related information [i.16] provides access to information about various VIM types.
properties	yes	map of string		Properties used for selecting VIM specific capabilities when setting the boot data. For example, it can set whether config_drive functionality is selected in case VIM support it. This property is reserved for future use in the present document.

6.2.59.3 Definition

The syntax of the BootDataVimSpecificProperties data type shall comply with the following definition:

```

tosca.datatypes.nfv.BootDataVimSpecificProperties:
  derived_from: tosca.datatypes.Root
  description: describes the VIM specific information used for selecting VIM
specific capabilities when setting the boot data.
  properties:
    vim_type:
      type: string
      description: Discriminator for the different types of the VIM information.
      required: false
    properties:
      type: map
      description: Properties used for selecting VIM specific capabilities when
setting the boot data
      entry_schema:
        type: string
        required: true

```

6.2.59.4 Examples

None.

6.2.59.5 Additional Requirements

None.

6.2.60 tosca.datatypes.nfv.VnfPackageChangeSelector

6.2.60.1 Description

The VnfPackageChangeSelector data type describes the source and destination VNFDs as well as source deployment flavour for a change current VNF Package. Table 6.2.60.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.60.1-1: Type name, shorthand, and URI

Shorthand Name	VnfPackageChangeSelector
Type Qualified Name	toscanfv: VnfPackageChangeSelector
Type URI	tosca.datatypes.nfv.VnfPackageChangeSelector

6.2.60.2 Properties

The properties of the VnfPackageChangeSelector data type shall comply with the provisions set out in table 6.2.60.2-1.

Table 6.2.60.2-1: Properties

Name	Required	Type	Constraints	Description
source_descriptor_id	yes	string		Identifier of the source VNFD and the source VNF package.
destination_descriptor_id	yes	string		Identifier of the destination VNFD and the destination VNF package.
source_flavour_id	yes	string		Identifier of the deployment flavour in the source VNF package for which this data type applies.

6.2.60.3 Definition

The syntax of the VnfPackageChangeSelector data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfPackageChangeSelector:
  derived_from: toska.datatypes.Root
  description: data type describes the source and destination VNFDs as well as
source deployment flavour for a change current VNF Package.
  properties:
    source_descriptor_id:
      type: string
      description: Identifier of the source VNFD and the source VNF package.
      required: true
    destination_descriptor_id:
      type: string
      description: Identifier of the destination VNFD and the destination VNF
package.
      required: true
    source_flavour_id:
      type: string
      description: Identifier of the deployment flavour in the source VNF
package for which this data type applies.
      required: true

```

6.2.60.4 Examples

See clause 6.10.15.5.

6.2.60.5 Additional Requirements

Either the source_descriptor_id or the destination_descriptor_id shall be equal to the vnfdId of the VNFD containing this version VnfPackageChangeSelector.

6.2.61 toska.datatypes.nfv.VnfPackageChangeComponentMapping

6.2.61.1 Description

The VnfPackageChangeComponentMapping data type describes a mapping between the identifier of a components or property in the source VNFD and the identifier of the corresponding component or property in the destination VNFD. Table 6.2.61.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.61.1-1: Type name, shorthand, and URI

Shorthand Name	VnfPackageChangeComponentMapping
Type Qualified Name	toscanfv: VnfPackageChangeComponentMapping
Type URI	tosca.datatypes.nfv.VnfPackageChangeComponentMapping

6.2.61.2 Properties

The properties of the VnfPackageChangeComponentMapping data type shall comply with the provisions set out in table 6.2.61.2-1.

Table 6.2.61.2-1: Properties

Name	Required	Type	Constraints	Description
component_type	yes	string	valid_values: [vdu, cp, virtual_link, virtual_storage, instantiation_level, scaling_aspect, deployable_module]	The type of component or property. Possible values differentiate whether changes concern to some VNF component (e.g. VDU, internal VLD, etc.) or property (e.g. a Scaling Aspect, etc.).
source_id	yes	string		Identifier of the component or property in the source VNFD.
destination_id	yes	string		Identifier of the component or property in the destination VNFD.
description	no	string		Human readable description of the component changes.

6.2.61.3 Definition

The syntax of the VnfPackageChangeComponentMapping data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfPackageChangeComponentMapping:
  derived_from: toska.datatypes.Root
  description: A mapping between the identifier of a components or property in
the source VNFD and the identifier of the corresponding component or property in
the destination VNFD.
  properties:
    component_type:
      type: string
      description: The type of component or property. Possible values
differentiate whether changes concern to some VNF component (e.g. VDU, internal
VLD, etc.) or property (e.g. a Scaling Aspect, etc.).
    constraints:
      - valid_values: [ vdu, cp, virtual_link, virtual_storage,
instantiation_level, scaling_aspect, deployable_module ]
      required: true
    source_id:
      type: string
```

```

    description: Identifier of the component or property in the source VNFD.
    required: true
  destination_id:
    type: string
    description: Identifier of the component or property in the destination
VNFD.
    required: true
  description:
    type: string
    description: Human readable description of the component changes.
    required: false

```

6.2.61.4 Examples

See clause 6.10.15.5.

6.2.61.5 Additional Requirements

None.

6.2.62 `tosca.datatypes.nfv.VnfChangeCurrentPackageOperation` Configuration

6.2.62.1 Description

The `VnfChangeCurrentPackageOperationConfiguration` data type represents information that affect the invocation of the change current VNF Package operation, as specified in ETSI GS NFV-IFA 011 [1]. This data type definition is reserved for future use in the present document. Table 6.2.62.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.62.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfChangeCurrentPackageOperationConfiguration</code>
Type Qualified Name	<code>toscanfv.VnfChangeCurrentPackageOperationConfiguration</code>
Type URI	<code>tosca.datatypes.nfv.VnfChangeCurrentPackageOperationConfiguration</code>

6.2.62.2 Properties

None.

6.2.62.3 Definition

The syntax of the `VnfChangeCurrentPackageOperationConfiguration` data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfChangeCurrentPackageOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the change
current VNF Package operation.
  # This data type definition is reserved for future use in the present document.
  # properties:
  #   # derived types are expected to introduce new properties, with their type
derived from toska.datatypes.nfv.VnfChangeCurrentPackageOperationConfiguration, with
the same name as the operation designated to the ChangeCurrentVnfPackage request

```

6.2.62.4 Examples

None.

6.2.62.5 Additional Requirements

None.

6.2.63 `tosca.datatypes.nfv.VnfCreateSnapshotOperationConfiguration`

6.2.63.1 Description

The `VnfCreateSnapshotOperationConfiguration` data type represents information that affect the invocation of the `CreateVnfSnapshot` operation, as specified in ETSI GS NFV-IFA 011 [1]. This data type definition is reserved for future use in the present document. Table 6.2.63.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.63.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfCreateSnapshotOperationConfiguration</code>
Type Qualified Name	<code>toscanfv:VnfCreateSnapshotOperationConfiguration</code>
Type URI	<code>tosca.datatypes.nfv.VnfCreateSnapshotOperationConfiguration</code>

6.2.63.2 Properties

None.

6.2.63.3 Definition

The syntax of the `VnfCreateSnapshotOperationConfiguration` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfCreateSnapshotOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the
CreateVnfSnapshot operation
  # This data type definition is reserved for future use in the present document.
  # properties:
```

6.2.63.4 Examples

None.

6.2.63.5 Additional Requirements

None.

6.2.64 `tosca.datatypes.nfv.VnfRevertToSnapshotOperationConfiguration`

6.2.64.1 Description

The `VnfRevertToSnapshotOperationConfiguration` data type represents information that affect the invocation of the `RevertToVnfSnapshot` operation, as specified in ETSI GS NFV-IFA 011 [1]. This data type definition is reserved for future use in the present document. Table 6.2.64.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.64.1-1: Type name, shorthand, and URI

Shorthand Name	VnfRevertToSnapshotOperationConfiguration
Type Qualified Name	toscanfv:VnfRevertToSnapshotOperationConfiguration
Type URI	tosca.datatypes.nfv.VnfRevertToSnapshotOperationConfiguration

6.2.64.2 Properties

None.

6.2.64.3 Definition

The syntax of the VnfRevertToSnapshotOperationConfiguration data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfRevertToSnapshotOperationConfiguration:
  derived_from: toscanfv.Root
  description: represents information that affect the invocation of the
RevertToVnfSnapshot operation
  # This data type definition is reserved for future use in the present document.
  # properties:
```

6.2.64.4 Examples

None.

6.2.64.5 Additional Requirements

None.

6.2.65 toscanfv.ServicePortData

6.2.65.1 Description

The ServicePortData data type supports the specification of requirements describing port properties exposed by VirtualCp, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.65.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.65.1-1: Type name, shorthand, and URI

Shorthand Name	ServicePortData
Type Qualified Name	toscanfv:ServicePortData
Type URI	tosca.datatypes.nfv.ServicePortData

6.2.65.2 Properties

The properties of the ServicePortData data type shall comply with the provisions set out in table 6.2.65.2-1.

Table 6.2.65.2-1: Properties

Name	Required	Type	Constraints	Description
name	yes	String		The name of the port exposed by the VirtualCp.
protocol	yes	String	Valid values: See YAML definition constraints	The L4 protocol for this port exposed by the VirtualCp. VALUES: • TCP • UDP • SCTP
port	yes	Integer		The L4 port number exposed by the VirtualCp.
portConfigurable	yes	Boolean		Specifies whether the port attribute value is allowed to be configurable.

6.2.65.3 Definition

The syntax of the ServicePortData data type shall comply with the following definition:

```

tosca.datatypes.nfv.ServicePortData:
  derived_from: toska.datatypes.Root
  description: describes the service identifying port properties exposed by the
VirtualCp
  properties:
    name:
      type: string
      description: The name of the port exposed by the VirtualCp.
      required: true
    protocol:
      type: string
      description: The L4 protocol for this port exposed by the VirtualCp.
      required: true
      constraints:
        - valid_values: [ tcp, udp, sctp ]
    port:
      type: integer
      description: The L4 port number exposed by the VirtualCp.
      required: true
      constraints:
        - greater_or_equal: 0
    portConfigurable:
      type: boolean
      description: Specifies whether the port attribute value is allowed to be
configurable.
      required: true

```

6.2.65.4 Examples

None.

6.2.65.5 Additional Requirements

None.

6.2.66 tosca.datatypes.nfv.AdditionalServiceData

6.2.66.1 Description

The AdditionalServiceData data type supports the specification of requirements related additional service data of the VirtualCp used to expose properties of the VirtualCp to NFV-MANO, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.66.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.66.1-1: Type name, shorthand, and URI

Shorthand Name	AdditionalServiceData
Type Qualified Name	toscanfv:AdditionalServiceData
Type URI	tosca.datatypes.nfv.AdditionalServiceData

6.2.66.2 Properties

The properties of the AdditionalServiceData data type shall comply with the provisions set out in table 6.2.66.2-1.

Table 6.2.66.2-1: Properties

Name	Required	Type	Constraints	Description
portData	yes	List of tosca.datatypes.nfv.ServicePortData		Service port numbers exposed by the VirtualCp.
serviceData	no	tosca.datatypes.nfv.ServiceData		Service matching information exposed by the VirtualCp. See note.
NOTE: This property shall only be present if HTTP/HTTPS based routes to identify the service termination within the VNF is supported, such as for example ingress resource in Kubernetes®.				

6.2.66.3 Definition

The syntax of the AdditionalServiceData data type shall comply with the following definition:

```
tosca.datatypes.nfv.AdditionalServiceData:
  derived_from: tosca.datatypes.Root
  description: describes the additional service data of the VirtualCp used to expose
properties of the VirtualCp to NFV-MANO.
  properties:
    portData:
      type: list
      entry_schema:
        type: tosca.datatypes.nfv.ServicePortData
        description: Service port numbers exposed by the VirtualCp.
        required: true
    serviceData:
      type: tosca.datatypes.nfv.ServiceData
      description: Service matching information exposed by the VirtualCp.
      required: false
```

6.2.66.4 Examples

None.

6.2.66.5 Additional Requirements

None.

6.2.67 `tosca.datatypes.nfv.VnfLcmOpCoord`

6.2.67.1 Description

The `VnfLcmOpCoord` data type describes a set of information used for a coordination action in a VNF lifecycle management operation for a given VNF.

Table 6.2.67.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.67.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfLcmOpCoord</code>
Type Qualified Name	<code>tosca.nfv.VnfLcmOpCoord</code>
Type URI	<code>tosca.datatypes.nfv.VnfLcmOpCoord</code>

6.2.67.2 Properties

The properties of the `VnfLcmOpCoord` data type shall comply with the provisions set out in table 6.2.67.2-1.

Table 6.2.67.2-1: Properties

Name	Required	Type	Constraints	Description
description	no	string		Human readable description of the coordination action.
endpoint_type	no	string	Valid values: See YAML definition constraints	Specifies the type of the endpoint exposing the LCM operation coordination such as operations supporting management systems (e.g. EM) or the VNF instance. valid values: • mgmt: coordination with other operations supporting management systems • vnf: coordination with the VNF instance If the VNF produces the LCM coordination interface, this property may be omitted or may have the value "vnf". If this attribute is omitted, the type of endpoint that provides the interface is determined at deployment time. If the VNF does not produce the LCM coordination interface but coordination via this interface is needed, it is expected that a management entity such as the EM exposes the coordination interface, and consequently, this attribute shall be present and shall have the value "mgmt".
coordination_stage	no	string	Valid values: See YAML definition constraints	Indicates whether the coordination action is invoked before or after all other changes performed by the VNF LCM operation. See note. Valid values: • start: the coordination action is invoked after receiving the grant and before the LCM operation performs any other changes.end: the coordination action is invoked after the LCM operation has performed all other changes. coordination_stage property shall be omitted if the coordination action is intended to be invoked at an intermediate stage of the LCM operation, i.e. neither at the start nor at the end. In this case, the time at which to invoke the coordination during the execution of the LCM operation is determined by means outside the scope of the present document such as VNFM-internal logic or LCM script.
input_parameters	no	tosca.datatypes.nfv.InputOpCoordParams		Input parameters to be provided in the LCM coordination request.
output_parameters	no	tosca.datatypes.nfv.OutputOpCoordParams		Output parameters provided in the LCM coordination response.
NOTE: The changes mentioned include changes to the VNF instance, its resources or its snapshots.				

6.2.67.3 Definition

The syntax of the VnfLcmOpCoord data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfLcmOpCoord:
  derived_from: tosca.datatypes.Root
  description: describes a set of information used for a coordination action in a
VNF lifecycle management operation for a given VNF.
  properties:
    description:
      type: string
      description: Human readable description of the coordination action.
      required: false
    endpoint_type:
      type: string
      description: Specifies the type of the endpoint exposing the LCM operation
coordination such as other operations supporting or management systems (e.g. an EM) or
the VNF instance. If the VNF produces the LCM coordination interface, this property
may be omitted or may have the value "vnf". If this attribute is omitted, the type of
endpoint that provides the interface is determined at deployment time. If the VNF does
not produce the LCM coordination interface but coordination via this interface is
needed, it is expected that a management entity such as the EM exposes the
coordination interface, and consequently, this attribute shall be present and shall
have the value "mgmt".
      required: false
      constraints:
        - valid_values: [ mgmt, vnf ]
    coordination_stage:
      type: string
      description: Indicates whether the coordination action is invoked before or
after all other changes performed by the VNF LCM operation. coordination_stage
property shall be omitted if the coordination action is intended to be invoked at an
intermediate stage of the LCM operation, i.e. neither at the start nor at the end. In
this case, the time at which to invoke the coordination during the execution of the
LCM operation is determined by means outside the scope of the present document such as
VNFM-internal logic or LCM script.
      required: false
      constraints:
        - valid_values: [ start, end ]
    input_parameters:
      type: tosca.datatypes.nfv.InputOpCoordParams
      description: Input parameters to be provided in the LCM coordination request.
      required: false
    output_parameters:
      type: tosca.datatypes.nfv.OutputOpCoordParams
      description: Output parameters provided in the LCM coordination response.
      required: false

```

6.2.67.4 Examples

See clause 6.10.15.

6.2.67.5 Additional Requirements

None.

6.2.68 `tosca.datatypes.nfv.InputOpCoordParams`

6.2.68.1 Description

The `InputOpCoordParams` data type is an empty base type for deriving data types for describing additional input operation coordination parameters for a given coordination action. Table 6.2.68.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.68.1-1: Type name, shorthand, and URI

Shorthand Name	<code>InputOpCoordParams</code>
Type Qualified Name	<code>tosca.nfv.InputOpCoordParams</code>
Type URI	<code>tosca.datatypes.nfv.InputOpCoordParams</code>

6.2.68.2 Properties

None.

6.2.68.3 Definition

The syntax of the `InputOpCoordParams` data type shall comply with the following definition:

```
tosca.datatypes.nfv.InputOpCoordParams:
  derived_from: toska.datatypes.Root
  description: is an empty base type for deriving data types for describing
additional input operation coordination parameters for a given coordination action
```

6.2.69 `tosca.datatypes.nfv.OutputOpCoordParams`

6.2.69.1 Description

The `OutputOpCoordParams` data type is an empty base type for deriving data types for describing additional Output operation coordination parameters for a given coordination action. Table 6.2.69.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.69.1-1: Type name, shorthand, and URI

Shorthand Name	<code>OutputOpCoordParams</code>
Type Qualified Name	<code>tosca.nfv.OutputOpCoordParams</code>
Type URI	<code>tosca.datatypes.nfv.OutputOpCoordParams</code>

6.2.69.2 Properties

None.

6.2.69.3 Definition

The syntax of the `OutputOpCoordParams` data type shall comply with the following definition:

```
tosca.datatypes.nfv.OutputOpCoordParams:
  derived_from: toska.datatypes.Root
  description: is an empty base type for deriving data types for describing
additional Output operation coordination parameters for a given coordination action
```

6.2.70 toska.datatypes.nfv.ExtendedResourceData

6.2.70.1 Description

The ExtendedResourceData data type supports the specification of requirements related to extended resources of a container, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.70.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.70.1-1: Type name, shorthand, and URI

Shorthand Name	ExtendedResourceData
Type Qualified Name	toscanfv:ExtendedResourceData
Type URI	tosca.datatypes.nfv.ExtendedResourceData

6.2.70.2 Properties

The properties of the ExtendedResourceData data type shall comply with the provisions set out in table 6.2.70.2-1.

Table 6.2.70.2-1: Properties

Name	Required	Type	Constraints	Description
extended_resource	yes	map of string	length: 1	The hardware platform specific extended resource. A map of string that contains one single key-value pair that describes one hardware platform specific container requirement. More information regarding the usage of this property is available at: https://register.etsi.org .
amount	yes	integer	greater_than: 0	Requested amount of the indicated extended resource.
amount_valid_values	no	tosca.datatypes.nfv.Integer Range		If this property is present the amount of the extended resource requested for the container can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the set of co-located container compute resources is instantiated with the value indicated in the amount property. If this property is not present the amount of the extended resource requested for the container is not configurable via the VNF LCM interface and is always equal to the value indicated in the amount attribute.

6.2.70.3 Definition

The syntax of the ExtendedResourceData data type shall comply with the following definition:

```
tosca.datatypes.nfv.ExtendedResourceData:
  derived_from: toska.datatypes.Root
  description: Supports the specification of requirements related to extended
resources of a container.
  properties:
    extended_resource:
      type: map
      description: The hardware platform specific extended resource. A map of string
that contains one single key-value pair that describes one hardware platform specific
container requirement.
```

```

    required: true
    entry_schema:
      type: string
    constraints:
      - min_length: 1
      - max_length: 1
  amount:
    type: integer
    description: Requested amount of the indicated extended resource.
    required: true
    constraints:
      - greater_than: 0
  amount_valid_values:
    type: toska.datatypes.nfv.IntegerRange
    description: If this property is present the amount of the extended resource
requested for the container can be indicated in a VNF LCM operation. If no value is
indicated in the VNF LCM operation, the set of co-located container compute resources
is instantiated with the value indicated in the amount property. If this property is
not present the amount of the extended resource requested for the container is not
configurable via the VNF LCM interface and is always equal to the value indicated in
the amount attribute.

```

6.2.70.4 Examples

The following example shows an OsContainer node template that contains requirements for an extended resource.

```

node templates:

  container_01:
    type: toska.nodes.nfv.Vdu.OsContainer
    properties:
      name: db_handler
      description: this container runs the process that writes and reads in the
storage resource.
      requested_cpu_resources: 1
      cpu_resource_limit: 1
      requested_memory_resources: 4 MB
      memory_resource_limit: 8 MB
      extended_resource_requests:
        - extended_resource:
            dongle: |
              mandatory: "true"
            amount: 2
            amount_valid_values: [ 2, 4 ]

```

The example assumes that the capability has been registered in the NFVI Platform Capability Registry, e.g.:

Name	dongle
Permitted Value	base
Version	1.0
Type	Generic
Description	dongle accelerator

6.2.70.5 Additional Requirements

None.

6.2.71 tosca.datatypes.nfv.Hugepages

6.2.71.1 Description

The Hugepages data type supports the specification of requirements on a particular hugepage size in terms of total memory needs. Table 6.2.71.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.71.1-1: Type name, shorthand, and URI

Shorthand Name	Hugepages
Type Qualified Name	toscanfv:Hugepages
Type URI	tosca.datatypes.nfv.Hugepages

6.2.71.2 Properties

The properties of the Hugepages data type shall comply with the provisions set out in table 6.2.71.2-1.

Table 6.2.71.2-1: Properties

Name	Required	Type	Constraints	Description
hugepage_size	yes	scalar-unit.size		Specifies the size of the hugepage.
requested_size	yes	scalar-unit.size		Specifies the total size required for all the hugepages of the size indicated by hugepage_size.
requested_size_valid_values	no	tosca.datatypes.nfv.SizeRange		If this property is present the total size required for all the hugepages of the size indicated by hugepage_size can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the value indicated in the requested_size property is used. If this property is not present the total size required for all the hugepages of the size indicated by hugepage_size is not configurable via the VNF LCM interface and is always equal to the value indicated in the requested_size attribute.

6.2.71.3 Definition

The syntax of the Hugepages data type shall comply with the following definition:

```
tosca.datatypes.nfv.Hugepages:
  derived_from: tosca.datatypes.Root
  description: Supports the specification of requirements on a particular hugepage
size in terms of total memory needs.
  properties:
    hugepage_size:
      type: scalar-unit.size
      description: Specifies the size of the hugepage.
      required: true
    requested_size:
      type: scalar-unit.size
      description: Specifies the total size required for all the hugepages of the
size indicated by hugepage_size.
      required: true
    requested_size_valid_values:
      type: tosca.datatypes.nfv.SizeRange
```

description: If this property is present the the total size required for all the hugepages of the size indicated by hugepage_size can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the value indicated in the requested_size property is used. If this property is not present the total size required for all the hugepages of the size indicated by hugepage_size is not configurable via the VNF LCM interface and is always equal to the value indicated in the requested_size attribute.

required: false

6.2.71.4 Examples

The following example shows an entity using a 'huge_pages_resources' property of `tosca.datatypes.nfv.Hugepages` data type to indicate a request of 100 MiB memory to hold hugepages of size 2 MiB, i.e. for 50 pages.

```
<some_tosca_entity>:
  properties:
    huge_pages_resources:
      hugepage_size: 2 MiB
      requested_size: 100 MiB
      requested_size_valid_values: [ 50 MiB, 150 MiB ]
```

6.2.71.5 Additional Requirements

None.

6.2.72 `tosca.datatypes.nfv.MaxNumberOfImpactedInstances`

6.2.72.1 Description

The `MaxNumberOfImpactedInstances` data type specifies the maximum number of instances of a given `Vdu.Compute` node or `VnfVirtualLink` node that may be impacted simultaneously without impacting the functionality of the group of a given size. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.72.1-1: Type name, shorthand, and URI

Shorthand Name	<code>MaxNumberOfImpactedInstances</code>
Type Qualified Name	<code>toscanfv:MaxNumberOfImpactedInstances</code>
Type URI	<code>tosca.datatypes.nfv.MaxNumberOfImpactedInstances</code>

6.2.72.2 Properties

The properties of the `MaxNumberOfImpactedInstances` data type shall comply with the provisions set out in table 6.2.72.2-1.

Table 6.2.72.2-1: Properties

Name	Required	Type	Constraints	Description
group_size	no	integer	greater_than: 0	Determines the size of the group for which the max_number_of_impacted_instances is specified. If not present the size is not limited. See notes 1 and 2.
max_number_of_impacted_instances	yes	integer	greater_than: 0	The maximum number of instances that can be impacted simultaneously within the group of the specified size. See notes 1 and 2.
NOTE 1: Each group_size value specified for a group of virtual resources shall be unique, and it shall be possible to form an ascending ordered list of group sizes.				
NOTE 2: The number of instances in the group for which the max_number_of_impacted_instances is specified may be equal to group_size or less. When the number of instances is less than group_size, it shall be at least 1 if this is the first group size in the ordered list of group sizes, or it shall be greater by at least 1 than the previous group size in the ordered list of group sizes.				

6.2.72.3 Definition

The syntax of the MaxNumberOfImpactedInstances data type shall comply with the following definition:

```

tosca.datatypes.nfv.MaxNumberOfImpactedInstances:
  derived_from: tosca.datatypes.Root
  description: Specifies the maximum number of instances of a given Vdu.Compute node
or VnfVirtualLink node that may be impacted simultaneously without impacting the
functionality of the group of a given size.
  properties:
    group_size:
      type: integer
      description: Determines the size of the group for which the
max_number_of_impacted_instances is specified. If not present the size is not limited.
      required: false
      constraints:
        - greater_than: 0
    max_number_of_impacted_instances:
      type: integer
      description: The maximum number of instances that can be impacted
simultaneously within the group of the specified size.
      required: true
      constraints:
        - greater_than: 0

```

6.2.72.4 Examples

None.

6.2.72.5 Additional Requirements

None.

6.2.73 tosca.datatypes.nfv.MinNumberOfPreservedInstances

6.2.73.1 Description

The MinNumberOfPreservedInstances data type specifies the minimum number of instances of a given Vdu.Compute node or VnfVirtualLink node which need to be preserved simultaneously. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.73.1-1: Type name, shorthand, and URI

Shorthand Name	MinNumberOfPreservedInstances
Type Qualified Name	toscanfv:MinNumberOfPreservedInstances
Type URI	tosca.datatypes.nfv.MinNumberOfPreservedInstances

6.2.73.2 Properties

The properties of the MinNumberOPreservedInstances data type shall comply with the provisions set out in table 6.2.73.2-1.

Table 6.2.73.2-1: Properties

Name	Required	Type	Constraints	Description
group_size	no	integer	greater_or_equal: 0	Determines the size of the group for which the min_number_of_preserved_instances is specified. If not present the size is not limited. See notes 1 and 2.
min_number_of_preserved_instances	yes	integer	greater_or_equal: 0	The minimum number of instances which need to be preserved simultaneously within the group of the specified size. See notes 1 and 2.
NOTE 1: Each group_size value specified for a group of virtual resources shall be unique, and it shall be possible to form an ascending ordered list of group sizes.				
NOTE 2: The number of instances in the group for which the min_number_of_preserved_instances is specified may be equal to group_size or less. When the number of instances is less than group_size.				

6.2.73.3 Definition

The syntax of the MinNumberOfPreservedInstances data type shall comply with the following definition:

```

tosca.datatypes.nfv.MinNumberOfPreservedInstances:
  derived_from: toscan.datatypes.Root
  description: Specifies the minimum number of instances of a given Vdu.Compute node
or VnfVirtualLink node which need to be preserved simultaneously.
  properties:
    group_size:
      type: integer
      description: Determines the size of the group for which the
min_number_of_preserved_instances is specified. If not present the size is not
limited.
      required: false
      constraints:
        - greater_than: 0
    min_number_of_preserved_instances:
      type: integer
      description: The minimum number of instances which need to be preserved
simultaneously within the group of the specified size.
      required: true
      constraints:
        - greater_than: 0

```

6.2.73.4 Examples

None.

6.2.73.5 Additional Requirements

None.

6.2.74 `tosca.datatypes.nfv.NfviMaintenanceInfo`

6.2.74.1 Description

The `NfviMaintenanceInfo` data type provides information related to the constraints and rules applicable to virtualised resources and their groups impacted due to NFVI maintenance operations, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.74.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.74.1-1: Type name, shorthand, and URI

Shorthand Name	<code>NfviMaintenanceInfo</code>
Type Qualified Name	<code>toscanfv:NfviMaintenanceInfo</code>
Type URI	<code>tosca.datatypes.nfv.NfviMaintenanceInfo</code>

6.2.74.2 Properties

The properties of the `NfviMaintenanceInfo` data type shall comply with the provisions set out in table 6.2.74.2-1.

Table 6.2.74.2-1: Properties

Name	Required	Type	Constraints	Description
<code>impact_notification_lead_time</code>	yes	scalar-unit.time		Specifies the minimum notification lead time requested for upcoming impact of the virtualised resource or their group (i.e. between the notification and the action causing the impact).
<code>is_impact_mitigation_requested</code>	yes	boolean	default: false	Indicates whether it is requested that at the time of the notification of an upcoming change that is expected to have an impact on the VNF, virtualised resource(s) of the same characteristics as the impacted ones is/are provided to compensate for the impact (TRUE) or not (FALSE).
<code>supported_migration_type</code>	no	list of string	valid values: <code>no_migration</code> , <code>offline_migration</code> , <code>live_migration</code>	Specifies the allowed migration types in the order of preference in case of an impact starting with the most preferred type. It is applicable to the <code>Vdu.Compute</code> node and to the <code>VirtualBlockStorage</code> , <code>VirtualObjectStorage</code> and <code>VirtualFileStorage</code> nodes. See note 1.
<code>max_undetectable_interruption_time</code>	no	scalar-unit.time		Specifies the maximum interruption time that can go undetected at the VNF level and therefore which will not trigger VNF-internal recovery during live migration. It is applicable to the <code>Vdu.Compute</code> node and to the <code>VirtualBlockStorage</code> , <code>VirtualObjectStorage</code> and <code>VirtualFileStorage</code> nodes. See note 1.
<code>min_recovery_time_between_impacts</code>	no	scalar-unit.time		Specifies the time required by the group to recover from an impact, thus, the minimum time requested between consecutive impacts of the group. See note 2.
<code>max_number_of_impacted_instances</code>	no	list of <code>tosca.datatypes.nfv.MaxNumberOfImpactedInstances</code>		Specifies for different group sizes the maximum number of instances that can be impacted simultaneously within the group of virtualised resources without losing functionality. See notes 2 and 3.

Name	Required	Type	Constraints	Description
min_number_of_preserved_instances	no	list of tosca.datatype.nfv.MinNumberOfPreservedInstances		Specifies for different group sizes the minimum number of instances which need to be preserved simultaneously within the group of virtualised resources. See notes 2 and 3.
NOTE 1: When the max_undetectable_interruption_time is specified it constrains the live migration. If it cannot be guaranteed on an NFVI that the interruption caused by the live migration will be less than the indicated maximum undetectable interruption time, then life migration should be downgraded according to the order of preference. NOTE 2: Impacts to instances of the group happening within the min_recovery_time_between_impacts are considered simultaneous impacts. NOTE 3: Either "max_number_of_impacted_instances" or "min_number_of_preserved_instances" may be provided, but not both.				

6.2.74.3 Definition

The syntax of the NfviMaintenanceInfo data type shall comply with the following definition:

```

tosca.datatypes.nfv.NfviMaintenanceInfo:
  derived_from: toska.datatypes.Root
  description: Provides information related to the constraints and rules applicable
to virtualised resources and their groups impacted due to NFVI maintenance operations
  properties:
    impact_notification_lead_time:
      type: scalar-unit.time
      description: Specifies the minimum notification lead time requested for
upcoming impact of the virtualised resource or their group (i.e. between the
notification and the action causing the impact).
      required: true
    is_impact_mitigation_requested:
      type: boolean
      description: Indicates whether it is requested that at the time of the
notification of an upcoming change that is expected to have an impact on the VNF,
virtualised resource(s) of the same characteristics as the impacted ones is/are
provided to compensate for the impact (TRUE) or not (FALSE).
      required: true
      default: false
    supported_migration_type:
      type: list
      description: Specifies the allowed migration types in the order of preference
in case of an impact starting with the most preferred type. It is applicable to the
Vdu.Compute node and to the VirtualBlockStorage, VirtualObjectStorage and
VirtualFileStorage nodes.
      required: false
      entry_schema:
        type: string
        constraints:
          - valid_values: [ no_migration, offline_migration, live_migration ]
    max_undetectable_interruption_time:
      type: scalar-unit.time
      description: Specifies the maximum interruption time that can go undetected at
the VNF level and therefore which will not trigger VNF-internal recovery during live
migration. It is applicable to the Vdu.Compute node and to the VirtualBlockStorage,
VirtualObjectStorage and VirtualFileStorage nodes.
      required: false
    min_recovery_time_between_impacts:
      type: scalar-unit.time
      description: Specifies the time required by the group to recover from an
impact, thus, the minimum time requested between consecutive impacts of the group..
      required: false

```

```

    max_number_of_impacted_instances:
      type: list
      description: Specifies for different group sizes the maximum number of
instances that can be impacted simultaneously within the group of virtualised
resources without losing functionality.
      required: false
      entry_schema:
        type: toska.datatypes.nfv.MaxNumberOfImpactedInstances
    min_number_of_preserved_instances:
      type: list
      description: Specifies for different group sizes the minimum number of
instances which need to be preserved simultaneously within the group of virtualised
resources.
      required: false
      entry_schema:
        type: toska.datatypes.nfv.MinNumberOfPreservedInstances

```

6.2.74.4 Examples

None.

6.2.74.5 Additional Requirements

None.

6.2.75 toska.datatypes.nfv.MciIdentificationData

6.2.75.1 Description

The MciIdentificationData data type contains data needed to identify an MCIO when interworking with the CISM, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.75.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.75.1-1: Type name, shorthand and URI

Shorthand Name	MciIdentificationData
Type Qualified Name	toscanfv: MciIdentificationData
Type URI	tosca.datatypes.nfv.MciIdentificationData

6.2.75.2 Properties

The properties of the MciIdentificationData data type shall comply with the provisions set out in table 6.2.75.2-1.

Table 6.2.75.2-1: Properties

Name	Required	Type	Constraints	Description
name	yes	String		The name of the mcio. See note 1.
type	yes	String	Valid values: See YAML definition constraints	The type of the mcio. See note 2.
NOTE 1: When the container infrastructure service is a Kubernetes® instance it is the value of the 'metadata.name' field in Kubernetes® manifest.				
NOTE 2: When the container infrastructure service is a Kubernetes® instance it is the value of the 'kind' field in Kubernetes® manifest.				

6.2.75.3 Definition

The syntax of the McioIdentificationData data type shall comply with the following definition:

```

tosca.datatypes.nfv.McioIdentificationData:
  derived_from: toska.datatypes.Root
  description: contains data needed to identify an MCIO when interworking with the
CISM.
  properties:
    name:
      type: string
      description: The name of the mcio.
      required: true
    type:
      type: string
      description: The type of the mcio.
      required: true
      constraints:
        - valid_values: [ Deployment, StatefulSet, DaemonSet ] #Name capitalized as
in k8s

```

6.2.75.4 Examples

None.

6.2.75.5 Additional Requirements

None.

6.2.76 toska.datatypes.nfv.VipCpLevel

6.2.76.1 Description

The VipCpLevel data type indicates for a given VipCp in a given level the number of instances to deploy. Table 6.2.76.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.76.1-1: Type name, shorthand and URI

Shorthand Name	VipCpLevel
Type Qualified Name	toscanfv:VipCpLevel
Type URI	tosca.datatypes.nfv.VipCpLevel

6.2.76.2 Properties

The properties of the VipCpLevel data type shall comply with the provisions set out in table 6.2.76.2-1.

Table 6.2.76.2-1: Properties

Name	Required	Type	Constraints	Description
number_of_instances	yes	integer	greater_or_equal: 0	Number of instances of VipCp based on the referenced VipCp node template to deploy for an instantiation level or for a scaling delta.

6.2.76.3 Definition

The syntax of the VipCpLevel data type shall comply with the following definition:

```
tosca.datatypes.nfv.VipCpLevel:
  derived_from: toska.datatypes.Root
  description: Indicates for a given VipCp in a given level the number of instances
to deploy
  properties:
    number_of_instances:
      type: integer
      description: Number of instances of VipCp based on the referenced VipCp node
template to deploy for an instantiation level or for a scaling delta.
      required: true
      constraints:
        - greater_or_equal: 0
```

6.2.76.4 Examples

None.

6.2.76.5 Additional Requirements

None.

6.2.77 toska.datatypes.nfv.ServiceData

6.2.77.1 Description

The ServiceData data type indicates the service matching information exposed by the VirtualCp. Table 6.2.77.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.77.1-1: Type name, shorthand and URI

Shorthand Name	ServiceData
Type Qualified Name	toscanfv:ServiceData
Type URI	tosca.datatypes.nfv.ServiceData

6.2.77.2 Properties

The properties of the ServiceData data type shall comply with the provisions set out in table 6.2.77.2-1.

Table 6.2.77.2-1: Properties

Name	Required	Type	Constraints	Description
host	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the fully qualified domain name of a network host as defined by IETF RFC 3986 [8]. See note.
path	no	List of string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the path component of a URI, as per IETF RFC 3986 [8]. See note.
NOTE: Either "host" or "path" shall be present.				

6.2.77.3 Definition

The syntax of the ServiceData data type shall comply with the following definition:

```
tosca.datatypes.nfv.ServiceData:
  derived_from: toska.datatypes.Root
```

```

description: Indicates the service matching information exposed by the VirtualCp
properties:
  host:
    type: string # shall comply with IETF RFC 3986
    description: Corresponds to the fully qualified domain name of a network host
    required: false
  path:
    type: list
    description: path component of a URI.
    required: false
  entry_schema:
    type: string # shall comply with IETF RFC 3986

```

6.2.77.4 Examples

None.

6.2.77.5 Additional Requirements

None.

6.2.78 **tosca.datatypes.nfv.** **SelectVnfDeployableModulesOperationConfiguration**

6.2.78.1 Description

The SelectVnfDeployableModulesOperationConfiguration data type represents information that affect the invocation of the SelectVnfDeployableModules operation, as specified in ETSI GS NFV-IFA 011 [1]. This data type definition is reserved for future use in the present document. Table 6.2.78.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.78.1-1: Type name, shorthand, and URI

Shorthand Name	SelectVnfDeployableModulesOperationConfiguration
Type Qualified Name	toscanfv:SelectVnfDeployableModulesOperationConfiguration
Type URI	tosca.datatypes.nfv.SelectVnfDeployableModulesOperationConfiguration

6.2.78.2 Properties

None.

6.2.78.3 Definition

The syntax of the SelectVnfDeployableModulesOperationConfiguration data type shall comply with the following definition:

```

tosca.datatypes.nfv.SelectVnfDeployableModulesOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the
SelectVnfDeployableModules operation
  # This data type definition is reserved for future use in the present document.
  # properties:

```

6.2.78.4 Examples

None.

6.2.78.5 Additional Requirements

None.

6.2.79 `tosca.datatypes.nfv.CertificateBaseProfile`

6.2.79.1 Description

The `CertificateBaseProfile` data type describes base profile for certificate used in a specific deployment flavour. Table 6.2.79.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.79.1-1: Type name, shorthand and URI

Shorthand Name	<code>CertificateBaseProfile</code>
Type Qualified Name	<code>tosca.nfv.CertificateBaseProfile</code>
Type URI	<code>tosca.datatypes.nfv.CertificateBaseProfile</code>

6.2.79.2 Properties

The properties of the `CertificateBaseProfile` data type shall comply with the provisions set out in table 6.2.79.2-1.

Table 6.2.79.2-1: Properties

Name	Required	Type	Constraints	Description
<code>id</code>	yes	string		The identifier of this certificate profile.
<code>issuer</code>	yes	string		Issuer of certificates. See note.
<code>issuer_unique_identifier</code>	yes	string		Identifier of this issuer of certificates. See note.
<code>subject</code>	yes	<code>tosca.datatypes.nfv.CertSubjectData</code>		Subject of certificates. See note.
<code>subject_unique_identifier</code>	yes	string		Identifier of this subject of certificates.
<code>basic_constraints</code>	yes	string		Basic constraints of certificates. See note.
<code>issuer_alt_name</code>	no	string		Alternative name of the issuer of certificates. See note.
<code>subject_alt_name</code>	no	string		Alternative name of the subject of certificates.
<code>name_constraints</code>	no	string		Name constraints of certificates.
NOTE : This value can be overridden at run-time.				

6.2.79.3 Definition

The syntax of the `CertificateBaseProfile` data type shall comply with the following definition:

```
tosca.datatypes.nfv.CertificateBaseProfile:
  derived_from: toska.datatypes.Root
  description: describes base profile for certificate used in a specific deployment
  flavour.
  properties:
    id:
      type: string
      description: The identifier of this certificate profile.
      required: true
    issuer:
      type: string
      description: Issuer of certificates.
      required: true
    issuer_unique_identifier:
```



```

    type: string
    description: Identifier of this issuer of certificates.
    required: true
  subject:
    type: toscadatatypes.nfv.CertSubjectData
    description: Subject of certificates.
    required: true
  subject_unique_identifier:
    type: string
    description: Identifier of this subject of certificates.
    required: true
  basic_constraints:
    type: string
    description: Basic constraints of certificates.
    required: true
  issuer_alt_name:
    type: string
    description: Alternative name of the issuer of certificates.
    required: false
  subject_alt_name:
    type: string
    description: Alternative name of the subject of certificates.
    required: false
  name_constraints:
    type: string
    description: Name constraints of certificates.
    required: false

```

6.2.79.4 Examples

None.

6.2.79.5 Additional requirements

None.

6.2.80 toscadatatypes.nfv.CSRRequirements

6.2.80.1 Description

The CSRRequirements data type describes requirements for certificate. Table 6.2.80.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.80.1-1: Type name, shorthand, and URI

Shorthand Name	CSRRequirements
Type Qualified Name	toscanfv.CSRRequirements
Type URI	tosca.datatypes.nfv.CSRRequirements

6.2.80.2 Properties

The properties of the CSRRequirements data type shall comply with the provisions set out in table 6.2.80.2-1.

Table 6.2.80.2-1: Properties

Name	Required	Type	Constraints	Description
id	yes	string		The identifier of this CSR requirements.

Name	Required	Type	Constraints	Description
supported_certificate_version	yes	string	Valid values: See YAML definition constraints	The certificate version can be supported by this VNF or VDU.
supported_signature	yes	string	Valid values: See YAML definition constraints	The signature algorithm can be supported by this VNF or VDU.
supported_max_keylength	no	integer	greater_or_equal: 0	Max key length can be supported this VNF or VDU. Default values is 4096 bits for RSA based algorithm. Otherwise default value is 512 bits.
supported_min_keylength	no	integer	greater_or_equal: 0	Min key length can be supported this VNF or VDU. Default value is 2048 bits for RSA based algorithm. Otherwise default value is 256 bits.
maximum_validity_period	no	integer	greater_or_equal: 0	Maximum validity period in days for this certificate.
minimum_validity_period	no	integer	greater_or_equal: 0	Minimum validity period in days for this certificate.
key_usages	no	list of string		Requirements for key usage of this certificate.

6.2.80.3 Definition

The syntax of the CSRRequirements data type shall comply with the following definition:

```

tosca.datatypes.nfv.CSRRequirements:
  derived_from: toska.datatypes.Root
  description: CSRRequirements describes requirements for certificate.
  properties:
    id:
      type: string
      description: The identifier of this CSR requirements
      required: true
    supported_certificate_version:
      type: string
      description: The certificate version can be supported by this VNF or VDU.
      required: true
      constraints:
        - valid_values: [ X.509_ver3 ]
    supported_signature:
      type: string
      description: The signature algorithm can be supported by this VNF or VDU.
      required: true
      constraints:
        - valid_values: [ sha256withRSAEncryption, sha384WithRSAEncryption,
sha512WithRSAEncryption ]
    supported_max_keylength:
      type: integer
      description: Max key length can be supported this VNF or VDU. Default
values is 4096 bits for RSA based algorithm. Otherwise default value is 512 bits.
      required: false
      constraints:
        - greater_or_equal: 0
    supported_min_keylength:
      type: integer
      description: Min key length can be supported this VNF or VDU. Default
value is 2048 bits for RSA based algorithm. Otherwise default value is 256 bits.
      required: false
      constraints:
        - greater_or_equal: 0
    maximum_validity_period:
      type: integer
      description: Maximum validity period in days for this certificate.
      required: false
      constraints:
        - greater_or_equal: 0
    minimum_validity_period:
      type: integer
      description: Minimum validity period in days for this certificate.
      required: false
      constraints:
        - greater_or_equal: 0
    key_usages:
      type: list
      description: Requirements for key usage of this certificate.
      required: false
      entry_schema:
        type: string

```

6.2.80.4 Examples

None.

6.2.80.5 Additional Requirements

None.

6.2.81 `tosca.datatypes.nfv.CertSubjectData`

6.2.81.1 Description

The `CertSubjectData` data type describes subject data of the certificate. Table 6.2.81.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.81.1-1: Type name, shorthand, and URI

Shorthand Name	<code>CertSubjectData</code>
Type Qualified Name	<code>tosca.nfv.CertSubjectData</code>
Type URI	<code>tosca.datatypes.nfv.CertSubjectData</code>

6.2.81.2 Properties

The properties of the `CertSubjectData` data type shall comply with the provisions set out in table 6.2.81.2-1.

Table 6.2.81.2-1: Properties

Name	Required	Type	Constraints	Description
<code>common_name</code>	no	string		Information of certification target subject FQDN.
<code>organization</code>	no	string		Information of certification target subject Organization.
<code>country</code>	no	string		Information of certification target subject Country.
<code>state</code>	no	string		Information of certification target subject State.
<code>locality</code>	no	string		Information of certification target subject Locality.
<code>email_address</code>	no	string		Information of certification contact email address.

6.2.81.3 Definition

The syntax of the `CertSubjectData` data type shall comply with the following definition:

```

tosca.datatypes.nfv.CertSubjectData:
  derived_from: toska.datatypes.Root
  description: CertSubjectData describes subject data of the certificate.
  properties:
    common_name:
      type: string
      description: Information of certification target subject FQDN.
      required: false
    organization:
      type: string
      description: Information of certification target subject Organization.
      required: false
    country:
      type: string
      description: Information of certification target subject Country.
      required: false
    state:
      type: string
      description: Information of certification target subject State.
      required: false
    locality:
      type: string
      description: Information of certification target subject Locality.
      required: false
    email_address:
      type: string
      description: Information of certification contact email address.
      required: false

```

6.2.81.4 Examples

None.

6.2.81.5 Additional Requirements

None.

6.2.82 toska.datatypes.nfv.SizeRange

6.2.82.1 Description

The SizeRange data type is a refined list type that supports the specification of a range of scalar-unit.size values by indicating the lower boundary in the first element of the list and the upper boundary in the second element of the list. Table 6.2.82.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.82.1-1: Type name, shorthand, and URI

Shorthand Name	SizeRange
Type Qualified Name	toscanfv:SizeRange
Type URI	tosca.datatypes.nfv.SizeRange

6.2.82.2 Properties

None.

6.2.82.3 Definition

The syntax of the SizeRange data type shall comply with the following definition:

```
tosca.datatypes.nfv.SizeRange:
  derived_from: toska.datatypes.List
  description: supports the specification of a range of scalar-unit.size values by
indicating the lower boundary in the first element of the list and the upper boundary
in the second element of the list.
  constraints:
    - length: 2
  entry_schema:
    type: scalar-unit.size
```

6.2.82.4 Examples

The following example shows an entity using a 'size_of_storage_valid_values' property of toska.datatypes.nfv.SizeRange data type to indicate a range of values between 2 GB and 4 GB.

```
<some_tosca_entity>:
  properties:
    size_of_storage_valid_values: [ 2 GB, 4 GB ]
```

6.2.82.5 Additional Requirements

None.

6.2.83 toska.datatypes.nfv.IntegerRange

6.2.83.1 Description

The IntegerRange data type is a refined list type that supports the specification of a range of integer values by indicating the lower boundary in the first element of the list and the upper boundary in the second element of the list.

Table 6.2.83.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.83.1-1: Type name, shorthand, and URI

Shorthand Name	IntegerRange
Type Qualified Name	toscanfv:IntegerRange
Type URI	tosca.datatypes.nfv.IntegerRange

6.2.83.2 Properties

None.

6.2.83.3 Definition

The syntax of the IntegerRange data type shall comply with the following definition:

```
tosca.datatypes.nfv.IntegerRange:
  derived_from: toska.datatypes.List
  description: supports the specification of a range of integer values by indicating
the lower boundary in the first element of the list and the upper boundary in the
second element of the list.
  constraints:
    - length: 2
```

```
entry_schema:
  type: integer
```

6.2.83.4 Examples

The following example shows an entity using a 'requested_cpu_resources_valid_values' property of `tosca.datatypes.nfv.IntegerRange` data type to indicate a range of values between 100 and 200 milli-CPU.

```
<some_tosca_entity>:
  properties:
    size_of_storage_valid_values: [ 100, 200 ]
```

6.2.83.5 Additional Requirements

None.

6.2.84 `tosca.datatypes.nfv.CpuPowerState`

6.2.84.1 Description

The `CpuPowerState` data type supports the specification of requirements related to the CPU power state configuration of a virtual compute resource or OS container, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.84.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.84.1-1: Type name, shorthand, and URI

Shorthand Name	<code>CpuPowerState</code>
Type Qualified Name	<code>toscanfv:CpuPowerState</code>
Type URI	<code>tosca.datatypes.nfv.CpuPowerState</code>

6.2.84.2 Properties

The properties of the `CpuPowerState` data type shall comply with the provisions set out in table 6.2.84.2-1.

Table 6.2.84.2-1: Properties

Name	Required	Type	Constraints	Description
<code>cpu_power_state_management_policy</code>	yes	string	Valid values: See YAML definition constraints	Indicates the policy for CPU power state management. The policy can take values of "static" or "dynamic". In case of "static", the virtual CPU cores are requested to be allocated to logical CPU cores according to the <code>cpu_operational_power_state</code> property. In case of "dynamic" the CPU power states of virtual CPU cores can be allocated to logical CPU cores whose power states can be adjusted dynamically depending on core utilization (see note 1).
<code>cpu_operational_power_state</code>	yes	list of string		Provides the list of operational power states (i.e. P and C states) defined for the virtual CPU (see note 2).

Name	Required	Type	Constraints	Description
NOTE 1: In case the value of the <code>cpu_power_state_management_policy</code> property is "dynamic", dynamic adjustment of CPU power states of logical CPU cores should adhere to the range of CPU P and C states defined in the <code>cpu_operational_power_state</code> property.				
NOTE 2: In case the value of the <code>cpu_power_state_management_policy</code> property is "static", the property <code>cpu_operational_power_state</code> shall contain only one set of CPU P and C states requested for the virtual CPU. If the value of the <code>cpu_power_state_management_policy</code> property is "dynamic", the list of CPU P and C states can be defined to indicate the range of allowed P/C state values during dynamic power state management.				

6.2.84.3 Definition

The syntax of the `CpuPowerState` data type shall comply with the following definition:

```

tosca.datatypes.nfv.CpuPowerState:
  derived_from: tosca.datatypes.Root
  description: Supports the specification of requirements related to the CPU power
state configuration of a virtual compute resource or OS container.
  properties:
    cpu_power_state_management_policy:
      type: string
      description: Indicates the policy for CPU power state management. The policy
can take values of "static" or "dynamic". In case of "static", the virtual CPU cores
are requested to be allocated to logical CPU cores according to the
cpu_operational_power_state property. In case of "dynamic" the CPU power states of
virtual CPU cores can be allocated to logical CPU cores whose power states can be
adjusted dynamically depending on core utilization.
      required: true
      constraints:
        - valid_values: [ static, dynamic ]
    cpu_operational_power_state:
      type: list
      description: Provides the list of operational power states (i.e. P and C
states) defined for the virtual CPU.
      required: true
      entry_schema:
        type: string

```

6.2.84.4 Examples

None.

6.2.84.5 Additional Requirements

None.

6.2.85 `tosca.datatypes.nfv.PowerProfile`

6.2.85.1 Description

The `PowerProfile` data type describes the additional information about power profiles supported by the VNF in a specific VNF deployment flavour.

Among others, the `PowerProfile` information element includes information and references to instantiation and scaling levels that further provide information about the VNF's resource structure (e.g. of VDUs, internal VLs, etc.) that are applicable for setting the VNF instance into a defined VNF power state.

Table 6.8.85.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.85.1-1: Type name, shorthand, and URI

Shorthand Name	PowerProfile
Type Qualified Name	toscanfv:PowerProfile
Type URI	tosca.datatypes.nfv.PowerProfile

6.2.85.2 Properties

The properties of the PowerProfile data type shall comply with the provisions set out in table 6.2.85.2-1.

Table 6.8.85.2-1: Properties

Name	Required	Type	Constraints	Description
name	yes	string		Name of the power profile.
associated_instantiation_level	no	string		Reference to the instantiation level in the deployment flavour, which is associated with the power profile. See notes 1 and 3.
associated_scale_level	no	map of tosca.datatypes.nfv.ScaleInfo		Provides information about an arbitrary scale level independent of a specific instantiation level, which is associated with this power profile. All scaling aspects needed for this scale level shall be included in this attribute. This attribute shall be present if the VNF supports scaling. See notes 1 and 3.
power_consumption_info	no	tosca.datatypes.nfv.PowerConsumptionInfo		Estimated power consumption information of the resources associated with this power profile.
NOTE 1: A power profile can be associated with an instantiation level in the deployment flavour or an arbitrary scale level that a VNF can achieve via appropriate VNF LCM operation using ScaleInfo, e.g. instantiate VNF, scale VNF to level, etc.				
NOTE 2: A power profile can also be associated with a specific set of VDUs and their resource configurations. Such kind of association can be helpful in linking VNF power profiles with vertical scaling use cases that increase or release a VNF capacity by achieving desired VDU resource configurations.				
NOTE 3: To avoid conflicting associations, only one of the associated_instantiation_level, associated_scale_level, or associated_vdu_resource_composition properties shall be present.				

6.2.85.3 Definition

The syntax of the PowerProfile data type shall comply with the following definition:

```

tosca.datatypes.nfv.PowerProfile:
  derived_from: toscanfv.Root
  description: supports the specification of requirements related to the additional
information about power profiles supported by the VNF in a specific VNF deployment
flavour.
  properties:
    name:
      type: string
      description: Name of the power profile.
      required: true
    associated_vdu_resourceComposition:
      type: list
      description: List of VDUs (in VNFD) and their resource composition associated
with this power profile.
      required: false
    entry_schema:
      type: string

```

```

    powerConsumptionInfo:
      type: string
      description: Estimated power consumption information of the resources
associated with this power profile..
      required: false
    capabilities:
      applicable_power_profile:
        type: toska.capabilities.nfv.ApplicablePowerProfile
        occurrences: [ 1, 1 ]

```

6.2.85.4 Examples

None.

6.2.85.5 Additional Requirements

None.

6.2.86 toska.datatypes.nfv.PowerConsumptionInfo

6.2.86.1 Description

The PowerConsumptionInfo data type describes power consumption specific deployment flavour. Table 6.2.86.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.2.86.1-1: Type name, shorthand and URI

Shorthand Name	PowerConsumptionInfo
Type Qualified Name	toscanfv: PowerConsumptionInfo
Type URI	tosca.datatypes.nfv.PowerConsumptionInfo

6.2.86.2 Properties

The properties of the PowerConsumptionInfo data type shall comply with the provisions set out in table 6.2.86.2-1.

Table 6.2.86.2-1: Properties

Name	Required	Type	Constraints	Description
min_power_consumption	yes	integer		Minimum power consumption of this power profile. Unit is KW.
max_power_consumption	yes	Integer		Maximum power consumption of this power profile. Unit is KW.
average_power_consumption	yes	integer		Average of power consumption of this power profile. Unit is KW

6.2.86.3 Definition

The syntax of the PowerConsumptionInfo data type shall comply with the following definition:

```

tosca.datatypes.nfv.PowerConsumptionInfo:
  derived_from: toska.datatypes.Root
  description: describes power consumption specific deployment flavour.
  properties:
    min_consumption:
      type: integer

```

```

    description: Minimum power consumption of this power profile. Unit is KW.
    required: true
  max_consumption:
    type: integer
    description: Maximum power consumption of this power profile. Unit is KW.
    required: true
  average_consumption:
    type: integer
    description: Average power consumption of this power profile. Unit is KW.
    required: true

```

6.2.86.4 Examples

None.

6.2.86.5 Additional requirements

None.

6.3 Artifact Types

6.3.1 `tosca.artifacts.nfv.SwImage`

6.3.1.1 Description

The SwImage artifact describes the software image which is directly loaded on the virtualisation container realizing of the VDU or is to be loaded on a virtual storage resource, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.3.1.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.3.1.1-1: Type name, shorthand, and URI

Shorthand Name	SwImage
Type Qualified Name	toscanfv:SwImage
Type URI	tosca.artifacts.nfv.SwImage

6.3.1.2 Properties

The properties of the SwImage artifacts type shall comply with the provisions set out in table 6.3.1.2-1.

Table 6.3.1.2-1: Properties

Name	Required	Type	Constraints	Description
name	yes	string		Name of this software image.
version	yes	string		Version of this software image.
provider	no	string		Provider of this software image.
checksum	no	tosca.data types.nfv. Checksum Data		Checksum of the software image file. See note 5.

Name	Required	Type	Constraints	Description
container_format	yes	string	Valid values: See YAML definition constraints	The container format describes the container file format in which software image is provided. Description of valid values: aki: a kernel image ami: a machine image ari: a ramdisk image bare: the image does not have a container or metadata envelope docker: docker container format ova: OVF package in a tarfile ovf: OVF container format Future versions of the present document may extend the list of possible values. See note 1.
disk_format	no	string	Valid values: See YAML definition constraints	The disk format of a software image is the format of the underlying disk image. Description of valid values: aki: a kernel image ami: a machine image ari: a ramdisk image iso: an archive format for the data contents of an optical disc, such as CD-ROM qcow2: a common disk image format, which can expand dynamically and supports copy on write raw: an unstructured disk image format vdi: a common disk image format vhd: a common disk image format vhdx: enhanced version of VHD format vmdk: a common disk image format Future versions of the present document may extend the list of possible values. See note 2 and note 3.
min_disk	no	scalar-unit.size	greater_or_equal: 0 B	The minimal disk size requirement for this software image. See note 3.
min_ram	no	scalar-unit.size	greater_or_equal: 0 B	The minimal RAM requirement for this software image. See note 4.
size	no	scalar-unit.size		The size of this software image. See note 5.
operating_system	no	string		Identifies the operating system used in the software image.
supported_virtualisation_environments	no	list of string		Identifies the virtualisation environments (e.g. hypervisor) compatible with this software image.
NOTE 1: The list of permitted values was taken from "Container formats" in [i.12]. NOTE 2: The list of permitted values was adapted from "Disk formats" in [i.12]. NOTE 3: This property shall be present when the SwImage artifact is contained in a Node template of type <code>tosca.nodes.nfv.Vdu.Compute</code> or <code>tosca.nodes.nfv.Vdu.VirtualBlockStorage</code>, and shall be absent otherwise. NOTE 4: This property may be present when the SwImage artifact is contained in a Node template of type <code>tosca.nodes.nfv.Vdu.Compute</code> or <code>tosca.nodes.nfv.Vdu.VirtualBlockStorage</code>, and shall be absent otherwise. NOTE 5: This property shall be present when the SwImage artifact is contained in a Node template of type <code>tosca.nodes.nfv.Vdu.Compute</code> or <code>tosca.nodes.nfv.Vdu.VirtualBlockStorage</code> and may be absent otherwise.				

6.3.1.3 Definition

```

tosca.artifacts.nfv.SwImage:
  derived_from: toska.artifacts.Deployment.Image
  description: describes the software image which is directly loaded on the
virtualisation container realizing of the VDU or is to be loaded on a virtual storage
resource
  properties:

```

```

name:
  type: string
  description: Name of this software image
  required: true
version:
  type: string
  description: Version of this software image
  required: true
provider:
  type: string
  description: Provider of this software image
  required: false
checksum:
  type: tosc.datatypes.nfv.ChecksumData
  description: Checksum of the software image file
  required: false
container_format:
  type: string
  description: The container format describes the container file format in which
software image is provided
  required: true
  constraints:
    - valid_values: [ aki, ami, ari, bare, docker, ova, ovf ]
disk_format:
  type: string
  description: The disk format of a software image is the format of the
underlying disk image
  required: false
  constraints:
    - valid_values: [ aki, ami, ari, iso, qcow2, raw, vdi, vhd, vhdx, vmdk ]
min_disk:
  type: scalar-unit.size # Number
  description: The minimal disk size requirement for this software image
  required: false
  constraints:
    - greater_or_equal: 0 B
min_ram:
  type: scalar-unit.size # Number
  description: The minimal RAM requirement for this software image
  required: false
  constraints:
    - greater_or_equal: 0 B
size:
  type: scalar-unit.size # Number
  description: The size of this software image
  required: false
operating_system:
  type: string
  description: Identifies the operating system used in the software image
  required: false
supported_virtualisation_environments:
  type: list
  description: Identifies the virtualisation environments (e.g. hypervisor)
compatible with this software image
  required: false
entry_schema:
  type: string

```

6.3.2 `tosca.artifacts.Implementation.nfv.Mistral`

6.3.2.1 Description

This artifact type represents a Mistral file that contains Mistral language [i.7] constructs that can be executed within a Mistral workbook. Support of this type is optional.

Table 6.3.2.1-1: Type name, shorthand, and URI

Shorthand Name	Mistral
Type Qualified Name	toscanfv:Mistral
Type URI	tosca.artifacts.Implementation.nfv.Mistral
derived_from	tosca.artifacts.Implementation

6.3.2.2 Definition

The syntax of the Mistral artifact type shall comply with the following definition:

```
tosca.artifacts.Implementation.nfv.Mistral:
  derived_from: toscanfv.Implementation
  description: artifacts for Mistral workflows
  mime_type: application/x-yaml
  file_ext: [ yaml ]
```

6.3.3 `tosca.artifacts.nfv.HelmChart`

6.3.3.1 Description

The HelmChart artifact is a file containing a Helm™ chart [23].

Table 6.3.3.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Whether the Helm chart contains custom resource definitions is out of scope of the present document.

Table 6.3.3.1-1: Type name, shorthand, and URI

Shorthand Name	HelmChart
Type Qualified Name	toscanfv:HelmChart
Type URI	tosca.artifacts.nfv.HelmChart

6.3.3.2 Properties

None.

6.3.3.3 Description

```
tosca.artifacts.nfv.HelmChart:
  derived_from: toscanfv.File
  description: describes the Helm chart artifact.
  file_ext: [ tar, tar.gz, tgz ]
```

6.3.4 tosca.artifacts.nfv.HelmParamMappingScript

6.3.4.1 Description

The HelmParamMappingScript artifact contains an executable file that generates in its standard output the contents of a Helm™ values.yaml file [23] to be passed to a Helm based CISM service interface for containerized workloads based on MCIOPs, when invoking a command triggered by a VNF LCM operation.

Table 6.3.4.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

The executable file runs in the VNFM execution environment. It is invoked, after receiving the VNF LCM operation and prior to sending the request to the Helm based CISM service interface, with the following list of ordered parameters:

- 1) URL of a readable file containing the complete task resource (e.g. InstantiateVnfRequest) as received in the VNF LCM operation, in JSON format.
- 2) URL of a readable zip file with the contents of the VNFD, as defined in clause 10.4.4.3.2 of ETSI GS NFV-SOL 003 [25] using the option without security information.
- 3) URL of the file referenced in the HelmParamMappingRule artifact defined in the same node template as the HelmParamMappingScript artifact, if any.

The executable file is only invoked for VNF LCM operations that trigger a request towards CISM, except the Query VNF operation, and if the artifact of type HelmParamMappingScript is defined for the applicable Mciop node template.

Table 6.3.4.1-1: Type name, shorthand, and URI

Shorthand Name	HelmParamMappingScript
Type Qualified Name	toscanfv:HelmParamMappingScript
Type URI	tosca.artifacts.nfv.HelmParamMappingScript

6.3.4.2 Properties

The properties of the HelmParamMappingScript artifacts type shall comply with the provisions set out in table 6.3.4.2-1.

Table 6.3.4.2-1: Properties

Name	Required	Type	Constraints	Description
language	yes	string	Valid values: See YAML definition constraints	The language in which the script is written

6.3.4.3 Definition

```
tosca.artifacts.nfv.HelmParamMappingScript:
  derived_from: tosca.artifacts.File
  description: contains an executable file that generates in its standard output the
  contents of a Helm values.yaml file to be passed to a Helm based CISM service
  interface for containerized workloads based on MCIOPs, when invoking a command
  triggered by a VNF LCM operation.
  properties:
    language:
      type: string
      constraints:
        - valid_values: [ bash, python]
```

6.3.5 tosa.artifacts.nfv.HelmParamMappingRule

6.3.5.1 Description

The HelmParamMappingRule artifact contains a file with rules used by the HelmParamMappingScript artifact defined in the same node template as this artifact, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.3.5.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

The contents of the file are VNF specific as they only need to be interpreted by the HelmParamMappingScript artifact.

Table 6.3.5.1-1: Type name, shorthand, and URI

Shorthand Name	HelmParamMappingRule
Type Qualified Name	toscanfv:HelmParamMappingRule
Type URI	tosca.artifacts.nfv.HelmParamMappingRule

6.3.5.2 Properties

None.

6.3.5.3 Definition

```
tosca.artifacts.nfv.HelmParamMappingRule:
  derived_from: tosa.artifacts.File
  description: contains a file with rules used by the HelmParamMappingScript
artifact defined in the same node template as this artifact.
```

6.4 Capability Types

6.4.1 tosa.capabilities.nfv.VirtualBindable

6.4.1.1 Description

The VirtualBindable capability indicates that the node that includes it can be pointed by a tosa.relationships.nfv.VirtualBindsTo relationship type which is used to model the VduHasCpd association illustrated in ETSI GS NFV-IFA 011 [1]. Table 6.4.1.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.4.1.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualBindable
Type Qualified Name	toscanfv:VirtualBindable
Type URI	tosca.capabilities.nfv.VirtualBindable

6.4.1.2 Properties

None.

6.4.1.3 Definition

The syntax of the VirtualBindable capability type shall comply with the following definition:

```
tosca.capabilities.nfv.VirtualBindable:
  derived_from: tosa.capabilities.Node
```


description: Indicates that the node that includes it can be pointed by a `tosca.relationships.nfv.VirtualBindsTo` relationship type which is used to model the `VduHasCpd` association

6.4.2 `tosca.capabilities.nfv.VirtualLinkable`

6.4.2.1 Description

The `VirtualLinkable` capability type is defined in clause 9.4.1 of the present document.

6.4.3 `tosca.capabilities.nfv.VirtualCompute`

6.4.3.1 Description

The `VirtualCompute` capability type describes the capabilities related to virtual compute resources, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.4.3.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.4.3.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VirtualCompute</code>
Type Qualified Name	<code>toscanfv:VirtualCompute</code>
Type URI	<code>tosca.capabilities.nfv.VirtualCompute</code>

6.4.3.2 Properties

The properties of the `VirtualCompute` capability type shall comply with the provisions set out in table 6.4.3.2-1.

Table 6.4.3.2-1: Properties

Name	Required	Type	Constraints	Description
<code>logical_node</code>	no	map of <code>tosca.datatypes.nfv.LogicalNodeData</code>		The Logical Node requirements.
<code>request_additional_capabilities</code>	no	map of <code>tosca.datatypes.nfv.RequestedAdditionalCapability</code>		Describes additional capability for a particular VDU.
<code>compute_requirements</code>	no	map of string		Describes compute requirements.
<code>virtual_memory</code>	yes	<code>tosca.datatypes.nfv.VirtualMemory</code>		Describes virtual memory of the virtualised compute.
<code>virtual_cpu</code>	yes	<code>tosca.datatypes.nfv.VirtualCpu</code>		Describes virtual CPU(s) of the virtualised compute.
<code>virtual_local_storage</code>	no	list of <code>tosca.datatypes.nfv.VirtualBlockStorageData</code>		A list of virtual system disks created and destroyed as part of the VM lifecycle.

6.4.3.3 Definition

The syntax of the VirtualCompute capability type shall comply with the following definition:

```

tosca.capabilities.nfv.VirtualCompute:
  derived_from: tosca.capabilities.Node
  description: Describes the capabilities related to virtual compute resources
  properties:
    logical_node:
      type: map
      description: Describes the Logical Node requirements
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.LogicalNodeData
    requested_additional_capabilities:
      type: map
      description: Describes additional capability for a particular VDU
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.RequestedAdditionalCapability
    compute_requirements:
      type: map
      required: false
      entry_schema:
        type: string
    virtual_memory:
      type: tosca.datatypes.nfv.VirtualMemory
      description: Describes virtual memory of the virtualised compute
      required: true
    virtual_cpu:
      type: tosca.datatypes.nfv.VirtualCpu
      description: Describes virtual CPU(s) of the virtualised compute
      required: true
    virtual_local_storage:
      type: list
      description: A list of virtual system disks created and destroyed as part of
the VM lifecycle
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.VirtualBlockStorageData
      description: virtual system disk definition

```

6.4.4 tosca.capabilities.nfv.VirtualStorage

6.4.4.1 Description

The VirtualStorage capability indicates that the node that includes it can be pointed by a `tosca.relationships.nfv.AttachesTo` relationship type which is used to model the `VduHasVirtualStorageDesc` association illustrated in ETSI GS NFV-IFA 011 [1]. Table 6.4.4.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.4.4.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualStorage
Type Qualified Name	tosca.nfv:VirtualStorage
Type URI	tosca.capabilities.nfv.VirtualStorage

6.4.4.2 Definition

The syntax of the VirtualStorage capability type shall comply with the following definition:

```
tosca.capabilities.nfv.VirtualStorage:
  derived_from: tosca.capabilities.Root
  description: Describes the attachment capabilities related to Vdu.Storage
```

6.4.5 tosca.capabilities.nfv.TrunkBindable

6.4.5.1 Description

The TrunkBindable capability indicates that the VduCp node that includes it can be pointed by a tosca.relationships.nfv.TrunkBindsTo relationship type which is used to model the trunkPortTopology illustrated in ETSI GS NFV-IFA 011 [1]. Table 6.4.5.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.4.5.1-1: Type name, shorthand, and URI

Shorthand Name	TrunkBindable
Type Qualified Name	toscanfv:TrunkBindable
Type URI	tosca.capabilities.nfv.TrunkBindable

6.4.5.2 Properties

None.

6.4.5.3 Definition

The syntax of the TrunkBindable capability type shall comply with the following definition:

```
tosca.capabilities.nfv.TrunkBindable:
  derived_from: tosca.capabilities.Node
  description: Indicates that the node that includes it can be pointed by a
  tosca.relationships.nfv.TrunkBindsTo relationship type which is used to model the
  trunkPortTopology.
```

6.4.6 tosca.capabilities.nfv.ContainerDeployable

6.4.6.1 Description

A node type that includes the ContainerDeployable capability indicates that it can be pointed by tosca.relationships.nfv.DeploysTo relationship type, which is used to model the grouping of Vdu.OsContainers into a Vdu.OsContainerDeployableUnit. Table 6.4.6.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.4.6.1-1: Type name, shorthand, and URI

Shorthand Name	ContainerDeployable
Type Qualified Name	toscanfv:ContainerDeployable
Type URI	tosca.capabilities.nfv.ContainerDeployable

6.4.6.2 Properties

None.

6.4.6.3 Definition

The syntax of the ContainerDeployable capability type shall comply with the following definition:

```
tosca.capabilities.nfv.ContainerDeployable:
  derived_from: tosca.capabilities.Node
  description: A node type that includes the ContainerDeployable capability
indicates that it can be pointed by tosca.relationships.nfv.DeploysTo relationship
type
```

6.4.7 tosca.capabilities.nfv.AssociableVdu

6.4.7.1 Description

The AssociableVdu capability indicates that the node that includes it can be pointed by a tosca.relationships.nfv.MciopAssociates relationship type which is used to model the associatedVdu property of the MciopProfile information element defined in ETSI GS NFV-IFA 011 [1]. Table 6.4.7.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.4.7.1-1: Type name, shorthand, and URI

Shorthand Name	AssociableVdu
Type Qualified Name	toscanfv:AssociableVdu
Type URI	tosca.capabilities.nfv.AssociableVdu

6.4.7.2 Properties

None.

6.4.7.3 Definition

The syntax of the AssociableVdu capability type shall comply with the following definition:

```
tosca.capabilities.nfv.AssociableVdu:
  derived_from: tosca.capabilities.Node
  description: Indicates that the node that includes it can be pointed by a
tosca.relationships.nfv.MciopAssociates relationship type which is used to model the
associatedVdu property of the MciopProfile information element defined in ETSI GS
NFV-IFA 011.
```

6.4.8 tosca.capabilities.nfv.DeployableModuleMember

6.4.8.1 Description

The DeployableModuleMember capability indicates that the node that includes it can be pointed by a tosca.relationships.nfv.DeployableModuleAssociates relationship type which is used to indicate that the pointed-to node belongs to a deployable module, represented by the pointing node. Table 6.4.8.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.4.8.1-1: Type name, shorthand, and URI

Shorthand Name	DeployableModuleMember
Type Qualified Name	toscanfv:DeployableModuleMember
Type URI	tosca.capabilities.nfv.DeployableModuleMember

6.4.8.2 Properties

None.

6.4.8.3 Definition

The syntax of the DeployableModuleMember capability type shall comply with the following definition:

```
tosca.capabilities.nfv.DeployableModuleMember:
  derived_from: tosca.capabilities.Node
  description: Indicates that the node that includes it can be pointed by a
tosca.relationships.nfv.DeployableModuleAssociates relationship type which is used to
indicate that the pointed-to node belongs to a deployable module, represented by the
pointing node.
```

6.4.9 tosca.capabilities.nfv.PaaSServiceRequestable

6.4.9.1 Description

The PaaSServiceRequestable capability indicates that the node that includes it can be pointed by a tosca.relationships.nfv.PaaSServiceRequestsTo relationship type. Table 6.4.9.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.4.9.1-1: Type name, shorthand, and URI

Shorthand Name	PaasServiceRequestable
Type Qualified Name	toscanfv:PaasServiceRequestable
Type URI	tosca.capabilities.nfv.PaasServiceRequestable

6.4.9.2 Properties

None.

6.4.9.3 Definition

The syntax of the PaaSServiceRequestable capability type shall comply with the following definition:

```
tosca.capabilities.nfv.PaaSServiceRequestable:
  derived_from: tosca.capabilities.Node
  description: Indicates that the node that includes it can be pointed by a
tosca.relationships.nfv.PaaSServiceRequestsTo relationship type.
```

6.4.10 tosca.capabilities.nfv.InstallableCertificate

6.4.10.1 Description

The InstallableCertificate capability indicates that the node that includes it can be pointed by a tosca.relationships.nfv.InstallsTo relationship type which is used to model the association between VDU and Certificate. Table 6.4.10.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.4.10.1-1: Type name, shorthand, and URI

Shorthand Name	InstallableCertificate
Type Qualified Name	toscanfv:InstallableCertificate
Type URI	tosca.capabilities.nfv.InstallableCertificate

6.4.10.2 Definition

The syntax of the InstallableCertificate capability type shall comply with the following definition:

```
tosca.capabilities.nfv.InstallableCertificate:
  derived_from: toska.capabilities.Root
  description: Describes the installation capabilities related to Certificate.
```

6.4.11 toska.capabilities.nfv.AssociablePaasService

6.4.11.1 Description

The AssociablePaasService capability type is defined in clause 9.4.3 of the present document.

6.5 Requirement Types

None.

6.6 Relationship Types

6.6.1 toska.relationships.nfv.VirtualBindsTo

6.6.1.1 Description

This relationship type represents an association between Vdu.Compute or Vdu.OsContainerDeployableUnit and VduCp node types. Table 6.6.1.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.6.1.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualBindsTo
Type Qualified Name	toscanfv:VirtualBindsTo
Type URI	tosca.relationships.nfv.VirtualBindsTo

6.6.1.2 Properties

None.

6.6.1.3 Definition

The syntax of the VirtualBindsTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.VirtualBindsTo:
  derived_from: toska.relationships.DependsOn
  description: Represents an association relationship between Vdu.Compute or
  Vdu.OsContainerDeployableUnit and VduCp node types
  valid_target_types: [ toska.capabilities.nfv.VirtualBindable ]
```

6.6.2 `tosca.relationships.nfv.VirtualLinksTo`

6.6.2.1 Description

The `VirtualLinksTo` relationship type is defined in clause 9.6.1 of the present document representing an association relationship between a `VduCp` and a `VnfVirtualLink` node type or a `VnfExtCp` and a `VnfVirtualLink` node type.

6.6.3 `tosca.relationships.nfv.AttachesTo`

6.6.3.1 Description

This relationship type represents an association between the `Vdu.Compute` or `Vdu.OsContainerDeployableUnit` and one of the following node types: `Vdu.VirtualBlockStorage`, `Vdu.VirtualObjectStorage` or `Vdu.VirtualFileStorage`. Table 6.6.3.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.6.3.1-1: Type name, shorthand, and URI

Shorthand Name	<code>AttachesTo</code>
Type Qualified Name	<code>toscanfv:AttachesTo</code>
Type URI	<code>tosca.relationships.nfv.AttachesTo</code>

6.6.3.2 Properties

None.

6.6.3.3 Definition

The syntax of the `AttachesTo` relationship type shall comply with the following definition:

```
tosca.relationships.nfv.AttachesTo:
  derived_from: toska.relationships.Root
  description: Represents an association relationship between the Vdu.Compute or
Vdu.OsContainerDeployableUnit and one of the node types, Vdu.VirtualBlockStorage,
Vdu.VirtualObjectStorage or Vdu.VirtualFileStorage
  valid_target_types: [ toska.capabilities.nfv.VirtualStorage ]
```

6.6.4 `tosca.relationships.nfv.TrunkBindsTo`

6.6.4.1 Description

This relationship type represents an association between a `VduCp` node used as a trunk port and other `VduSubCp` nodes used as subports of the same trunk. Table 6.6.4.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.6.4.1-1: Type name, shorthand, and URI

Shorthand Name	<code>TrunkBindsTo</code>
Type Qualified Name	<code>toscanfv:TrunkBindsTo</code>
Type URI	<code>tosca.relationships.nfv.TrunkBindsTo</code>

6.6.4.2 Properties

None.

6.6.4.3 Definition

The syntax of the TrunkBindsTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.TrunkBindsTo:
  derived_from: toska.relationships.DependsOn
  description: Represents the association relationship between a VduCp node used as
a trunk port and other VduSubCp nodes used as subports of the same trunk.
  valid_target_types: [ toska.capabilities.nfv.TrunkBindable ]
```

6.6.5 toska.relationships.nfv.DeploysTo

6.6.5.1 Description

This relationship type represents an association between Vdu.OsContainerDeployableUnit and Vdu.OsContainer node types. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.6.5.1-1: Type name, shorthand, and URI

Shorthand Name	DeploysTo
Type Qualified Name	toscanfv: DeploysTo
Type URI	tosca.relationships.nfv.DeploysTo

6.6.5.2 Properties

None.

6.6.5.3 Definition

The syntax of the DeployssTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.DeploysTo:
  derived_from: toska.relationships.DependsOn
  description: Represents an association relationship between
Vdu.OsContainerDeployableUnit and Vdu.OsContainer node types
  valid_target_types: [tosca.capabilities.nfv.ContainerDeployable]
```

6.6.6 toska.relationships.nfv.MciopAssociates

6.6.6.1 Description

This relationship type represents an association between Mciop and Vdu.OsContainerDeployableUnit node types. Table 6.6.6.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.6.6.1-1: Type name, shorthand, and URI

Shorthand Name	MciopAssociates
Type Qualified Name	toscanfv:MciopAssociates
Type URI	tosca.relationships.nfv.MciopAssociates

6.6.6.2 Properties

None.

6.6.6.3 Definition

The syntax of the MciopAssociates relationship type shall comply with the following definition:

```
tosca.relationships.nfv.MciopAssociates:
  derived_from: toska.relationships.DependsOn
  description: Represents an association relationship between Mciop and
Vdu.OsContainerDeployableUnit node types
  valid_target_types: [ toska.capabilities.nfv.AssociableVdu ]
```

6.6.7 toska.relationships.nfv.DeployableModuleAssociates

6.6.7.1 Description

This relationship type represents an association between a DeployableModule node type and either a Vdu.Compute node type or an Mciop node type. Table 6.6.7.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.6.7.1-1: Type name, shorthand, and URI

Shorthand Name	DeployableModuleAssociates
Type Qualified Name	toscanfv:DeployableModuleAssociates
Type URI	tosca.relationships.nfv.DeployableModuleAssociates

6.6.7.2 Properties

None.

6.6.7.3 Definition

The syntax of the DeployableModuleAssociates relationship type shall comply with the following definition:

```
tosca.relationships.nfv.DeployableModuleAssociates:
  derived_from: toska.relationships.DependsOn
  description: Represents an association relationship between a DeployableModule
node type and either a Vdu.Compute node type or an Mciop node type
  valid_target_types: [ toska.capabilities.nfv.DeployableModuleMember ]
```

6.6.8 toska.relationships.nfv.PaaSServiceRequestsTo

6.6.8.1 Description

This relationship type represents an association between PaasServiceProfile and PaasServiceRequest node types. Table 6.6.8.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.6.8.1-1: Type name, shorthand, and URI

Shorthand Name	PaasServiceRequestsTo
Type Qualified Name	toscanfv:PaasServiceRequestsTo
Type URI	tosca.relationships.nfv.PaaSServiceRequestsTo

6.6.8.2 Properties

None.

6.6.8.3 Definition

The syntax of the PaasServiceRequestsTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.PaasServiceRequestsTo:
  derived_from: toska.relationships.DependsOn
  description: Represents an association relationship between PaasServiceProfile and
PaasServiceRequest node types
  valid_target_types: [ toska.capabilities.nfv.PaasServiceRequestable ]
```

6.6.9 toska.relationships.nfv.InstallsTo

6.6.9.1 Description

This relationship type represents an association between Vdu.Compute or Vdu.OsContainerDeployableUnit and Certificate. Table 6.6.9.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.6.9.1-1: Type name, shorthand, and URI

Shorthand Name	InstallsTo
Type Qualified Name	toscanfv:InstallsTo
Type URI	tosca.relationships.nfv.InstallsTo

6.6.9.2 Properties

None.

6.6.9.3 Definition

The syntax of the InstallsTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.InstallsTo:
  derived_from: toska.relationships.Root
  description: Represents an association relationship between Vdu.Compute or
Vdu.OsContainerDeployableUnit and Certificate.
  valid_target_types: [ toska.capabilities.nfv.InstallableCertificate ]
```

6.6.10 toska.relationships.nfv.PaasServiceAssociates

6.6.10.1 Description

The PaasServiceAssociates relationship type is defined in clause 9.6.4 of the present document.

6.7 Interface Types

6.7.1 toska.interfaces.nfv.Vnflcm

6.7.1.1 Description

The toska.interfaces.nfv.Vnflcm contains a set of TOSCA operations corresponding to the following VNF LCM operations defined in ETSI GS NFV-IFA 007 [i.1]:

- Instantiate VNF

- Terminate VNF
- Modify VNF information
- Change VNF Flavour
- Change External VNF Connectivity
- Operate VNF
- Heal VNF
- Scale VNF
- Scale VNF To Level
- Create VNF Snapshot
- Revert to VNF Snapshot
- Change VNF current package

In addition, the VNFM shall also support TOSCA operations corresponding to preamble and postamble to the execution of the base operation. The name of these operations is constructed according to the following pattern:

- <base_operation_name>_start for a preamble
- <base_operation_name>_end for a postamble

The designations ("_start", "_end") in the name of TOSCA operations are postfixes so that related operations are adjacent in an alphabetical listing.

The `tosca.interfaces.nfv.Vnflcm` also contains a set of TOSCA notifications corresponding to the following VNF LCM operations defined in ETSI GS NFV-IFA 007 [i.1]:

- Change VNF current package

In addition, the VNFM shall also support TOSCA notifications corresponding to preamble and postamble to the base notification. The name of these notifications is constructed according to the following pattern:

- <base_notification_name>_start for a preamble
- <base_notification_name>_end for a postamble

Table 6.7.1.1-1 specifies the declared names for this interface type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.7.1.1-1: Type name, shorthand, and URI

Shorthand Name	Vnflcm
Type Qualified Name	toscanfv:Vnflcm
Type URI	tosca.interfaces.nfv.Vnflcm

6.7.1.2 Definition

The syntax of the Vnflcm interface type shall comply with the following definition:

```
tosca.interfaces.nfv.Vnflcm:
  derived_from: toska.interfaces.Root
  description: This interface encompasses a set of TOSCA operations corresponding to
the VNF LCM operations defined in ETSI GS NFV-IFA 007 as well as to preamble and
postamble procedures to the execution of the VNF LCM operations.
  operations:
    instantiate:
```

```

description: Invoked upon receipt of an Instantiate VNF request
inputs:
  additional_parameters:
    type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
    required: false
    # derived types are expected to introduce additional_parameters with its
    # type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
instantiate_start:
  description: Invoked before instantiate
instantiate_end:
  description: Invoked after instantiate
terminate:
  description: Invoked upon receipt Terminate VNF request
inputs:
  additional_parameters:
    type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
    required: false
    # derived types are expected to introduce additional_parameters with its
    # type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
terminate_start:
  description: Invoked before terminate
terminate_end:
  description: Invoked after terminate
modify_information:
  description: Invoked upon receipt of a Modify VNF Information request
modify_information_start:
  description: Invoked before modify_information
modify_information_end:
  description: Invoked after modify_information
change_flavour:
  description: Invoked upon receipt of a Change VNF Flavour request
inputs:
  additional_parameters:
    type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
    required: false
    # derived types are expected to introduce additional_parameters with its
    # type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
change_flavour_start:
  description: Invoked before change_flavour
change_flavour_end:
  description: Invoked after change_flavour
change_external_connectivity:
  description: Invoked upon receipt of a Change External VNF Connectivity
request
inputs:
  additional_parameters:
    type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
    required: false
    # derived types are expected to introduce additional_parameters with its
    # type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
change_external_connectivity_start:
  description: Invoked before change_external_connectivity
change_external_connectivity_end:
  description: Invoked after change_external_connectivity
operate:
  description: Invoked upon receipt of an Operate VNF request
inputs:
  additional_parameters:
    type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
    required: false
    # derived types are expected to introduce additional_parameters with its

```

```

    # type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
operate_start:
  description: Invoked before operate
operate_end:
  description: Invoked after operate
heal:
  description: Invoked upon receipt of a Heal VNF request
inputs:
  additional_parameters:
    type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
    required: false
  # derived types are expected to introduce additional_parameters with its
  # type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
  cause:
    type: string
    description: Indicates the reason why a healing procedure is required.
    required: false
  vnfc_instance_ids:
    type: list
    entry_schema:
      type: string
    description: List of VNFC instances requiring a healing action.
    required: false
heal_start:
  description: Invoked before heal
heal_end:
  description: Invoked after heal
scale:
  description: Invoked upon receipt of a Scale VNF request
inputs:
  additional_parameters:
    type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
    required: false
  # derived types are expected to introduce additional_parameters with its
  # type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
  type:
    type: string
    description: Indicates the type of the scale operation requested.
    required: false
    constraints:
      - valid_values: [ scale_out, scale_in ]
  aspect:
    type: string
    description: Identifier of the scaling aspect.
    required: false
  number_of_steps:
    type: integer
    description: Number of scaling steps to be executed.
    required: true
    constraints:
      - greater_than: 0
    default: 1
scale_start:
  description: Invoked before scale
scale_end:
  description: Invoked after scale
scale_to_level:
  description: Invoked upon receipt of a Scale VNF to Level request
inputs:
  additional_parameters:
    type: toasca.datatypes.nfv.VnfOperationAdditionalParameters

```

```

        required: false
        # derived types are expected to introduce additional_parameters with its
        # type derived from tosca.datatypes.nfv.VnfOperationAdditionalParameters
        instantiation_level:
            type: string
            description: Identifier of the target instantiation level of the current
deployment flavour to which the VNF is requested to be scaled. Either
instantiation_level or scale_info or power_profile shall be provided, but not multiple
of them.
        required: false
        scale_info:
            type: map # key: aspectId
            description: For each scaling aspect of the current deployment flavour,
indicates the target scale level to which the VNF is to be scaled. Either
instantiation_level or scale_info or power_profile shall be provided, but not multiple
of them.
        required: false
        entry_schema:
            type: tosca.datatypes.nfv.ScaleInfo
        power_profile:
            type: string
            description: Identifier of the target power profile in the current
deployment flavour, as defined in the VNFD, from which the target power state can be
selected and be switched to. Either instantiation_level or scale_info or power_profile
shall be provided, but not multiple of them.
        required: false
        scale_to_level_start:
            description: Invoked before scale_to_level
        scale_to_level_end:
            description: Invoked after scale_to_level
        create_snapshot:
            description: Invoked upon receipt of a Create VNF snapshot request
        inputs:
            additional_parameters:
                type: tosca.datatypes.nfv.VnfOperationAdditionalParameters
                required: false
            # derived types are expected to introduce additional_parameters with its
            # type derived from tosca.datatypes.nfv.VnfOperationAdditionalParameters
        create_snapshot_start:
            description: Invoked before create_snapshot
        create_snapshot_end:
            description: Invoked after create_snapshot
        revert_to_snapshot:
            description: Invoked upon receipt of a Revert to VNF snapshot request
        inputs:
            additional_parameters:
                type: tosca.datatypes.nfv.VnfOperationAdditionalParameters
                required: false
            # derived types are expected to introduce additional_parameters with its
            # type derived from tosca.datatypes.nfv.VnfOperationAdditionalParameters
        revert_to_snapshot_start:
            description: Invoked before revert_to_snapshot
        revert_to_snapshot_end:
            description: Invoked after revert_to_snapshot
        change_current_package:
            description: Invoked by tosca.policies.nfv.VnfPackageChange
        inputs:
            additional_parameters:
                type: tosca.datatypes.nfv.VnfOperationAdditionalParameters
                required: false
            # derived types are expected to introduce additional_parameters with its

```

```

    # type derived from toska.datatypes.nfv.VnfOperationAdditionalParameters
    change_current_package_start:
      description: Invoked by toska.policies.nfv.VnfPackageChange
    change_current_package_end:
      description: Invoked by toska.policies.nfv.VnfPackageChange
  notifications:
    change_current_package_notification:
      description: Invoked upon receipt of a ChangeCurrentVnfPackage request
    change_current_package_start_notification:
      description: Invoked before the operation designated to changing the current
VNF package
    change_current_package_end_notification:
      description: Invoked after the operation designated to changing the current
VNF package

```

6.7.1.3 Additional Requirements

All VNF supported LCM operations shall be listed in the service template, except "instantiate" and "terminate" that may be omitted, as specified in ETSI GS NFV-IFA 011 [1] for the supportedOperation attribute of a deployment flavour.

The implementation and inputs keynames specified in TOSCA-Simple-Profile-YAML-v1.3 [20] for an operation definition may be included for each operation listed in the Vnflcm interface definition.

If a TOSCA operation representing a VNF LCM operation is listed in the service template without an associated implementation, then it means that:

- the VNF LCM operation is supported (i.e. this is the manifestation of the supportedOperation attribute as per ETSI GS NFV-IFA 011 [1]); and
- the processing logic associated with the LCM operation is the default implementation provided by the VNFM.

If an implementation is associated to a TOSCA operation that represents a preamble or a postamble to a VNF LCM operation, the implementation logic is executed before or after the execution of the VNF LCM operation implementation, respectively.

The VNFM shall make available all parameters from the message invoking the VNF LCM operation as inputs to the corresponding TOSCA interface operations.

In the operation definitions on the Vnflcm interface, the additional_parameters (VNF-specific extension of the toska.datatypes.nfv.VnfOperationAdditionalParameters) of the inputs section describes the name and type of the additional parameters (additionalParams) that can be submitted in the VNF LCM operation request. See an example in clause 6.2.43 (toska.datatypes.nfv.VnfOperationAdditionalParameters).

The inputs keyname can also be used to specify additional input parameters for executing the TOSCA operation, beyond those received in the VNF LCM operation request. To distinguish them from the latter ones, such input parameters shall not be named "additional_parameters".

The implementation of the operation corresponding to preamble and postamble TOSCA operations (instantiate_start, instantiate_end, scale_start, scale_end, etc.), if present, shall be invoked with the same parameters as the corresponding base operations ones (instantiate, scale, etc.). The inputs of the operations corresponding to the postamble and preamble operations shall not be defined in the VNFD.

Starting with version 3.3.1 of the present document, the Vnflcm interface type definition grammar was changed to support notifications and operations. For backward compatibility, TOSCA-Simple-Profile-YAML-v1.3 [20], clause 3.7.5.5 specifies the provisions for handling the previous grammar. Support of the Release 2 Vnflcm interface type definition grammar can be removed in subsequent versions of the present document.

6.7.1.4 Support of LCM scripts

In ETSI GS NFV-IFA 011 [1], the definition of the "LifeCycleManagementScript" information element of the VNFD associates scripts with events, where an event can be an external or an internal stimulus. These events are mapped to TOSCA operations of the VNF node type in the following way:

- external stimuli are mapped to TOSCA operations corresponding to the VNF LCM operations defined in ETSI GS NFV-IFA 007 [i.1];
- internal stimuli are mapped to preamble and postamble of these TOSCA operations;
- events that cannot be mapped to these TOSCA operations (lcmTransitionEvent as described in ETSI GS NFV-IFA 011 [1], clause 7.1.13) can be mapped to further TOSCA operations by extending the TOSCA interface.

The LCM scripts can be regarded as artifacts that provide a VNF-specific implementation of the TOSCA operation corresponding to the stimulus. The script input parameters shall be provided to the script according to the declaration in the inputs field of the operation definition. The artifact type definition shall enable identifying the DSL used by the script. The artifact type definition for Python is provided in section 5.4.4.1 of TOSCA-Simple-Profile-YAML-v1.3 [20]. The artifact definition for Mistral is provided in clause 6.3.2 of the present document.

NOTE 1: As all input parameters needed for operations corresponding to external and internal stimuli are defined in the "input parameters of the external stimuli operations", the VNF Designer is expected to make the list of parameters as complete as needed to handle not only the external stimuli but also the internal stimuli.

NOTE 2: The stimulus, external or internal, that triggers the Scale VNF operation, or its preamble or postamble, can result in a horizontal or a vertical scaling, i.e. there is not different VNF lifecycle stimulus for each of the two scale modes. It is up to the script logic to analyse the input of the VNF LCM operation (i.e. the 'type' attribute, see clause 5.5.2.5 in ETSI GS NFV-SOL 003 [25]) and implement the corresponding function, horizontal or vertical scaling.

6.7.1.5 Examples

The following example template fragments illustrate the concepts.

```
tosca_definitions_version: tosca_simple_yaml_1_3

imports:
- ..

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    ..

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    ..

  node_templates:
    SunshineDB:
      type: MyCompany.SunshineDB.1_0.1_0
    ..

    interfaces:
      Vnflcm:
        operations:
          instantiate: {}
    ..
```

In the above example, as there is no implementation and inputs specified to the operations, the built-in implementation of the operation is invoked when the Instantiate VNF request is received on the LCM interface of the Or-Vnfm reference point. The received parameters (flavourId, instantiationLevelId, etc.) are passed to the built-in implementation (as flavour_id, instantiation_level_id).


```

tosca_definitions_version: tosca_simple_yaml_1_3

imports:
- ..

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    ..

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    ..

  node_templates:
    SunshineDB:
      type: MyCompany.SunshineDB.1_0.1_0
      ..
      interfaces:
        Vnflcm:
          operations:
            instantiate:
              implementation: instantiate-script
            ..
      ..

```

In the above example, the `instantiate-script` is invoked when the `Instantiate VNF` request is received, passing the received parameters to it similarly to the previous example. This example does not imply a one-to-one mapping between operations and script names.

```

tosca_definitions_version: tosca_simple_yaml_1_3

imports:
- ..

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    ..

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    ..

  node_templates:
    SunshineDB:
      type: MyCompany.SunshineDB.1_0.1_0
      ..
      interfaces:
        Vnflcm:
          operations:
            ..
            scale_start: pre-scale-script
            scale_end: post-scale-script
            ..
      ..

```

In the above example, LCM scripts are associated with the "scale start" and "scale end" internal stimuli. As no script is associated to the `scale` operation, its default implementation runs (after running the `pre-scale-script`, and before running the `post-scale-script`).

```
tosca_definitions_version: tosca_simple_yaml_1_3

imports:
- ..

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    ..

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    ..

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    ..
    interfaces:
      Vnflcm:
        operations:
          instantiate:
            implementation: instantiate-script
            inputs:
              script_input_1: value_1
              script_input_2: value_2

  artifacts:
    instantiate-script:
      description: Instantiate workflow script
      type: tosca.artifacts.Implementation.Python
      file: instantiate.py
      #repository: ..
      #deploy_path: ..
    ..
```

In the above example:

- The inputs section provides additional input values to the `instantiate-script` (i.e. the manifestation of the `scriptInput` attribute of `LifeCycleManagementScript` as defined in ETSI GS NFV-IFA 011 [1]).

NOTE: There is another kind of input called `additional_parameters` dedicated to the additional parameters (`additionalParams`) received in the message invoking the VNF LCM operation; this input is not illustrated by the above examples; see clause 6.2.43.4 on how to declare `additional_parameters` in the derived VNF node type.

- TOSCA artifacts definition is used to convey the type of DSL used as a scripting language that is associated with an operation (i.e. the manifestation of the `scriptDsl` attribute of `LifeCycleManagementScript` as per ETSI GS NFV-IFA 011 [1]).

6.7.2 `tosca.interfaces.nfv.VnfIndicator`

6.7.2.1 Description

The `tosca.interfaces.nfv.VnfIndicator` is an empty base interface type for deriving VNF specific interface types that include VNF indicator specific notifications.

Table 6.7.2.1-1 specifies the declared names for this interface type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.7.2.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfIndicator</code>
Type Qualified Name	<code>toscanfv:VnfIndicator</code>
Type URI	<code>tosca.interfaces.nfv.VnfIndicator</code>

6.7.2.2 Definition

The syntax of the `VnfIndicator` interface type shall comply with the following definition:

```
tosca.interfaces.nfv.VnfIndicator:
  derived_from: toska.interfaces.Root
  description: This interface is an empty base interface type for deriving VNF
specific interface types that include VNF indicator specific notifications.
```

6.7.2.3 Examples

See clause 6.8.1.9.

6.7.3 `tosca.interfaces.nfv.ChangeCurrentVnfPackage`

6.7.3.1 Description

The `tosca.interfaces.nfv.ChangeCurrentVnfPackage` is an empty base interface type for deriving VNF specific interface types that include VNF Change Current VNF Package specific operation.

Table 6.7.3.1-1 specifies the declared names for this interface type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.7.3.1-1: Type name, shorthand, and URI

Shorthand Name	<code>ChangeCurrentVnfPackage</code>
Type Qualified Name	<code>toscanfv:ChangeCurrentVnfPackage</code>
Type URI	<code>tosca.interfaces.nfv.ChangeCurrentVnfPackage</code>

6.7.3.2 Definition

The syntax of the `ChangeCurrentVnfPackage` interface type shall comply with the following definition:

```
tosca.interfaces.nfv.ChangeCurrentVnfPackage:
  derived_from: toska.interfaces.Root
  description: This interface is an empty base interface type for deriving VNF
specific interface types that include VNF Change Current VNF Package specific
operation.
  # operations:
    # operation_name: name of a VNF-specific operation serving the Change current
VNF Package request.
```

```
# description: Invoked by tosca.policies.nfv.VnfPackageChange
# inputs:
#   additional_parameters:
#     type: tosca.datatypes.nfv.VnfOperationAdditionalParameters
#     required: false
# derived types are expected to introduce additional_parameters with its
# type derived from tosca.datatypes.nfv.VnfOperationAdditionalParameters
```

6.7.3.3 Examples

See clause 6.10.15.5.

6.8 Node Types

6.8.1 tosca.nodes.nfv.VNF

6.8.1.1 Description

The VNF node type is the generic abstract type from which all VNF specific node types shall be derived to form, together with other node types, the TOSCA service template(s) representing the VNFD information element as defined in ETSI GS NFV-IFA 011 [1]. Table 6.8.1.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.1.1-1: Type name, shorthand, and URI

Shorthand Name	VNF
Type Qualified Name	toscanfv:VNF
Type URI	tosca.nodes.nfv.VNF

6.8.1.2 Properties

The properties of the VNF node type shall comply with the provisions set out in table 6.8.1.2-1.

Table 6.8.1.2-1: Properties

Name	Required	Type	Constraints	Description
descriptor_id	yes	string		Identifier of this VNFD information element. This attribute shall be globally unique. See note 3. The VNFD Identifier shall be used as the unique identifier of the VNF Package that contains this VNFD. Any modification of the content of the VNFD or the VNF Package shall result in a new VNFD Identifier.
ext_invariant_id	no	string		Identifies the VNFD in a version independent manner. This property is invariant across versions of the VNFD that fulfil certain conditions related to the external connectivity and management of the VNF. See notes 3 and 6.

Name	Required	Type	Constraints	Description
				When used in a VNF node template in an NSD it allows for VNF instances during NS LCM the use of a VNFD different from the one referenced by the descriptor_id property, provided they have the same ext_invariant_id. See note 7.
descriptor_version	yes	string		Identifies the version of the VNFD.
provider	yes	string		Provider of the VNF and of the VNFD.
product_name	yes	string		Name to identify the VNF Product. Invariant for the VNF Product lifetime.
software_version	yes	string		Software version of the VNF. This is changed when there is any change to the software that is included in the VNF Package.
product_info_name	no	string		Human readable name for the VNF Product. Can change during the VNF Product lifetime.
product_info_description	no	string		Human readable description of the VNF Product. Can change during the VNF Product lifetime.
vnfm_info	yes	list of string		Identifies VNFM(s) compatible with the VNF described in this version of the VNFD. To indicate that a VNF can be managed by any ETSI NFV compliant VNFM, the string value shall be the concatenation of the string "etsivnfm" and the minimum version of ETSI GS NFV-SOL 002 [22] to be supported by this VNFM (e.g. etsivnfm:v2.3.1). If the VNF is compatible with multiple versions, multiple values may be included. See note 1. To indicate a specific VNFM product, the string value shall be the concatenation of the IANA enterprise number of the VNFM provider [5], followed by a product-specific string.
localization_languages	no	list of string	Valid values: string values that comply with IETF RFC 5646 [13]	Information about localization languages of the VNF (includes e.g. strings in the VNFD). This allows to provide one or more localization languages to support selecting a specific localization language at VNF instantiation time.
default_localization_language	no	string	Valid values: string values that comply with IETF RFC 5646 [13]	Default localization language that is instantiated if no information about selected localization language is available. Shall be present if "localizationLanguage" is present and shall be absent otherwise.
configurable_properties	no	tosca.datatypes.nfv.VnfConfigurableProperties		Describes the configurable properties of the VNF (e.g. related to auto scaling and auto healing).
modifiable_attributes	no	tosca.datatypes.nfv.VnfInfoModifiableAttributes		Describes the modifiable attributes of the VNF.

Name	Required	Type	Constraints	Description
lcm_operations_configuration	no	tosca.datatypes.nfv.VnfLcmOperationConfiguration		Describes the configuration parameters for the VNF LCM operations.
monitoring_parameters	no	map of tosc.datatypes.nfv.VnfMonitoringParameter		Describes monitoring parameters applicable to the VNF. See note 4 and note 5.
flavour_id	yes	string		Identifier of this DF within the VNFD.
flavour_description	yes	string		Human readable description of the DF.
vnf_profile	no	tosca.datatypes.nfv.VnfProfile		Describes a profile for instantiating VNFs of a particular NS DF according to a specific VNFD and VNF DF. See note 2.
change_selected_deployable_modules_op	no	list of string	Valid values: See YAML definition constraints	Indicates in which VNF LCM operations in this DF the VNF supports the change of the selected deployable modules. When change VNF DF or change current VNF package is indicated, it refers to change of DF or VNF package, respectively, to the one where the attribute is indicated.
NOTE 1: When LCM scripts are used, the support of this minimum version might not be sufficient to ensure that the VNF can be managed by a VNFM. The support of the domain specific language(s) used by these LCM scripts is another criterion for determining the compatibility of the VNF with a VNFM. NOTE 2: This property is only used in an NSD service template when describing a VNF node template with the corresponding VnfProfile information. NOTE 3: The value of the descriptor_id string shall comply with an UUID format as specified in section 3 of [9]. NOTE 4: This property is only used in a VNFD service template when describing a VNF node template with the corresponding monitoring information. NOTE 5: This property shall not be present in a VNFD service template when all the virtualisation containers of the VNF are realized as OsContainers. NOTE 6: Different versions of a VNFD have different descriptor_ids but can have the same ext_invariant_id. Different versions of the VNFD with the same ext_invariant_id shall have the same number and name of VNF deployment flavours, where each of them exposes: • same external connectivity, i.e. same number and name of the requirements for VirtualLinkable capability that represent external connection points • same number and name of VNF instantiation levels NOTE 6a: The content of each VNF instantiation level may change. • same VNF scaling aspects and same number of levels per aspect NOTE 6b: The constituents of each scaling aspect and the deltas between levels may change. • same VNF indicators: same attribute names and possible values NOTE 6c: This version of the present document does not support the indication of the possible values a VNF indicator can take. However, it supports Vnfindicator policies where conditions on specific values that a VNF indicator may take can be specified (see auto-scale and auto-heal policies in clause A.15.2). Conditions related to VNF indicator attributes in those policies shall be preserved unchanged across VNFDs with the same ext_invariant_id, since those values may be used for designing NsAutoScale policies in the NSD. This condition implies that VNFDs with the same ext_invariant_id preserve external invariance. Therefore, fulfilling this condition allows to use a different version of a VNFD in an NS instance without modification of the NSD on which the NS instance is based. The use of a different version is ultimately under the control of the service provider and it should consider if the NSD fulfils the requirements of the VnfExtCps (e.g. bitrate, IP version, etc.). NOTE 7: When the VNF node is used in an NSD, this property may only be included if the VNF node type definition in the VNFD includes the property with a value. If the property is supported in the VNF node in the VNFD, it should be included in the VNF node in the NSD to avoid changes in the NSD caused by version changes in the VNFD.				

The syntax of the `vnfm_info` string values shall comply with the following ABNF [6] snippet:

```
value = any_etsi_nfv_compliant_product | product_specific

any_etsi_nfv_compliant_product = "etsivnfm" SEP version

version = "v" version_identifier
version_identifier = 1*2DIGIT DOT 1*2DIGIT DOT 1*2DIGIT
; the version identifier is encoded as a sequence of items of 1 or 2 digits separated by dots
representing the 3 fields (major, technical and editorial) of the version of an ETSI deliverable.

product_specific = enterprise_number SEP product_specific_string
enterprise_number = 1*DIGIT
product_specific_string = *(ALPHA / DIGIT / "-" / ".")

SEP = ":"
DOT = "."
```

This implies that `vnfm_info` string values shall also comply with the pattern defined by the following regular expression [15]: `(^etsivnfm:v[0-9]?[0-9]\.[0-9]?[0-9]\.[0-9]?[0-9]$)|(^([0-9]+:[a-zA-Z0-9.-]+)$)`.

6.8.1.3 Attributes

The attribute of the VNF node type shall comply with the provisions set out in table 6.8.1.3-1.

Table 6.8.1.3-1: Attributes

Name	Required	Type	Constraints	Description
scale_status	no	map of tosca.datatypes.nfv. ScaleInfo		Scale status of the VNF, one entry per aspect. Represents for every scaling aspect how "big" the VNF has been scaled with reference to that aspect.

If the VNF supports VNF indicators, the VNF node type definition shall include one TOSCA attribute of a primitive type for each supported VNF indicator.

NOTE 1: In this version of the present document, the type of VNF indicators is constrained to primitive types. This is due to the limitations in the TOSCA-Simple-Profile-YAML-v1.3 [20] to define conditions on attributes of complex types.

If the `scale_status` attribute is used in a VNF indicator policy, e.g. an auto-scale policy, the VNF specific node type definitions may include additional attribute definitions of type integer, one for each scaling aspect. See example in clause A.15.

NOTE 2: As the `scale_status` attribute is complex, the `scale_level` property of the individual scaling aspects can be retrieved by passing a path to the `get_attribute` function: `{ get_attribute: [ SELF, scale_status, {scaling_aspect}, scale_level ] }`. If the value of the `scale_level` property is needed in a constraint (`tosca.policies.nfv.VnfIndicator`), then the value can be retrieved in an indirect way by accessing the aforementioned additional attributes. This is due to the limitation mentioned in the previous note.

VNF indicators may be defined in the VNFD to allow for the asynchronous notification of VNF specific information to the VNFM.

An attribute defined in the VNF node type for a VNF indicator holds the value for that indicator during the lifecycle of the VNF. A notification defined in the derived interface for VNF indicators (see clause 6.7.2) produces an output value which is assigned to the attribute, as per TOSCA-Simple-Profile-YAML-v1.3 [20] syntax. Examples of such assignments are shown in clause 6.8.1.9. Thus, the value of the VNF indicator may change every time a notification is received.

6.8.1.4 Requirements

The requirements of the VNF node type shall comply with the provisions set out in table 6.8.1.4-1.

Table 6.8.1.4-1: Requirements

Name	Required	Capability type	Constraints	Description
virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Describes the requirements for linking to virtual link
associated_paas_service	no	tosca.capabilities.AssociablePaasService		Describes the requirements for associating to a PaaS Service.

6.8.1.5 Capabilities

None.

6.8.1.6 Definition

The syntax of the VNF node type shall comply with the following definition:

```

tosca.nodes.nfv.VNF:
  derived_from: tosca.nodes.Root
  description: The generic abstract type from which all VNF specific node types
shall be derived to form, together with other node types, the TOSCA service
template(s) representing the VNFD
  properties:
    descriptor_id: # instead of vnfd_id
      type: string # UUID
      description: Identifier of this VNFD information element. This attribute shall
be globally unique
      required: true
    ext_invariant_id:
      type: string # UUID
      description: Identifies the VNFD in a version independent manner. This
property is invariant across versions of the VNFD that fulfil certain conditions
related to the external connectivity and management of the VNF. When used in a VNF
node template in an NSD it allows for VNF instances during NS LCM the use of a VNFD
different from the one referenced by the descriptor_id property, provided they have
the same ext_invariant_id. This attribute shall be globally unique.
      required: false
    descriptor_version: # instead of vnfd_version
      type: string
      description: Identifies the version of the VNFD
      required: true
    provider: # instead of vnf_provider
      type: string
      description: Provider of the VNF and of the VNFD
      required: true
    product_name: # instead of vnf_product_name
      type: string
      description: Human readable name for the VNF Product
      required: true
    software_version: # instead of vnf_software_version
      type: string
      description: Software version of the VNF
      required: true
    product_info_name: # instead of vnf_product_info_name
      type: string
      description: Human readable name for the VNF Product
      required: false
    product_info_description: # instead of vnf_product_info_description
      type: string
      description: Human readable description of the VNF Product

```



```

    required: false
  vnfm_info:
    type: list
    required: true
    description: Identifies VNFM(s) compatible with the VNF
    entry_schema:
      type: string
      constraints:
        - pattern: (^etsivnfm:v[0-9]?[0-9]\.[0-9]?[0-9]\.[0-9]?[0-9]$)|(^([0-9]+:[a-zA-Z0-9.-]+)$)
    localization_languages:
      type: list
      description: Information about localization languages of the VNF
      required: false
      entry_schema:
        type: string #IETF RFC 5646 string
    default_localization_language:
      type: string #IETF RFC 5646 string
      description: Default localization language that is instantiated if no
information about selected localization language is available
      required: false
    configurable_properties:
      type: tosa.datatypes.nfv.VnfConfigurableProperties
      description: Describes the configurable properties of the VNF
      required: false
    # derived types are expected to introduce configurable_properties
    # with its type derived from tosa.datatypes.nfv.VnfConfigurableProperties
    modifiable_attributes:
      type: tosa.datatypes.nfv.VnfInfoModifiableAttributes
      description: Describes the modifiable attributes of the VNF
      required: false
    # derived types are expected to introduce modifiable_attributes
    # with its type derived from
    # tosa.datatypes.nfv.VnfInfoModifiableAttributes
    lcm_operations_configuration:
      type: tosa.datatypes.nfv.VnfLcmOperationsConfiguration
      description: Describes the configuration parameters for the VNF LCM operations
      required: false
    monitoring_parameters:
      type: map # key: id
      entry_schema:
        type: tosa.datatypes.nfv.VnfMonitoringParameter
        description: Describes monitoring parameters applicable to the VNF.
        required: false
    flavour_id:
      type: string
      description: Identifier of the Deployment Flavour within the VNFD
      required: true
    flavour_description:
      type: string
      description: Human readable description of the DF
      required: true
    vnf_profile:
      type: tosa.datatypes.nfv.VnfProfile
      description: Describes a profile for instantiating VNFs of a particular NS DF
according to a specific VNFD and VNF DF
      required: false
    change_selected_deployable_modules_op:
      type: list
      description: Indicates in which VNF LCM operations in this DF the VNF supports
the change of the selected deployable modules. When CHANGE_VNF_DF or

```

```

CHANGE_CURRENT_VNF_PACKAGE is indicated, it refers to change of DF or VNF package,
respectively, to the one where the attribute is indicated.
    required: false
    entry_schema:
        type: string
        constraints:
            - valid_values: [ CHANGE_VNF_DF, CHANGE_CURRENT_VNF_PACKAGE, SELECT_DEP_MOD
]
    attributes:
        scale_status:
            type: map # key: aspectId
            description: Scale status of the VNF, one entry per aspect. Represents for
every scaling aspect how "big" the VNF has been scaled w.r.t. that aspect.
            entry_schema:
                type: tosca.datatypes.nfv.ScaleInfo
        requirements:
            - virtual_link:
                capability: tosca.capabilities.nfv.VirtualLinkable
                relationship: tosca.relationships.nfv.VirtualLinksTo
                occurrences: [ 0, 1 ]
            - associated_paas_service:
                capability: tosca.capabilities.nfv.AssociablePaasService
                relationship: tosca.relationships.nfv.PaasServiceAssociates
                occurrences: [ 0, UNBOUNDED ]
        # Additional requirements shall be defined in the VNF specific node type (deriving
from tosca.nodes.nfv.VNF) corresponding to NS virtual links that need to connect to
VnfExtCps
        interfaces:
            Vnflcm:
                type: tosca.interfaces.nfv.Vnflcm
            VnfIndicator:
                type: tosca.interfaces.nfv.VnfIndicator
        # derived types are expected to introduce Vnf Indicator interfaces
        # with their type derived from tosca.interfaces.nfv.VnfIndicator

```

6.8.1.7 Artifact

None.

6.8.1.8 Additional Requirements

For a given VNFD, a new VNF node type shall be defined following the below requirements:

- a) The node type shall be derived from: `tosca.nodes.nfv.VNF`.
- b) The following properties listed in `tosca.nodes.nfv.VNF` where the "required:" field is set to "true" shall be included with their values indicated as constraints and as default values or assigned as final fixed values if only one value is permitted (see clause 6.8.1.9 for an example):
 - a. `descriptor_id`
 - b. `descriptor_version`
 - c. `provider`
 - d. `product_name`
 - e. `software_version`
 - f. `vnfm_info`

g. flavour_id

NOTE 1: Indicating their values as default or assigning them a fixed value allows not to include them in property assignments in node templates, e.g. in the NSD.

NOTE 2: Assignment of a fixed value to the flavour_id property is not applicable if multiple deployment flavours exist.

- c) An empty string shall be indicated as the default value of the flavour_description property, without providing constraints.
- d) The capabilities, requirements, interfaces of toasca.nodes.nfv.VNF shall be preserved.
- e) Depending on the number of external connection points of the VNF that need to connect to NS virtual links, additional requirements for VirtualLinkable capability shall be defined with the occurrences set to [0, 1]. In this case, it is the VNFD author's choice to use the requirement for VirtualLinkable capability defined in the toasca.nodes.nfv.VNF node type or use only the additional requirements defined in the derived VNF specific node type. In the latter case, the virtual_link requirement should be included in the node type definition with occurrences [0, 0].

If the external connection point exposes a VipCp, a new requirement for VirtualLinkableCapability using the VipVirtualLinksTo relationship shall be defined for this connection point.

- f) The rule for naming this node type in the service template should be:
 - provider.product_name.software_version.descriptor_version, by concatenating the values of the corresponding properties of the created VNF node type.

NOTE 3: If the software_version value or descriptor_version value contains a dot (i.e. "."), this character should be replaced with an underscore (i.e. "_").

- g) If the VNF supports VNF indicators, the VNF node type definition shall include an interface definition of a VNF specific interface type indicating the mapping of notification outputs to the VNF node attributes and, optionally, toasca.policies.nfv.VnfIndicator policies that may invoke auto-scale or auto-heal operations. For each of the VNF indicators, the name of the notification output shall be the same as the name of the corresponding VNF attribute.

NOTE 4: The notifications keyname in TOSCA interface is defined in TOSCA-Simple-Profile-YAML-v1.3 [20].

- h) If "additionalParams" are expected in the Change current VNF Package request on the API (ETSI GS NFV-SOL 003 [25] or ETSI GS NFV-SOL 002 [22]), then they shall be defined as "additional_parameters" inputs of the change_current_package operation on the Vnflcm interface (in case the same LCM script with the same set of "additionalParams" is suitable for all change paths) or the VNF-specific operations on the ChangeCurrentVnfPackage interface (in case different change paths require different LCM scripts potentially with different sets of "additionalParams").
- i) If the VNFD supports external invariancy the VNF node type definition shall include the ext_invariant_id property with its value indicated as constraint.

VNF Providers shall use the following types to derive the VNF specific modifiable attributes and additional configurable properties:

- toasca.datatypes.nfv.VnfInfoModifiableAttributesExtensions.
- toasca.datatypes.nfv.VnfInfoModifiableAttributesMetadata.
- toasca.datatypes.nfv.VnfAdditionalConfigurableProperties.
- toasca.datatypes.nfv.VnfInfoModifiableAttributes.
- toasca.datatypes.nfv.VnfConfigurableProperties.

See illustrative examples in clauses 6.8.1.9, 6.2.35.4 and 6.2.31.4.

In the derived VNF node type, the `modifiable_attributes` and `configurable_properties` (VNF-specific extension of the `tosca.datatypes.nfv.VnfInfoModifiableAttributes` and `tosca.datatypes.nfv.VnfConfigurableProperties`, respectively, by extending the above listed types) describe the name and type of the modifiable attributes (extensions and metadata) and configurable properties (`vnfConfigurableProperties`).

The `modifiable_attributes` and `configurable_properties` information provided in the node type is sufficient for the client of the VNF LCM API for providing values to these properties. A value provided via the VNF LCM API to such a property overrides the value (if any) assigned in the node template or defined as default value in the node type definition.

Node templates of the VNF specific node type shall not include the `vnf_profile` property when they are part of a VNFD service template.

For a given NSD, when describing a referenced VNFD as a node templates, the `vnf_profile` property shall be included with a valid value.

For a given NSD, when describing a referenced VNFD as a node templates, the `monitoring_parameters` property shall not be included.

6.8.1.9 Example

Example usage of `modifiable_attributes`:

```
tosca_definitions_version: tosa_simple_yaml_1_3

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosa.nodes.nfv.VNF
    properties:
      flavour_id:
        constraints:
          - valid_values: [ simple, complex ]
    modifiable_attributes:
      type: mycompany.datatypes.nfv.VnfInfoModifiableAttributes

data_types:
  mycompany.datatypes.nfv.VnfInfoModifiableAttributes:
    derived_from: tosa.datatypes.nfv.VnfInfoModifiableAttributes
    properties:
      extensions:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions

  mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions:
    derived_from: tosa.datatypes.nfv.VnfInfoModifiableAttributesExtensions
    properties:
      http_proxy:
        type: string
        required: true
      https_proxy:
        type: string
        required: false
```

Example usage of `lcm_operations_configuration`:

Top level service template:

```
tosca_definitions_version: tosa_simple_yaml_1_3

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosa.nodes.nfv.VNF
```

```

properties:
  flavour_id:
    type: string
    constraints:
      - valid_values: [ simple, complex ]
  lcm_operations_configuration:
    scale:
      scaling_by_more_than_one_step_supported: true
    scale_to_level:
      arbitrary_target_levels_supported: true
    heal:
      causes:
        - service_unavailable
        - performance_degraded
    terminate:
      min_graceful_termination_timeout: 60 s
      max_recommended_graceful_termination_timeout: 600 s
    operate:
      min_graceful_stop_timeout: 60 s
      max_recommended_graceful_stop_timeout: 600 s

```

Example usage of describing a VNF node template with `vnf_profile` in an NSD TOSCA service template:

```

tosca_definitions_version: tosca_simple_yaml_1_3

description: an example of NSD TOSCA service template

topology template:
...

node_templates:
  VNF_1:
    type: tosca.nodes.nfv.exampleVNF
    properties:
      flavour_id: small
      descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider: MyCompany
      product_name: SunshineDB
      software_version: 1.0
      descriptor_version: 1.0
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 2
        max_number_of_instances: 6
      # other properties omitted for brevity
    requirements:
      - virtual_link: NsVirtualLink

```

Example usage of VNF indicators attributes in VNF node type definition and VNF indicator notifications in interface definition:

```

tosca_definitions_version: tosca_simple_yaml_1_3

interface_types:
  tosca.interfaces.nfv.MyCompanyVnfIndicator
    derived_from: tosca.interfaces.nfv.VnfIndicator
    notifications:
      health:
        description: this notification is used to received asynchronous

```

```

information of value change of the health_vnf_indicator
  utilization:
    description: this notification is used to received asynchronous
information of value change of the utilization_vnf_indicator

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      flavour_id:
        constraints:
          - valid_values: [ simple, complex ]
      modifiable_attributes:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributes
    attributes:
      health_vnf_indicator:
        type: string
        constraints:
          - valid_values: [ green, red, yellow ]
      utilization_vnf_indicator:
        type: float
        constraints:
          - in_range [ 0.0, 100.0 ]
    interfaces:
      Vnflcm:
        type: tosca.interfaces.nfv.Vnflcm
        operations:
          # omitted for brevity
      VnfIndicator:
        type: tosca.interfaces.nfv.MyCompanyVnfIndicator
        notifications:
          health:
            output:
              health_vnf_indicator: [ SELF, health_vnf_indicator ]
          utilization:
            output:
              utilization_vnf_indicator: [ SELF, utilization_vnf_indicator ]

```

Examples of VNF-specific node type definition illustrating the two methods to assign values to required properties:

Example with constraints and default values:

```

tosca_definitions_version: tosca_simple_yaml_1_3

description: A simple example VNF

imports:
  - etsi_nfv_sol001_vnfd_types.yaml

node_types:

  MyCompany.MultiFlavourVNF.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
    provider:
      type: string

```

```

    constraints: [ equal: MyCompany ]
    default: MyCompany
product_name:
  type: string
  constraints: [ equal: SunshineDB ]
  default: SunshineDB
software_version:
  type: string
  constraints: [ equal: '1.0' ]
  default: '1.0'
descriptor_version:
  type: string
  constraints: [ equal: '1.0' ]
  default: '1.0'
flavour_id:
  type: string
  constraints: [ valid_values: [ simple, complex ] ]
  default: simple
flavour_description:
  type: string
  default: "" #empty string
vnfm_info:
  type: list
  entry_schema:
    type: string
  constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
  default: [ '0:MyCompany-1.0.0' ]

```

Example with fixed value assignments (except for flavour_id and flavour_description) using single-line grammar for parameter definitions:

```

MyCompany.MultiFlavourVNF.1_0.1_1:
  derived_from: toasca.nodes.nfv.VNF
  properties:
    descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1178
    provider: MyCompany
    product_name: SunshineDB
    software_version: '1.0'
    descriptor_version: '1.0'
    flavour_id:
      constraints: [ valid_values: [ simple, complex ] ]
      default: simple
    flavour_description:
      default: "" #empty string
    vnfm_info: [ '0:MyCompany-1.0.0' ]

```

Example with fixed value assignments using multi-line grammar for parameter definitions:

```
MyCompany.SimpleVNF.1_0.1_1:
  derived_from: toska.nodes.nfv.VNF
  properties:
    descriptor_id:
      value: "b1bb0ce7-ebca-4fa7-95ed-4840d70a1178"
    provider:
      value: "MyCompany"
    product_name:
      value: "SimpleVNF"
    software_version:
      value: "1.0"
    descriptor_version:
      value: "1.0"
    flavour_id:
      value: simple
    flavour_description:
      default: "" #empty string
    vnfm_info:
      value: [ '0:MyCompany-1.0.0' ]
```

Example usage of metadata without properties assignment value:

```
tosca_definitions_version: toska_simple_yaml_1_3

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: toska.nodes.nfv.VNF
    properties:
      flavour_id:
        constraints:
          - valid_values: [ simple, complex ]
    modifiable_attributes:
      type: mycompany.datatypes.nfv.VnfInfoModifiableAttributes

data_types:
  mycompany.datatypes.nfv.VnfInfoModifiableAttributes:
    derived_from: toska.datatypes.nfv.VnfInfoModifiableAttributes
    properties:
    metadata:
      type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesMetadata

  mycompany.datatypes.nfv.VnfInfoModifiableAttributesMetadata:
    derived_from: toska.datatypes.nfv.VnfInfoModifiableAttributesMetadata
    properties:
      supported_configuration_protocols:
        type: string
        required: false
      supported_configuration_data_models:
        type: string
        required: false
```

In the above example, properties definitions are provided, and properties assignment values are not necessary. The properties values are available in the API.

6.8.2 `tosca.nodes.nfv.VnfExtCp`

6.8.2.1 Description

The `VnfExtCp` node type represents the `VnfExtCpd` information element as defined in ETSI GS NFV-IFA 011 [1], which describes a logical external connection point, exposed by this VNF enabling connecting with an external Virtual Link. Table 6.8.2.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.2.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfExtCp</code>
Type Qualified Name	<code>tosca.nfv.VnfExtCp</code>
Type URI	<code>tosca.nodes.nfv.VnfExtCp</code>

6.8.2.2 Properties

The properties of the `VnfExtCp` node type shall comply with the provisions set out in table 6.8.2.2-1.

Table 6.8.2.2-1: Properties

Name	Required	Type	Constraints	Description
<code>virtual_network_interface_requirements</code>	no	list of <code>tosca.datatypes.nfv.VirtualNetworkInterfaceRequirements</code>		The actual virtual NIC requirements that is been assigned when instantiating the connection point.

6.8.2.3 Attributes

None.

6.8.2.4 Requirements

The requirements of the `VnfExtCp` node type shall comply with the provisions set out in table 6.8.2.4-1.

Table 6.8.2.4-1: Requirements

Name	Required	Capability type	Constraints	Description
<code>external_virtual_link</code>	no	<code>tosca.capabilities.nfv.VirtualLinkable</code>		Specifies that CP instances require to be connected to a node that has a <code>VirtualLinkable</code> capability
<code>internal_virtual_link</code>	yes	<code>tosca.capabilities.nfv.VirtualLinkable</code>		Specifies that CP instances require to be connected to a node that has a <code>VirtualLinkable</code> capability

6.8.2.5 Capabilities

None.

6.8.2.6 Definition

The syntax of the `VnfExtCp` node type shall comply with the following definition:

```
tosca.nodes.nfv.VnfExtCp:
  derived_from: toska.nodes.nfv.Cp
  description: Describes a logical external connection point, exposed by the VNF
               enabling connection with an external Virtual Link
```

```

properties:
  virtual_network_interface_requirements:
    type: list
    description: The actual virtual NIC requirements that is been assigned when
instantiating the connection point
    required: false
    entry_schema:
      type: toasca.datatypes.nfv.VirtualNetworkInterfaceRequirements
  requirements:
    - external_virtual_link:
      capability: toasca.capabilities.nfv.VirtualLinkable
      relationship: toasca.relationships.nfv.VirtualLinksTo
      occurrences: [0, 1]
    - internal_virtual_link:
      capability: toasca.capabilities.nfv.VirtualLinkable
      relationship: toasca.relationships.nfv.VirtualLinksTo
      occurrences: [1, 1]

```

6.8.2.7 Additional Requirements

A node template of this type is used to represent a VNF external connection point only in the case the VnfExtCp is connected to an internal virtual link. The node template has the following requirements:

- internal_virtual_link requirement to allow to connect it to an internal virtual link;
- external_virtual_link requirement to allow to connect it to an external virtual link.

In the case where a VNF external connection point is re-exposing a VduCp (internal connection point) or a VipCp or a VirtualCp or a VduSubCp, the VduCp or VipCp or VirtualCp or VduSubCp node type shall be used in the service template, instead of the VnfExtCp node type.

6.8.2.8 Example

In a typical scenario, the VnfExtCp node template will be part of a service template representing a certain VNF deployment flavour. The service template substitutes for a VNF specific node type. In this substitution, the virtual_link requirement is mapped to the external_virtual_link requirement of the VnfExtCp node. This example is illustrated in clause A.3.3.

When a VNF external connection point re-exposes a Vdu connection point, the service template does not include an explicit node template of type VnfExtCp in a typical scenario where a VNF specific node type is substituted by a service template representing a certain VNF deployment flavour. In this substitution, the virtual_link requirement is mapped to the virtual_link requirement of the VduCp node. This example is illustrated in clause A.3.2.

When a VNF external connection point re-exposes a VIP connection point, the service template does not include an explicit node template of type VnfExtCp in a typical scenario where a VNF specific node type is substituted by a service template representing a certain VNF deployment flavour. In this substitution, the virtual_link requirement is mapped to the virtual_link requirement of the VipCp node. This example is illustrated in clause A.13.

When a VNF external connection point re-exposes a Subport connection point in the trunk mode, the service template does not include an explicit node template of type VnfExtCp in a typical scenario where a VNF specific node type is substituted by a service template representing a certain VNF deployment flavour. In this substitution, the virtual_link requirement is mapped to the virtual_link requirement of the VduSubCp node.

When a VNF external connection point re-exposes a Virtual connection point, the service template does not include an explicit node template of type VnfExtCp in a typical scenario where a VNF specific node type is substituted by a service template representing a certain VNF deployment flavour. In this substitution, the virtual_link requirement is mapped to the virtual_link requirement of the VirtualCp node. This example is illustrated in clause A.18.

6.8.3 toska.nodes.nfv.Vdu.Compute

6.8.3.1 Description

The Vdu.Compute node type describes the virtual compute part of a VDU (when realized as a VM) which is a construct supporting the description of the deployment and operational behaviour of a VNFC, as defined in ETSI GS NFV-IFA 011 [1].

Table 6.8.3.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.3.1-1: Type name, shorthand, and URI

Shorthand Name	Vdu.Compute
Type Qualified Name	toscanfv:Vdu.Compute
Type URI	tosca.nodes.nfv.Vdu.Compute

6.8.3.2 Properties

The properties of the Vdu.Compute node type shall comply with the provisions set out in table 6.8.3.2-1.

Table 6.8.3.2-1: Properties

Name	Required	Type	Constraints	Status	Description
name	yes	string			Human readable name of the Vdu.
description	yes	string			Human readable description of the Vdu.
boot_order	yes	boolean			It indicates whether the order of the virtual_storage requirements is used as the boot index (the first requirement represents the lowest index and defines highest boot priority). If no boot order is indicated or the value is false, the default boot order defined in the VIM or NFVI shall be used.
nfvi_constraints	no	map of string			Describes constraints on the NFVI for the VNFC instance(s) created from this Vdu. For example, aspects of a secure hosting environment for the VNFC instance that involve additional entities or processes. This property is reserved for future use in the present document.
monitoring_parameters	no	map of toska.data.types.nfv.VnfcMonitoringParameter			Describes monitoring parameters applicable to a VNFC based on this VDU.
configurable_properties	no	tosca.data.types.nfv.VnfcConfigurableProperties			Describes the configurable properties of all VNFC instances based on this VDU.

Name	Required	Type	Constraints	Status	Description
boot_data	no	tosca.data types.nfv. BootData			Contains the information used to customize a virtualised compute resource at boot time. See note. The bootData may contain variable parts that are replaced by deployment specific values before being sent to the VIM. For "volatile" parameters, i.e. those that exist only during the lifetime of an LCM operation occurrence, the parameters of each variable part shall be declared in a type derived from tosca.datatypes.nfv.VnfOperationAdditionalParameters. For "persistent" parameters, i.e. those that exist during the lifetime of the VNF instance beyond the lifetime of a single LCM operation occurrence, the parameters shall be declared in a type derived from tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions or tosca.datatypes.nfv.VnfConfigurableProperties.
vdu_profile	yes	tosca.data types.nfv. VduProfile			Defines additional instantiation data for the VDU.Compute node.
NOTE: The boot_data structure passed to a VNFC instance cannot be changed after the boot time of the VNFC instance.					

6.8.3.3 Attributes

None.

6.8.3.4 Requirements

The requirements of the Vdu.Compute node type shall comply with the provisions set out in table 6.8.3.4-1.

Table 6.8.3.4-1: Requirements

Name	Required	Capability type	Constraints	Description
virtual_storage	no	tosca.capabilities.nfv.VirtualStorage		Describes storage requirements for a virtual_storage instance attached to the virtualisation container created from virtual_compute defined for this vdu
associated_paas_service	no	tosca.capabilities.AssociablePaasService		Describes the requirements for associating to a PaaS Service.
installable_certificate	no	tosca.capabilities.nfv.InstallableCertificate		Describes the certificate to be used by this VDU. Shall be present when using delegation mode for this VDU. Otherwise it shall be absent

6.8.3.5 Capabilities

The capabilities of the Vdu.Compute node type shall comply with the provisions set out in table 6.8.3.5-1.

Table 6.8.3.5-1: Capabilities

Name	Type	Constraints	Description
virtual_compute	tosca.capabilities.nfv.VirtualCompute		Describes virtual compute resources capabilities.
virtual_binding	tosca.capabilities.nfv.VirtualBindable		Defines ability of VirtualBindable.
deployable_module_member	tosca.capabilities.nfv.DeployableModuleMember		Defines ability of being member of a deployable module.

6.8.3.6 Definition

The syntax of the Vdu.Compute node type shall comply with the following definition:

```

tosca.nodes.nfv.Vdu.Compute:
  derived_from: tosca.nodes.Root
  description: Describes the virtual compute part of a VDU which is a construct
supporting the description of the deployment and operational behavior of a VNFC
  properties:
    name:
      type: string
      description: Human readable name of the VDU
      required: true
    description:
      type: string
      description: Human readable description of the VDU
      required: true
    boot_order:
      type: boolean
      description: indicates whether the order of the virtual_storage requirements
is used as the boot index (the first requirement represents the lowest index and
defines highest boot priority)
      required: true
      default: false
    nfvi_constraints:
      type: map
      description: Describes constraints on the NFVI for the VNFC instance(s)
created from this VDU. This property is reserved for future use in the present
document.
      required: false
    entry_schema:
      type: string
    monitoring_parameters:
      type: map # key: id
      description: Describes monitoring parameters applicable to a VNFC instantiated
from this VDU
      required: false
    entry_schema:
      type: tosca.datatypes.nfv.VnfcMonitoringParameter
    configurable_properties:
      type: tosca.datatypes.nfv.VnfcConfigurableProperties
      required: false
  # derived types are expected to introduce
  # configurable_properties with its type derived from
  # tosca.datatypes.nfv.VnfcConfigurableProperties
  vdu_profile:
    type: tosca.datatypes.nfv.VduProfile
    description: Defines additional instantiation data for the VDU.Compute node
    required: true

```

```

    boot_data:
      type: tosca.datatypes.nfv.BootData
      description: Contains the information used to customize a virtualised compute
resource at boot time. The bootData may contain variable parts that are replaced by
deployment specific values before being sent to the VIM.
      required: false
    capabilities:
      virtual_compute:
        type: tosca.capabilities.nfv.VirtualCompute
        occurrences: [ 1, 1 ]
      virtual_binding:
        type: tosca.capabilities.nfv.VirtualBindable
        occurrences: [ 1, UNBOUNDED ]
      deployable_module_member:
        type: tosca.capabilities.nfv.DeployableModuleMember
        occurrences: [ 1, UNBOUNDED ]
    requirements:
      - virtual_storage:
          capability: tosca.capabilities.nfv.VirtualStorage
          relationship: tosca.relationships.nfv.AttachesTo
          occurrences: [ 0, UNBOUNDED ]
      - associated_paas_service:
          capability: tosca.capabilities.nfv.AssociablePaasService
          relationship: tosca.relationships.nfv.PaasServiceAssociates
          occurrences: [ 0, UNBOUNDED ]
      - installable_certificate:
          capability: tosca.capabilities.nfv.InstallableCertificate
          relationship: tosca.relationships.nfv.InstallsTo
          occurrences: [ 0, UNBOUNDED ]

```

6.8.3.7 Additional requirements

Node templates of type `tosca.nodes.nfv.Vdu.Compute` may contain an artifact definition of type `tosca.artifacts.nfv.SwImage`. There shall be a maximum number of one such artifact definition in a `tosca.nodes.nfv.Vdu.Compute` node template. The node template name of type `tosca.nodes.nfv.Vdu.Compute` fulfils the purpose of the "id" attribute of the `SwImageDesc` information element in ETSI GS NFV-IFA 011 [1] and hence it will be used in APIs to identify the software image id from the VNFD perspective. See example in clause 6.8.3.8.

When VNF-specific configurable properties are defined at the VDU-level, VNF providers shall define a VNF/VDU specific `Vdu.Compute` node type, where the `configurable_properties` property has a datatype derived from `tosca.datatypes.nfv.VnfcConfigurableProperties`. See example in clause 6.2.10.4.

The VNF/VDU specific `Vdu.Compute` node type shall be defined as follows:

- All properties listed in `tosca.nodes.nfv.Vdu.Compute` where the "required:" field is set to "true" shall be included.
- The capabilities and requirements of `tosca.nodes.nfv.Vdu.Compute` shall be preserved.
- The `configurable_properties` property shall have a datatype derived from `tosca.datatypes.nfv.VnfcConfigurableProperties`, according to the rules defined in clause 5.7.2 of the present document.

The definition of a VNF/VDU specific node type shall be included in one of the following yaml files:

- 4) In the yaml file which contains the corresponding VNF specific node type definition.
- 5) In low-level service templates or in the single TOSCA service template representing the VNFD in case of a single deployment flavour design with a single TOSCA service template.

- 6) In a standalone yaml file, to be imported from the low-level TOSCA service templates or from the single TOSCA service template representing the VNFD in case of a single deployment flavour design with a single TOSCA service template.
- 7) In any other VNF-specific files containing type definitions used by the VNFD TOSCA service template.

In the derived `Vdu.Compute` node type, the additional `_vnfc_configurable_properties` (VNF/VDU-specific extension of the `tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties` data type) describe the name and type of the VNFC configurable properties.

The `additional_vnfc_configurable_properties` information provided in the node type is sufficient for the client of the VNF LCM API for providing values to these properties. A value provided via the VNF LCM API to such a property overrides the value (if any) assigned in the node template or defined as default value in the node type definition.

The node template name of type `tosca.nodes.nfv.Vdu.Compute` fulfils the purpose of the 'virtualComputeDescId' attribute of the `virtualComputeDesc` information element in ETSI GS NFV-IFA 011 [1] and hence it will be used in APIs to identify the virtual compute id (`vnfdVirtualComputeDescId`).

NOTE: The use of the node template name of type `tosca.nodes.nfv.Vdu.Compute` for the 'virtualComputeDescId' attribute of the `virtualComputeDesc` information element in ETSI GS NFV-IFA 011 [1] implies in the present document version a one-to-one mapping of `virtualComputeDesc` with VDU. This deviates from the ETSI GS NFV-IFA 011 [1] modelling that defines a mapping where a `virtualComputeDesc` can be reused by one or more VDU, i.e. it implies a one-to-many mapping of `virtualComputeDesc` with VDU. This can have an impact in the determination of the number of compute flavours needed to be created with the VIM.

6.8.3.8 Example

This example illustrates boot data containing `kvp_data` by using `modifiable_attributes`.

```
tosca_definitions_version: toska_simple_yaml_1_3
..
node_types:
  mycompany.nodes.nfv.SunshineDB.1_0.1_0:
    derived_from: toska.nodes.nfv.VNF
    properties:
      ..
      modifiable_attributes:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributes
      ..
data_types:
  mycompany.datatypes.nfv.VnfInfoModifiableAttributes:
    derived_from: toska.datatypes.nfv.VnfInfoModifiableAttributes
    properties:
      extensions:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions
        required: false
  mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions:
    derived_from: toska.datatypes.nfv.VnfInfoModifiableAttributesExtensions
    properties:
      http_proxy:
        type: string
        required: true
      https_proxy:
        type: string
        required: false
      ip_address_1:
        type: string
        required: false
      vm_name:
```

```

    type: string
    required: false

topology_template:
  inputs:
    extensions:
      type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions

  substitution_mappings:
    node_type: mycompany.nodes.nfv.SunshineDB.1_0.1_0
    ..

  node_templates:
    vnf:
      type: mycompany.nodes.nfv.SunshineDB.1_0.1_0
      properties:
        ..
        modifiable_attributes:
          extensions: { get_input: extensions }

    dbBackend:
      type: toska.nodes.nfv.Vdu.Compute
      properties:
        ..
        boot_data:
          kvp_data:
            data:
              ip_address_1: { get_property: [vnf, modifiable_attributes, extensions,
ip_address_1 ] }
        ..

```

This example illustrates boot data containing kvp_data by using configurable_properties.

```

tosca_definitions_version: toska_simple_yaml_1_3
..
node_types:
  mycompany.nodes.nfv.SunshineDB.1_0.1_0:
    derived_from: toska.nodes.nfv.VNF
    properties:
      ..
      configurable_properties:
        type: mycompany.datatypes.nfv.VnfConfigurableProperties
      ..

data_types:
  mycompany.datatypes.nfv.VnfConfigurableProperties:
    derived_from: toska.datatypes.nfv.VnfConfigurableProperties
    properties:
      additional_configurable_properties:
        type: mycompany.datatypes.nfv.VnfAdditionalConfigurableProperties
        required: false

  mycompany.datatypes.nfv.VnfAdditionalConfigurableProperties:
    derived_from: toska.datatypes.nfv.VnfAdditionalConfigurableProperties
    properties:
      host_name:
        type: string
        required: false

```



```

topology_template:
  inputs:
    configurable_properties:
      type: mycompany.datatypes.nfv.VnfConfigurableProperties

  substitution_mappings:
    node_type: mycompany.nodes.nfv.SunshineDB.1_0.1_0
    ..

  node_templates:
    vnf:
      type: mycompany.nodes.nfv.SunshineDB.1_0.1_0
      properties:
        ..
        configurable_properties: { get_input: configurable_properties }

  dbBackend:
    type: toska.nodes.nfv.Vdu.Compute
    properties:
      ..
      boot_data:
        kvp_data:
          data:
            ip_address_1: { get_property: [vnf, configurable_properties,
additional_configurable_properties, host_name ] }
      ..

```

This example illustrates fetching the boot data value by using content_or_file_data.

```

tosca_definitions_version: toska_simple_yaml_1_3
..
node_types:
  mycompany.nodes.nfv.SunshineDB.1_0.1_0:
    derived_from: toska.nodes.nfv.VNF
    properties:
      ..
      modifiable_attributes:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributes
      ..

data_types:
  mycompany.datatypes.nfv.VnfInfoModifiableAttributes:
    derived_from: toska.datatypes.nfv.VnfInfoModifiableAttributes
    properties:
      extensions:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions
        required: false

  mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions:
    derived_from: toska.datatypes.nfv.VnfInfoModifiableAttributesExtensions
    properties:
      http_proxy:
        type: string
        required: true
      https_proxy:
        type: string
        required: false
      ip_address_1:
        type: string

```

```

    required: false
  vm_name:
    type: string
    required: false

topology_template:
  inputs:
    extensions:
      type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions

substitution_mappings:
  node_type: mycompany.nodes.nfv.SunshineDB.1_0.1_0
  ..

node_templates:
  vnf:
    type: mycompany.nodes.nfv.SunshineDB.1_0.1_0
    properties:
      ..
      modifiable_attributes:
        extensions: { get_input: extensions }

  dbBackend:
    type: toska.nodes.nfv.Vdu.Compute
    properties:
      ..
      boot_data:
        content_or_file_data
        contents: { concat: [ "#!/bin/bash\n",
"echo setting HTTP proxy to: ", { get_property: [vnf, modifiable_attributes,
extensions, http_proxy ] }, "\n", "..."] }

```

This example illustrates fetching the boot data value from a file by using `content_or_file_data`.

```

tosca_definitions_version: toska_simple_yaml_1_3
..
topology_template:
  ..

node_templates:
  dbBackend:
    type: toska.nodes.nfv.Vdu.Compute
    properties:
      ..
      boot_data:
        content_or_file_data
        data:
          http_proxy: { get_property: [vnf, modifiable_attributes, extensions,
http_proxy ] }
          source_path: { get_artifact : [ SELF, boot_data ] }
    artifacts:
      sw_image:
        type: toska.artifacts.nfv.SwImage
        file: images/dbBackend.v1.0.1.qcow2
      boot_data:
        type: toska.artifacts.example
        file: implementation/templates/boot_data.file
  ..

```

This example illustrates fetching the boot data value from a file by using `content_or_file_data` with `destination_path`.

```
tosca_definitions_version: tosca_simple_yaml_1_3
..
topology_template:
..

node_templates:
  dbBackend:
    type: tosca.nodes.nfv.Vdu.Compute
    properties:
      ..
      boot_data:
        content_or_file_data:
          data:
            vm_name: get_property: [vnf, modifiable_attributes, extensions, vm_name ]
            source_path: { get_artifact : [ SELF, boot_data ] }
            destination_path: /etc/
  artifacts:
    sw_image:
      type: tosca.artifacts.nfv.SwImage
      file: images/dbBackend.v1.0.1.qcow2
    boot_data:
      type: tosca.artifacts.example
      file: implementation/templates/boot_data.file
..
```

This example illustrates the association of a software image artifact to a `Vdu.Compute` node. The name of the `Vdu.Compute` node template "dbBackend" will be used in external APIs to identify the image.

```
tosca_definitions_version: tosca_simple_yaml_1_3
..
topology_template:
..

node_templates:
  dbBackend:
    type: tosca.nodes.nfv.Vdu.Compute
    properties:
      ..
      ..
  artifacts:
    sw_image:
      type: tosca.artifacts.nfv.SwImage
      file: images/dbBackend.v1.0.1.qcow2
      properties:
        ..
..
```

This example illustrates the association of a software image artifact to more than `Vdu.Compute` nodes. The name of the `Vdu.Compute` node template "dbBackend" and "oamService" will be used in external APIs to identify the image of each `Vdu.Compute` node.

```
tosca_definitions_version: tosca_simple_yaml_1_3
..
topology_template:
..

node_templates:
```

```

dbBackend:
  type: toska.nodes.nfv.Vdu.Compute
  properties:
  ..
  ..
artifacts:
  sw_image:
    type: toska.artifacts.nfv.SwImage
    file: images/dbBackend.v1.0.1.qcow2
    properties:

oamService:
  type: toska.nodes.nfv.Vdu.Compute
  properties:
  ..
  ..
artifacts:
  sw_image:
    type: toska.artifacts.nfv.SwImage
    file: images/dbBackend.v1.0.1.qcow2
    properties:

..

```

This example illustrates the association of a Certificate to a Vdu.Compute node.

```

tosca_definitions_version: toska_simple_yaml_1_3
..
topology_template:
  ..

  node_templates:
    Vdu1:
      type: toska.nodes.nfv.Vdu.Compute
      # properties: omitted for brevity
      requirements:
        installable_certificate: Vdu1_Certificate

    Vdu1_Certificate:
      type: toska.nodes.nfv.Certificate
      # properties omitted for brevity
      capabilities:
        installable_certificate

```

6.8.4 toska.nodes.nfv.Vdu.VirtualBlockStorage

6.8.4.1 Description

The VirtualBlockStorage node type describes the specifications of requirements related to virtual block storage resources, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.8.4.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.4.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualBlockStorage
Type Qualified Name	toscanfv:VirtualBlockStorage
Type URI	tosca.nodes.nfv.Vdu.VirtualBlockStorage

6.8.4.2 Properties

The properties of the VirtualBlockStorage node type shall comply with the provisions set out in table 6.8.4.2-1.

Table 6.8.4.2-1: Properties

Name	Required	Type	Constraints	Status	Description
virtual_block_storage_data	yes	tosca.datatypes.nfv.VirtualBlockStorage			Describes the block storage characteristics.
per_vnfc_instance	yes	boolean	default: true		Indicates whether the virtual storage descriptor shall be instantiated per VNFC instance.
nfvi_maintenance_info	no	tosca.datatypes.nfv.NfviMaintenanceInfo			Provides information on the rules to be observed when an instance based on this VirtualBlockStorage is impacted during NFVI operation and maintenance (e.g. NFVI resource upgrades).

6.8.4.3 Attributes

None.

6.8.4.4 Requirements

None.

6.8.4.5 Capabilities

The capabilities of the VirtualBlockStorage node type shall comply with the provisions set out in table 6.8.4.5-1.

Table 6.8.4.5-1: Capabilities

Name	Type	Constraints	Description
virtual_storage	tosca.capabilities.nfv.VirtualStorage		Defines the capabilities of virtual_storage.

6.8.4.6 Definition

The syntax of the VirtualBlockStorage node type shall comply with the following definition:

```
tosca.nodes.nfv.Vdu.VirtualBlockStorage:
  derived_from: toska.nodes.Root
  description: This node type describes the specifications of requirements related
to virtual block storage resources
  properties:
    virtual_block_storage_data:
      type: toska.datatypes.nfv.VirtualBlockStorageData
      description: Describes the block storage characteristics.
      required: true
```

```

    per_vnfc_instance:
      type: boolean
      description: Indicates whether the virtual storage descriptor shall be
instantiated per VNFC instance.
      required: true
      default: true
    nfvi_maintenance_info:
      type: toska.datatypes.nfv.NfviMaintenanceInfo
      description: Provides information on the rules to be observed when an instance
based on this VirtualBlockStorage is impacted during NFVI operation and maintenance
(e.g. NFVI resource upgrades).
      required: false
    capabilities:
      virtual_storage:
        type: toska.capabilities.nfv.VirtualStorage
        description: Defines the capabilities of virtual_storage.

```

6.8.4.7 Additional requirements

Node templates of type `tosca.nodes.nfv.Vdu.VirtualBlockStorage` may contain an artifact definition of type `tosca.artifacts.nfv.SwImage`. There shall be a maximum number of one such artifact definition in a `tosca.nodes.nfv.Vdu.VirtualBlockStorage` node template when attached to the node with type `tosca.nodes.nfv.Vdu.Compute`, otherwise, such artifact definition shall not be present. The node template name of type `tosca.nodes.nfv.Vdu.VirtualBlockStorage` fulfils the purpose of the "id" attribute of the `SwImageDesc` information element in ETSI GS NFV-IFA 011 [1] and hence it will be used in APIs to identify the software image id from the VNFD descriptor. See example in clause 6.8.3.8.

6.8.5 toska.nodes.nfv.Vdu.VirtualObjectStorage

6.8.5.1 Description

The `VirtualObjectStorage` node type describes the specifications of requirements related to virtual object storage resources, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.8.5.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.5.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VirtualObjectStorage</code>
Type Qualified Name	<code>toscanfv:VirtualObjectStorage</code>
Type URI	<code>tosca.nodes.nfv.Vdu.VirtualObjectStorage</code>

6.8.5.2 Properties

The properties of the `VirtualObjectStorage` node type shall comply with the provisions set out in table 6.8.5.2-1.

Table 6.8.5.2-1: Properties

Name	Required	Type	Constraints	Description
<code>virtual_object_storage_data</code>	yes	<code>tosca.datatypes.nfv.VirtualObjectStorageData</code>		Describes the object storage characteristics.
<code>per_vnfc_instance</code>	yes	<code>boolean</code>	default: true	Indicates whether the virtual storage descriptor shall be instantiated per VNFC instance.

Name	Required	Type	Constraints	Description
nfvi_maintenance_info	no	tosca.datatypes.nfv.NfviMaintenanceInfo		Provides information on the rules to be observed when an instance based on this VirtualObjectStorage is impacted during NFVI operation and maintenance (e.g. NFVI resource upgrades).

6.8.5.3 Attributes

None.

6.8.5.4 Requirements

None.

6.8.5.5 Capabilities

The capabilities of the VirtualObjectStorage node type shall comply with the provisions set out in table 6.8.5.5-1.

Table 6.8.5.5-1: Capabilities

Name	Type	Constraints	Description
virtual_storage	tosca.capabilities.nfv.VirtualStorage		Defines the capabilities of virtual_storage.

6.8.5.6 Definition

The syntax of the VirtualObjectStorage node type shall comply with the following definition:

```

tosca.nodes.nfv.Vdu.VirtualObjectStorage:
  derived_from: tosca.nodes.Root
  description: This node type describes the specifications of requirements
related to virtual object storage resources
  properties:
    virtual_object_storage_data:
      type: tosca.datatypes.nfv.VirtualObjectStorageData
      description: Describes the object storage characteristics.
      required: true
    per_vnfc_instance:
      type: boolean
      description: Indicates whether the virtual storage descriptor shall be
instantiated per VNFC instance.
      required: true
      default: true
    nfvi_maintenance_info:
      type: tosca.datatypes.nfv.NfviMaintenanceInfo
      description: Provides information on the rules to be observed when an
instance based on this VirtualObjectStorage is impacted during NFVI operation and
maintenance (e.g. NFVI resource upgrades).
      required: false
    capabilities:
      virtual_storage:
        type: tosca.capabilities.nfv.VirtualStorage
        description: Defines the capabilities of virtual_storage.

```

6.8.5.7 Additional requirements

None.

6.8.6 `tosca.nodes.nfv.Vdu.VirtualFileStorage`

6.8.6.1 Description

The VirtualFileStorage node type describes the specifications of requirements related to virtual file storage resources, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.8.6.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.6.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualFileStorage
Type Qualified Name	toscanfv:VirtualFileStorage
Type URI	tosca.nodes.nfv.Vdu.VirtualFileStorage

6.8.6.2 Properties

The properties of the VirtualFileStorage node type shall comply with the provisions set out in table 6.8.6.2-1.

Table 6.8.6.2-1: Properties

Name	Required	Type	Constraints	Description
virtual_file_storage_data	yes	tosca.datatypes.nfv.VirtualFileStorageData		Describes the file storage characteristics.
per_vnfc_instance	yes	boolean	default: true	Indicates whether the virtual storage descriptor shall be instantiated per VNFC instance.
nfvi_maintenance_info	no	tosca.datatypes.nfv.NfviMaintenanceInfo		Provides information on the rules to be observed when an instance based on this VirtualFileStorage is impacted during NFVI operation and maintenance (e.g. NFVI resource upgrades).

6.8.6.3 Attributes

None.

6.8.6.4 Requirements

The requirements of the VirtualFileStorage node type shall comply with the provisions set out in table 6.8.6.4-1.

Table 6.8.6.4-1: Requirements

Name	Required	Capability type	Constraints	Description
virtual_link	yes	tosca.capabilities.nfv.VirtualLinkable		Describes the requirements for linking to virtual link

6.8.6.5 Capabilities

The capabilities of the VirtualFileStorage node type shall comply with the provisions set out in table 6.8.6.5-1.

Table 6.8.6.5-1: Capabilities

Name	Type	Constraints	Description
virtual_storage	tosca.capabilities.nfv.VirtualStorage		Defines the capabilities of virtual_storage.

6.8.6.6 Definition

The syntax of the VirtualFileStorage node type shall comply with the following definition:

```

tosca.nodes.nfv.Vdu.VirtualFileStorage:
  derived_from: tosca.nodes.Root
  description: This node type describes the specifications of requirements
related to virtual file storage resources
  properties:
    virtual_file_storage_data:
      type: tosca.datatypes.nfv.VirtualFileStorageData
      description: Describes the file storage characteristics.
      required: true
    per_vnfc_instance:
      type: boolean
      description: Indicates whether the virtual storage descriptor shall be
instantiated per VNFC instance.
      required: true
      default: true
    nfvi_maintenance_info:
      type: tosca.datatypes.nfv.NfviMaintenanceInfo
      description: Provides information on the rules to be observed when an
instance based on this VirtualFileStorage is impacted during NFVI operation and
maintenance (e.g. NFVI resource upgrades).
      required: false
    capabilities:
      virtual_storage:
        type: tosca.capabilities.nfv.VirtualStorage
        description: Defines the capabilities of virtual_storage.
  requirements:
    - virtual_link:
      capability: tosca.capabilities.nfv.VirtualLinkable
      relationship: tosca.relationships.nfv.VirtualLinksTo
      occurrences: [1, 1]
      # description: Describes the requirements for linking to virtual link

```

6.8.6.7 Additional requirements

None.

6.8.7 tosca.nodes.nfv.Cp

6.8.7.1 Description

The Cp node type is defined in clause 9.8.1 of the present document.

6.8.8 toska.nodes.nfv.VduCp

6.8.8.1 Description

A VduCp node type represents the VduCpd information element as defined in ETSI GS NFV-IFA 011 [1], which describes network connectivity between a VNFC instance (based on VDU) and an internal VL. Table 6.8.8.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.8.1-1: Type name, shorthand, and URI

Shorthand Name	VduCp
Type Qualified Name	toscanfv.VduCp
Type URI	tosca.nodes.nfv.VduCp

6.8.8.2 Properties

The properties of the VduCp node type shall comply with the provisions set out in table 6.8.8.2-1.

Table 6.8.8.2-1: Properties

Name	Required	Type	Constraints	Description
bitrate_requirement	no	integer	greater_or_equal: 0	Bitrate requirement in bit per second on this connection point. See note 2.
virtual_network_interface_requirements	no	list of toska.datatypes.nfv.VirtualNetworkInterfaceRequirements		Specifies requirements on a virtual network interface realizing the CPs instantiated from this CPD. See note 2.
order	no	integer	greater_or_equal: 0	The order of the NIC on the compute instance (e.g. eth2). See note 1. compute (also known as multi vNICs) and ordering is desired, it is <i>*mandatory*</i> that all ports will be set with an order value and. The order values shall represent a positive, arithmetic progression that starts with 0 (e.g. 0, 1, 2, ..., n). If the property is not present, it shall be left to the VIM to assign the value when creating the instance. See note 2.
vnic_type	no	string	Allowed values: normal, macvtap, direct, baremetal, virtio-forwarder, direct-physical, smart-nic, bridge, ipvlan, loopback, macvlan, ptp, vlan, host-device	Describes the type of the virtual network interface realizing the CPs instantiated from this CPD. This is used to determine which mechanism driver(s) to be used to bind the port. Additional values of the attribute for VDUs realized by one or set of OS containers: • bridge • ipvlan • loopback • macvlan • ptp • vlan host-device

Name	Required	Type	Constraints	Description
NOTE 1: When binding more than one port to a single compute (also known as multi vNICs) and ordering is desired, it is mandatory that all ports will be set with an order value. The order values shall represent a positive, arithmetic progression that starts with 0 (i.e. 0, 1, 2,..., n).				
NOTE 2: For VDUs realized by one or a set of OS containers, the ability to configure virtualised resources based on this property might not be supported by all container technologies.				
NOTE 3: For CPs of VDUs realized by one or set of OS containers and used by the OS containers to connect to the primary container cluster external network, the ability to configure virtualised resources based on cpRole and trunkMode properties might not be supported by all container technologies.				

6.8.8.3 Attributes

None.

6.8.8.4 Requirements

The requirements of the VduCp node type shall comply with the provisions set out in table 6.8.8.4-1.

Table 6.8.8.4-1: Requirements

Name	Required	Capability type	Constraints	Description
virtual_binding	no	tosca.capabilities.nfv.VirtualBindable		Describe the requirement for binding with VDU
virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Describes the requirements for linking to virtual link

6.8.8.5 Capabilities

The capabilities of the VduCp node type shall comply with the provisions set out in table 6.8.8.5-1. This capability is available only the trunk_mode property value of this VduCp is "true" and there is at least one VduSubCp defined as subport of the same trunk.

Table 6.8.8.5-1: Capabilities

Name	Type	Constraints	Description
trunk_binding	tosca.capabilities.nfv.TrunkBindable		Defines ability of TrunkBindable.

6.8.8.6 Definition

The syntax of the VduCp node type shall comply with the following definition:

```
tosca.nodes.nfv.VduCp:
  derived_from: tosca.nodes.nfv.Cp
  description: describes network connectivity between a VNFC instance based on this
  VDU and an internal VL
  properties:
    bitrate_requirement:
      type: integer # in bits per second
      description: Bitrate requirement in bit per second on this connection point
      required: false
      constraints:
        - greater_or_equal: 0
    virtual_network_interface_requirements:
      type: list
      description: Specifies requirements on a virtual network interface realizing
      the CPs instantiated from this CPD
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.VirtualNetworkInterfaceRequirements
    order:
      type: integer
      description: The order of the NIC on the compute instance (e.g.eth2)
      required: false
      constraints:
        - greater_or_equal: 0
    vnic_type:
      type: string
      description: Describes the type of the virtual network interface realizing the
      CPs instantiated from this CPD
      required: false
      constraints:
        - valid_values: [ normal, macvtap, direct, baremetal, virtio-forwarder,
        direct-physical, smart-nic, bridge, ipvlan, loopback, macvlan, ptp, vlan, host-device
        ]
    capabilities:
      trunk_binding: # This capability is available only the trunk_mode property value
      of this VduCp is true and there is at least one VduSubCp defined as subport of the
      same trunk.
      type: tosca.capabilities.nfv.TrunkBindable
      occurrences: [ 0, UNBOUNDED ]
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [0, 1]
      - virtual_binding:
          capability: tosca.capabilities.nfv.VirtualBindable
          relationship: tosca.relationships.nfv.VirtualBindsTo
          occurrences: [0, 1]
```

6.8.8.7 Additional Requirements

The occurrence 0 of the virtual_binding requirement is applicable for node templates of tosca.nodes.nfv.VduSubCp node type derived from tosca.nodes.nfv.VduCp. For node templates of tosca.nodes.nfv.VduCp node type occurrence 1 applies.

6.8.9 toska.nodes.nfv.VnfVirtualLink

6.8.9.1 Description

The VnfVirtualLink node type represents the VnfVirtualLinkDesc information element as defined in ETSI GS NFV-IFA 011 [1], which describes the information about an internal VNF VL. Table 6.8.9.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.9.1-1: Type name, shorthand, and URI

Shorthand Name	VnfVirtualLink
Type Qualified Name	toscanfv:VnfVirtualLink
Type URI	tosca.nodes.nfv.VnfVirtualLink

6.8.9.2 Properties

The properties of the VnfVirtualLink node type shall comply with the provisions set out in table 6.8.9.2-1.

Table 6.8.9.2-1: Properties

Name	Required	Type	Constraints	Description
connectivity_type	yes	ConnectivityType		Specifies the protocol exposed by the VL and the flow pattern supported by the VL.
description	no	string		Provides human-readable information on the purpose of the VL (e.g. control plane traffic).
test_access	no	list of string	Valid values: See YAML definition constraints	Test access facilities available on the VL.
vl_profile	yes	tosca.datatypes.nfv.VIProfile		Defines additional data for the VL: maximum and minimum bit rate requirements and QoS.
monitoring_parameters	no	map of toska.datatypes.nfv.VirtualLinkMonitoringParameter		Describe monitoring parameters applicable to a VL instantiated from this node type. See note.
nfvi_maintenance_info	no	tosca.datatypes.nfv.NfviMaintenanceInfo		Provides information on the rules to be observed when an instance based on this VnfVirtualLink is impacted during NFVI operation and maintenance (e.g. NFVI resource upgrades).
externally_managed	no	string	Valid values: See YAML definition constraints	Specifies the intent of the VNF designer with reference to the internal VL instances created from this descriptor being externally managed, i.e. whether it is "allowed" or "required" that these are externally managed. If this property is absent, the value "allowed" is assumed. If the VNFD does not reference any LCM script and if the "vnfm_info" property in the VNF-specific node type derived from the toska.nodes.nfv.VNF node type indicates that the VNF can be managed by any ETSI NFV compliant VNFM, this property shall not be present.

Name	Required	Type	Constraints	Description
NOTE: This property shall not be present in a VNFD service template when all the virtualisation containers of the VNF are realized as OsContainers.				

6.8.9.3 Requirements

None.

6.8.9.4 Capabilities

The capabilities of the VnfVirtualLink node type shall comply with the provisions set out in table 6.8.9.4-1.

Table 6.8.9.4-1: Capabilities

Name	Type	Constraints	Description
virtual_linkable	tosca.capabilities.nfv.VirtualLinkable		Defines ability of VirtualLinkable.

6.8.9.5 Definition

The syntax of the VnfVirtualLink node type shall comply with the following definition:

```

tosca.nodes.nfv.VnfVirtualLink:
  derived_from: tosca.nodes.Root
  description: Describes the information about an internal VNF VL
  properties:
    connectivity_type:
      type: tosca.datatypes.nfv.ConnectivityType
      description: Specifies the protocol exposed by the VL and the flow pattern
supported by the VL
      required: true
    description:
      type: string
      description: Provides human-readable information on the purpose of the VL
      required: false
    test_access:
      type: list
      description: Test access facilities available on the VL
      required: false
    entry_schema:
      type: string
      constraints:
        - valid_values: [ passive_monitoring, active_loopback ]
    vl_profile:
      type: tosca.datatypes.nfv.VlProfile
      description: Defines additional data for the VL
      required: true
    monitoring_parameters:
      type: map #key: id
      entry_schema:
        type: tosca.datatypes.nfv.VirtualLinkMonitoringParameter
        description: Describes monitoring parameters applicable to the VL
        required: false
    nfvi_maintenance_info:
      type: tosca.datatypes.nfv.NfviMaintenanceInfo
      description: Provides information on the rules to be observed when an instance
based on this VnfVirtualLink is impacted during NFVI operation and maintenance (e.g.
NFVI resource upgrades).
      required: false

```

```

externally_managed:
  type: string
  description: Specifies the intent of the VNF designer w.r.t. the external
management of the internal VL instances created from this descriptor, i.e. whether it
is "allowed" or "required" that these are externally managed. If this property is
absent, the value "allowed" is assumed. If the VNFD does not reference any LCM script
and if the "vnfm_info" property in the VNF-specific node type derived from the
tosca.nodes.nfv.VNF node type indicates that the VNF can be managed by any ETSI NFV
compliant VNFM, this property shall not be present.
  required: false
  constraints:
    - valid_values: [ allowed, required ]
capabilities:
  virtual_linkable:
    type: toska.capabilities.nfv.VirtualLinkable

```

6.8.10 toska.nodes.nfv.VipCp

6.8.10.1 Description

A VipCp node type represents the VipCpd information element as defined in ETSI GS NFV-IFA 011 [1], which describes a connection point to allocate one or a set of virtual IP addresses. Table 6.8.10.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.10.1-1: Type name, shorthand, and URI

Shorthand Name	VipCp
Type Qualified Name	toscanfv:VipCp
Type URI	tosca.nodes.nfv.VipCp

6.8.10.2 Properties

The properties of the VipCp node type shall comply with the provisions set out in table 6.8.10.2-1.

Table 6.8.10.2-1: Properties

Name	Required	Type	Constraints	Description
dedicated_ip_addresses	yes	boolean		Indicates whether the VIP address shall be different from the addresses allocated to all associated VduCp instances or shall be the same as one of them. If set to true, the VIP address shall be different from the addresses allocated to all of the VduCp instances associated to it. If set to false, the VIP address shall be the same as one of the VduCp instances associated to it.
vip_function	yes	string	valid values: See YAML definition constraints	Indicates the function the virtual IP address is used for: high availability or load balancing. See note.
NOTE: When used for high availability, only one of the internal VDU CP instances or VNF external CP instances that share the virtual IP is bound to the VIP address at a time, i.e. only one is configured in the external (to the VNF) router to receive the packets e.g. as a result of a G-ARP message previously sent by this instance. When used for load balancing purposes all CP instances that share the virtual IP are bound to it. A load balancing function sends the packet to one or the other, but not to both.				

6.8.10.3 Attributes

None.

6.8.10.4 Requirements

The requirements of the VipCp node type shall comply with the provisions set out in table 6.8.10.4-1.

Table 6.8.10.4-1: Requirements

Name	Required	Type	Constraints	Description
target	yes	tosca.capabilities.Node		Describes the requirement for connecting to VDU CP instances that share the virtual IP address.
virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Describes the requirements for linking to virtual link.

6.8.10.5 Definition

The syntax of the VipCp node type shall comply with the following definition:

```

tosca.nodes.nfv.VipCp:
  derived_from: tosca.nodes.nfv.Cp
  description: Describes a connection point to allocate one or a set of virtual IP
addresses
  properties:
    dedicated_ip_address:
      type: boolean
      description: Indicates whether the VIP address shall be different from the
addresses allocated to all associated VduCp instances or shall be the same as one of
them.
      required: true
      default: true
    vip_function:
      type: string
      description: "Indicates the function the virtual IP address is used for: high
availability or load balancing. When used for high availability, only one of the
internal VDU CP instances or VNF external CP instances that share the virtual IP is
bound to the VIP address at a time. When used for load balancing purposes all CP
instances that share the virtual IP are bound to it."
      required: true
      constraints:
        - valid_values: [ high_availability, load_balance ]
  requirements:
    - target:
        capability: tosca.capabilities.Node
        node: tosca.nodes.nfv.VduCp
        relationship: tosca.relationships.DependsOn
        occurrences: [ 1, UNBOUNDED ]
    - virtual_link:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VipVirtualLinksTo
        occurrences: [0, 1]

```

6.8.10.6 Example

See clause A.13.

6.8.11 toska.nodes.nfv.VduSubCp

6.8.11.1 Description

A VduSubCp node type represents the Subport information element as defined in ETSI GS NFV-IFA 011 [1], which describes network connectivity between a VNFC instance (based on VDU) and an internal VL through a trunk port. Table 6.8.11.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.11.1-1: Type name, shorthand, and URI

Shorthand Name	VduSubCp
Type Qualified Name	toscanfv.VduSubCp
Type URI	tosca.nodes.nfv.VduSubCp

6.8.11.2 Properties

The properties of the VduSubCp node type shall comply with the provisions set out in table 6.8.11.2-1.

Table 6.8.11.2-1: Properties

Name	Required	Type	Constraints	Description
segmentation_type	no	string	Valid values: See YAML definition constraints	Specifies the encapsulation type for the traffics coming in and out of the trunk subport. VALUES: - vlan: the subport uses VLAN as encapsulation type. - vxlan: the subport uses VXLAN as encapsulation type. - nvgre: the subport uses NVGRE as encapsulation type. - inherit: the subport gets its segmentation type from the network it is connected to. Default value is vlan.
segmentation_id	no	integer	greater_or_equal: 0	Specifies the segmentation ID for the subport, which is used to differentiate the traffics on different networks coming in and out of the trunk port. If a value is provided here it may be overridden by a value provided at run time when the infrastructure does not support mapping of segmentation IDs.
is_automatic_discovery	no	boolean	default: false	Indicates if additional parameters are exchanged through the use of a routing protocol. See note.
NOTE: In the case of VXLAN as encapsulation type, when using the VduSubCp node to describe a Network Virtualization Edge (NVE), this may exchange additional information with other NVEs through the use of routing protocols such as iBGP. If segmentation_type value is not set to vxlan, is_automatic_discovery shall not be present.				

6.8.11.3 Attributes

None.

6.8.11.4 Requirements

The requirements of the VduSubCp node type shall comply with the provisions set out in table 6.8.11.4-1.

Table 6.8.11.4-1: Requirements

Name	Required	Capability type	Constraints	Description
trunk_binding	yes	tosca.capabilities.nfv.TrunkBindable		Describes the requirements for binding with trunk parent port.

6.8.11.5 Definition

The syntax of the VduSubCp node type shall comply with the following definition:

```

tosca.nodes.nfv.VduSubCp:
  derived_from: tosca.nodes.nfv.VduCp
  description: describes network connectivity between a VNFC instance based on this
VDU and an internal VL through a trunk port
  properties:
    segmentation_type:
      type: string
      description: Specifies the encapsulation type for the traffics coming in and
out of the trunk subport.
      required: false
      constraints:
        - valid_values: [ vlan, inherit, vxlan, nvgre ]
    segmentation_id:
      type: integer
      description: Specifies the segmentation ID for the subport, which is used to
differentiate the traffics on different networks coming in and out of the trunk port.
      required: false
      constraints:
        - greater_or_equal: 0
    is_automatic_discovery:
      type: boolean
      description: Indicates if additional parameters are exchanged through the use
of a routing protocol.
      required: false
      default: false
    requirements:
      - trunk_binding:
          capability: tosca.capabilities.nfv.TrunkBindable
          relationship: tosca.relationships.nfv.TrunkBindsTo
          node: tosca.nodes.nfv.VduCp
          occurrences: [1, 1]

```

6.8.11.6 Example

See clause A.16.

6.8.11.7 Additional Requirements

The trunk_mode property of the VduSubCp node shall be set as false.

6.8.12 toska.nodes.nfv.Vdu.OsContainer

6.8.12.1 Description

The Vdu.OsContainer node type represents the OsContainerDesc information element as defined in ETSI GS NFV-IFA 011 [1]. Table 6.8.12.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.12.1-1: Type name, shorthand, and URI

Shorthand Name	Vdu.OsContainer
Type Qualified Name	toscanfv.Vdu.OsContainer
Type URI	tosca.nodes.nfv.Vdu.OsContainer

6.8.12.2 Properties

The properties of the Vdu.OsContainer node type shall comply with the provisions set out in table 6.8.12.2-1.

Table 6.8.12.2-1: Properties

Name	Required	Type	Constraints	Status	Description
name	yes	string			Human readable name of the OS Container.
description	yes	string			Human readable description of the OS Container.
requested_cpu_resources	no	integer	greater_than: 0		Number of CPU resources requested for the OS container in milli-CPU.
requested_cpu_resources_valid_values	no	tosca.datatypes.nfv.IntegerRange			Indicates valid values for the number of CPU resources requested for the container in milli-CPU. If this property is present the number of CPU resources requested for the container can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the set of co-located container compute resources is instantiated with the value indicated in the requested_cpu_resources property. If this property is not present the number of CPU resources requested for the container is not configurable via the VNF LCM interface and it is set to the value indicated in the requested_cpu_resources property, if this property is present. This property shall not be present if requested_cpu_resources is not present.
cpu_resource_limit	no	integer	greater_than: 0		Number of CPU resources the OS container can maximally use in milli-CPU.

Name	Required	Type	Constraints	Status	Description
cpu_resource_limit_valid_values	no	tosca.datatypes.nfv.IntegerRange			Indicates valid values for the number of CPU resources the container can maximally use in milli-CPU. If this property is present the number of CPU resources the container can maximally use can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the set of co-located container compute resources is instantiated allowing the container to maximally use the value indicated in the cpu_resource_limit property. If this property is not present the number of CPU resources the container can maximally use is not configurable via the VNF LCM interface and it is set to the value indicated in the cpu_resource_limit property, if this property is present. This property shall not be present if cpu_resource_limit is not present.
requested_memory_resources	no	scalar-unit.size	greater_than: 0		Amount of memory resources requested for the OS container (e.g. in MB).
requested_memory_resources_valid_values	no	tosca.datatypes.nfv.SizeRange			Indicates valid values for the amount of memory resources requested for the container (e.g. in MB). If this property is present the amount of memory resources requested for the container can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the set of co-located container compute resources is instantiated with the value indicated in the requested_memory_resources property. If this property is not present the amount of memory resources requested for the container is not configurable via the VNF LCM interface and it is set to the value indicated in the requested_memory_resources property, if this property is present. This property shall not be present if requested_memory_resources is not present.
memory_resource_limit	no	scalar-unit.size	greater_than: 0		Amount of memory resources the OS container can maximum use (e.g. in MB).

Name	Required	Type	Constraints	Status	Description
memory_resource_limit_val id_values	no	tosca.dataty pes.nfv.Size Range			Indicates valid values for the amount of memory resources the container can maximally use (e.g. in MB). If this property is present the amount of memory resources the container can maximally use can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the set of co-located container compute resources is instantiated allowing the container to maximally use the value indicated in the memory_resource_limit property. If this property is not present the amount of memory resources the container can maximally use is not configurable via the VNF LCM interface and it is set to the value indicated in the memory_resource_limit property, if this property is present. This property shall not be present if memory_resource_limit is not present.
requested_ephemeral_stor age_resources	no	scalar- unit.size	greater_than: 0		Size of ephemeral storage resources requested for the OS container (e.g. in GB).
requested_ephemeral_stor age_resources_valid_value s	no	tosca.dataty pes.nfv.Size Range			Indicates valid values for the size of ephemeral storage resources requested for the container (e.g. in GB). If this property is present the amount of ephemeral storage resources requested for the container can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the set of co-located container compute resources is instantiated with the value indicated in the requested_ephemeral_storage_reso urces property. If this property is not present the amount of ephemeral storage resources requested for the container is not configurable via the VNF LCM interface and it is set to the value indicated in the requested_ephemeral_storage_reso urces property, if this property is present. This property shall not be present if requested_ephemeral_storage_reso urces is not present.
ephemeral_storage_resour ce_limit	no	scalar- unit.size	greater_than: 0		Size of ephemeral storage resources the OS container can maximum use (e.g. in GB).

Name	Required	Type	Constraints	Status	Description
ephemeral_storage_resource_limit_valid_values	no	tosca.datatypes.nfv.SizeRange			Indicates valid values for the size of ephemeral storage resources the container can maximally use (e.g. in GB). If this property is present the size of ephemeral storage resources the container can maximally use can be indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation, the set of co-located container compute resources is instantiated allowing the container to maximally use the value indicated in the ephemeral_storage_resource_limit property. If this attribute is not present the amount of ephemeral storage resources the container can maximally use is not configurable via the VNF LCM interface and it is set to the value indicated in the ephemeral_storage_resource_limit, if this property is present. This property shall not be present if ephemeral_storage_resource_limit is not present.
extended_resource_requests	no	list of toscadataypes.nfv.ExtendedResourceData			Extended resources and their respective amount required by the container. It may also optionally indicate a range of valid values of the amount required if this can be configurable via the VNF LCM interface.
huge_pages_resources	no	list of toscadataypes.nfv.Hugepages			The requirement for huge pages resources. Each element in the list indicates a hugepage size and the total memory requested for hugepages of that size. It may also optionally indicate a range of valid values for the total memory requested for hugepages of that size if the amount required can be configurable via the VNF LCM interface.
cpu_pinning_requirements	no	tosca.datatypes.nfv.VirtualCpuPinning			Requirements for CPU pinning configuration.
cpu_power_state_requirements	no	tosca.datatypes.nfv.CpuPowerState			Requirements for CPU power state configuration for the OS container.

6.8.12.3 Attributes

None.

6.8.12.4 Capabilities

The capabilities of the Vdu.OsContainer node type shall comply with the provisions set out in table 6.8.12.4-1.

Table 6.8.12.4-1: Capabilities

Name	Type	Constraints	Description
container_deployable	tosca.capabilities.nfv.ContainerDeployable		Defines ability of ContainerDeployable.

6.8.12.5 Definition

The syntax of the Vdu.OsContainer node type shall comply with the following definition:

```

tosca.nodes.nfv.Vdu.OsContainer:
  derived_from: tosca.nodes.Root
  description: Describes the resources of a single OS container within a VDU
  properties:
    name:
      type: string
      description: Human readable name of the OS container
      required: true
    description:
      type: string
      description: Human readable description of the OS container
      required: true
    requested_cpu_resources:
      type: integer
      description: Number of CPU resources requested for the OS container in milli-
CPU.
      required: false
      constraints:
        - greater_or_equal: 0
      requested_cpu_resources_valid_values:
        type: tosca.datatypes.nfv.IntegerRange
        description: Indicates valid values for the number of CPU resources requested
for the container in milli-CPU. If this property is present the number of CPU
resources requested for the container can be indicated in a VNF LCM operation. If no
value is indicated in the VNF LCM operation, the set of co-located container compute
resources is instantiated with the value indicated in the requested_cpu_resources
property. If this property is not present the number of CPU resources requested for
the container is not configurable via the VNF LCM interface and it is set to the value
indicated in the requested_cpu_resources property, if this property is present.
      required: false
    cpu_resource_limit:
      type: integer
      description: Number of CPU resources the OS container can maximally use in
milli-CPU.
      required: false
      constraints:
        - greater_or_equal: 0
      cpu_resource_limit_valid_values:
        type: tosca.datatypes.nfv.IntegerRange
        description: Indicates valid values for the number of CPU resources the
container can maximally use in milli-CPU. If this property is present the number of
CPU resources the container can maximally use can be indicated in a VNF LCM operation.
If no value is indicated in the VNF LCM operation, the set of co-located container
compute resources is instantiated allowing the container to maximally use the value
indicated in the cpu_resource_limit property. If this property is not present the
number of CPU resources the container can maximally use is not configurable via the
VNF LCM interface and it is set to the value indicated in the cpu_resource_limit
property, if this property is present.
      required: false
    requested_memory_resources:

```

```

    type: scalar-unit.size
    description: Amount of memory resources requested for the OS container (e.g.
in MB).
    required: false
    requested_memory_resources_valid_values:
      type: toska.datatypes.nfv.SizeRange
      description: Indicates valid values for the amount of memory resources
requested for the container. If this property is present the amount of memory
resources requested for the container can be indicated in a VNF LCM operation. If no
value is indicated in the VNF LCM operation, the set of co-located container compute
resources is instantiated with the value indicated in the requested_memory_resources
property. If this property is not present the amount of memory resources requested for
the container is not configurable via the VNF LCM interface and it is set to the value
indicated in the requested_memory_resources property, if this property is present.
    required: false
    memory_resource_limit:
      type: scalar-unit.size
      description: Amount of memory resources the OS container can maximum use (e.g.
in MB).
    required: false
    memory_resource_limit_valid_values:
      type: toska.datatypes.nfv.SizeRange
      description: Indicates valid values for the amount of memory resources the
container can maximally use (e.g. in MB). If this property is present the amount of
memory resources the container can maximally use can be indicated in a VNF LCM
operation. If no value is indicated in the VNF LCM operation, the set of co-located
container compute resources is instantiated allowing the container to maximally use
the value indicated in the memory_resource_limit property. If this property is not
present the amount of memory resources the container can maximally use is not
configurable via the VNF LCM interface and it is set to the value indicated in the
memory_resource_limit property, if this property is present.
    required: false
    requested_ephemeral_storage_resources:
      type: scalar-unit.size
      description: Size of ephemeral storage resources requested for the OS
container (e.g. in GB).
    required: false
    requested_ephemeral_storage_resources_valid_values:
      type: toska.datatypes.nfv.SizeRange
      description: Indicates valid values for the size of ephemeral storage
resources requested for the container (e.g. in GB). If this property is present the
amount of ephemeral storage resources requested for the container can be indicated in
a VNF LCM operation. If no value is indicated in the VNF LCM operation, the set of co-
located container compute resources is instantiated with the value indicated in the
requested_ephemeral_storage_resources property. If this property is not present the
amount of ephemeral storage resources requested for the container is not configurable
via the VNF LCM interface and it is set to the value indicated in the
requested_ephemeral_storage_resources property, if this property is present.
    required: false
    ephemeral_storage_resource_limit:
      type: scalar-unit.size
      description: Size of ephemeral storage resources the OS container can maximum
use (e.g. in GB).
    required: false
    ephemeral_storage_resource_limit_valid_values:
      type: toska.datatypes.nfv.SizeRange
      description: Indicates valid values for the size of ephemeral storage
resources the container can maximally use (e.g. in GB). If this property is present
the size of ephemeral storage resources the container can maximally use can be
indicated in a VNF LCM operation. If no value is indicated in the VNF LCM operation,
the set of co-located container compute resources is instantiated allowing the

```


container to maximally use the value indicated in the `ephemeral_storage_resource_limit` property. If this attribute is not present the amount of ephemeral storage resources the container can maximally use is not configurable via the VNF LCM interface and it set to the value indicated in the `ephemeral_storage_resource_limit`, if this property is present.

```

    required: false
    extended_resource_requests:
      type: list
      description: Extended resources and their respective amount required by the
        container. It may also optionally indicate a range of valid values of the amount
        required if this can be configurable via the VNF LCM interface.
      required: false
      entry_schema:
        type: tosa.datatypes.nfv.ExtendedResourceData
      constraints:
        - min_length: 1
    huge_pages_resources:
      type: list
      description: The requirement for huge pages resources. Each element in the
        list indicates a hugepage size and the total memory requested for hugepages of that
        size. It may also optionally indicate a range of valid values for the total memory
        requested for hugepages of that size if the amount required can be configurable via
        the VNF LCM interface.
      required: false
      entry_schema:
        type: tosa.datatypes.nfv.Hugepages
    cpu_pinning_requirements:
      type: tosa.datatypes.nfv.VirtualCpuPinning
      description: Requirements for CPU pinning configuration.
      required: false
    cpu_power_state_requirements:
      type: tosa.datatypes.nfv.CpuPowerState
      description: Requirements for CPU power state configuration for the OS
        container.
      required: false
    capabilities:
      container_deployable:
        type: tosa.capabilities.nfv.ContainerDeployable
        occurrences: [ 1, UNBOUNDED ]

```

6.8.12.6 Additional requirements

Node templates of type `tosca.nodes.nfv.Vdu.OsContainer` shall contain an artifact definition of type `tosca.artifacts.nfv.SwImage`. There shall be a maximum number of one such artifact definition in a `tosca.nodes.nfv.Vdu.OsContainer` node template. The node template name of type `tosca.nodes.nfv.Vdu.OsContainer` fulfils the purpose of the "id" attribute of the `SwImageDesc` information element in ETSI GS NFV-IFA 011 [1] and hence it will be used in APIs to identify the software image id from the VNFD perspective.

6.8.13 `tosca.nodes.nfv.Vdu.OsContainerDeployableUnit`

6.8.13.1 Description

The `Vdu.OsContainerDeployableUnit` node type describes the aggregate of OS containers of a VDU (when realized as OS containers) which is a construct supporting the description of the deployment and operational behaviour of a VNFC.

Table 6.8.13.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.13.1-1: Type name, shorthand, and URI

Shorthand Name	Vdu.OsContainerDeployableUnit
Type Qualified Name	toscanfv:Vdu.OsContainerDeployableUnit
Type URI	tosca.nodes.nfv.Vdu.OsContainerDeployableUnit

6.8.13.2 Properties

The properties of the Vdu.OsContainerDeployableUnit node type shall comply with the provisions set out in table 6.8.13.2-1.

Table 6.8.13.2-1: Properties

Name	Required	Type	Constraints	Status	Description
name	yes	string			Human readable name of the Vdu.
description	yes	string			Human readable description of the Vdu.
logical_node	no	Map of tosca.datatypes.nfv.LogicalNodeData			Describes the Logical Node requirements.
requested_additional_capabilities	no	Map of tosca.datatypes.nfv.RequestedAdditionalCapability			Describes additional capability for a particular OS container.
nfvi_constraints	no	map of string			Describes constraints on the NFVI for the VNFC instance(s) created from this Vdu. For example, aspects of a secure hosting environment for the VNFC instance that involve additional entities or processes. This property is reserved for future use in the present document.

Name	Required	Type	Constraints	Status	Description
configurable_properties	no	map of <code>tosca.datatypes.nfv.VnfcConfigurableProperties</code>			Describes the configurable properties of all VNFC instances based on this VDU.
vdu_profile	yes	<code>tosca.datatypes.nfv.VduProfile</code>			Defines additional instantiation data for the <code>VDU.OsContainerDeployableUnit</code> node.
mcio_constraint_params	no	list of string	Valid values: See YAML definition constraints		Defines the parameter names for constraints expected to be assigned to MCIOs realizing this <code>Vdu.OsContainerDeployableUnit</code> . The value specifies the standardized semantical context of the MCIO constraints.
mcio_identification_data	yes	<code>tosca.datatypes.nfv.MciIdentificationData</code>			Name and type of the MCIO that realizes this <code>Vdu.OsContainerDeployableUnit</code> . It allows the VNFM to identify the MCIO e.g. when querying the CISM.
is_num_of_instances_cluster_based	yes	boolean	default: false		Indicates whether the <code>VDU.OsContainerDeployableUnit</code> is a template for a VNFC that is instantiated a number of times based on the instantiation level or scale level (FALSE) or it is a template describing a workload that is instantiated in every CIS-node, or in every CIS-node that fulfills certain characteristics (TRUE). See notes 1 and 2.
NOTE 1: An example of a workload that is deployed per CIS-node, or per CIS-node that fulfills certain characteristics, when the container infrastructure service is a Kubernetes® instance, is a DaemonSet. NOTE 2: If the workload is to be deployed on CIS-nodes that fulfil certain characteristics, these are indicated with the <code>mcio_constraint_params</code> property or with other placement related properties (such as <code>requested_additional_capabilities</code>, <code>extended_resource_requests</code>, etc.). In the case of <code>mcio_constraint_params</code>, only capability related values and the <code>node_pool</code> value are valid, affinity or anti-affinity related values are not applicable.					

6.8.13.3 Attributes

None.

6.8.13.4 Requirements

The requirements of the `Vdu.OsContainerDeployableUnit` node type shall comply with the provisions set out in table 6.8.13.4-1.

Table 6.8.13.4-1: Requirements

Name	Required	Capability type	Constraints	Description
virtual_storage	no	tosca.capabilities.nfv.VirtualStorage		Describes storage requirements for a virtual_storage instance attached to the VDU released by the Vdu.OSContainerDeployableUnit
container	no	tosca.capabilities.nfv.ContainerDeployable		Describes the OsContainer requirements that encompass the Vdu.OSContainerDeployableUnit
associated_paas_service	no	tosca.capabilities.AssociablePaasService		Describes the requirements for associating to a PaaS Service.
installable_certificate	no	tosca.capabilities.nfv.InstallableCertificate		Describes the certificate to be used by this VDU. Shall be present when using delegation mode for this VDU. Otherwise it shall be absent.

6.8.13.5 Capabilities

The capabilities of the Vdu.OSContainerDeployableUnit node type shall comply with the provisions set out in table 6.8.13.5-1.

Table 6.8.13.5-1: Capabilities

Name	Type	Constraints	Description
virtual_binding	tosca.capabilities.nfv.VirtualBindable		Defines ability of VirtualBindable.
associable	tosca.capabilities.nfv.AssociableVdu		Defines ability of AssociableVDU.

6.8.13.6 Definition

The syntax of the Vdu.OSContainerDeployableUnit node type shall comply with the following definition:

```

tosca.nodes.nfv.Vdu.OSContainerDeployableUnit:
  derived_from: tosca.nodes.Root
  description: Describes the aggregate of container of a VDU (when realized as OS
containers) which supporting the description of the deployment and operational
behavior of a VNFC
  properties:
    name:
      type: string
      description: Human readable name of the VDU
      required: true
    description:
      type: string
      description: Human readable description of the VDU
      required: true
    logical_node:
      type: map
      description: Describes the Logical Node requirements
      required: false
    entry_schema:
      type: tosca.datatypes.nfv.LogicalNodeData
    requested_additional_capabilities:
      type: map
      description: Describes additional capability for a particular OS container
      required: false
    entry_schema:

```

```

    type: tosca.datatypes.nfv.RequestedAdditionalCapability
  nfvi_constraints:
    type: map
    description: Describes constraints on the NFVI for the VNFC instance(s)
created from this VDU. This property is reserved for future use in the present
document.
    required: false
    entry_schema:
      type: string
  configurable_properties:
    type: tosca.datatypes.nfv.VnfcConfigurableProperties
    required: false
#   derived types are expected to introduce
#   configurable_properties with its type derived from
#   tosca.datatypes.nfv.VnfcConfigurableProperties
  vdu_profile:
    type: tosca.datatypes.nfv.VduProfile
    description: Defines additional instantiation data for the
Vdu.OsContainerDeployableUnit node
    required: true
  mcio_constraint_params:
    type: list
    entry_schema:
      type: string
      constraints:
        - valid_values:
            - affinity_nfvi_pop
            - affinity_zone
            - affinity_zone_group
            - affinity_nfvi_node
            - affinity_cis_node
            - anti_affinity_nfvi_pop
            - anti_affinity_zone
            - anti_affinity_zone_group
            - anti_affinity_nfvi_node
            - anti_affinity_cis_node
            - local_affinity_nfvi_pop
            - local_affinity_zone
            - local_affinity_zoneGroup
            - local_affinity_nfvi_node
            - local_affinity_cis_node
            - local_anti_affinity_nfvi_pop
            - local_anti_affinity_zone
            - local_anti_affinity_zone_group
            - local_anti_affinity_nfvi_node
            - local_anti_affinity_cis_node
            - node_additional_capability_ssd
            - node_additional_capability_dpdk
            - node_additional_capability_sriov
            - node_additional_capability_gpu
            - node_additional_capability_fpga
            - node_additional_capability_cpu_pin
            - node_capability_logical_numa
            - node_pool
        description: Defines the parameter names for constraints expected to be
assigned to MCIOs realizing this Vdu.OsContainerDeployableUnit. The value specifies
the standardized semantical context of the MCIO constraints.
    required: false
  mcio_identification_data:
    type: tosca.datatypes.nfv.McioIdentificationData

```

```

    description: Name and type of the MCIO that realizes this
Vdu.OsContainerDeployableUnit. It allows the VNFM to identify the MCIO e.g. when
querying the CISM.
    is_num_of_instances_cluster_based:
      type: boolean
      description: Indicates whether the VDU.OsContainerDeployableUnit is a template
for a VNFC that is instantiated a number of times based on the instantiation level or
scale level (FALSE) or it is a template describing a workload that is instantiated in
every CIS-node, or in every CIS-node that fulfills certain characteristics (TRUE).
      required: true
      default: false
    capabilities:
      virtual_binding:
        type: toska.capabilities.nfv.VirtualBindable
        occurrences: [ 0, UNBOUNDED ]
      associable:
        type: toska.capabilities.nfv.AssociableVdu
        occurrences: [ 1, 1 ]
    requirements:
      - virtual_storage:
          capability: toska.capabilities.nfv.VirtualStorage
          relationship: toska.relationships.nfv.AttachesTo
          occurrences: [ 0, UNBOUNDED ]
      - container:
          capability: toska.capabilities.nfv.ContainerDeployable
          relationship: toska.relationships.nfv.DeploysTo
          occurrences: [ 0, UNBOUNDED ]
      - associated_paas_service:
          capability: toska.capabilities.nfv.AssociablePaasService
          relationship: toska.relationships.nfv.PaasServiceAssociates
          occurrences: [ 0, UNBOUNDED ]
      - installable_certificate:
          capability: toska.capabilities.nfv.InstallableCertificate
          relationship: toska.relationships.nfv.InstallsTo
          occurrences: [ 0, UNBOUNDED ]

```

6.8.13.7 Additional Requirements

In case a node template of type `tosca.nodes.nfv.Vdu.OsContainerDeployableUnit` is present in a VNFD service template, while no node template of type `tosca.nodes.nfv.Vdu.OsContainer` is present, at least one node template of type `tosca.nodes.nfv.Mciop` shall be present in the VNFD service template.

6.8.14 toska.nodes.nfv.Mciop

6.8.14.1 Description

The `Mciop` node type does not correspond to an information element defined in ETSI GS NFV-IFA 011 [1]. It is a representation of the object described by the `mciop` artifact, capable of being profiled by the properties of the `MciopProfile` information element defined in ETSI GS NFV-IFA 011 [1]. Table 6.8.14.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.14.1-1: Type name, shorthand, and URI

Shorthand Name	<code>Mciop</code>
Type Qualified Name	<code>toscanfv:Mciop</code>
Type URI	<code>tosca.nodes.nfv.Mciop</code>

6.8.14.2 Properties

None.

6.8.14.3 Attributes

None.

6.8.14.4 Requirements

The requirements of the Mciop node type shall comply with the provisions set out in table 6.8.14.4-1.

Table 6.8.14.4-1: Requirements

Name	Required	Capability type	Constraints	Description
associatedVdu	yes	tosca.capabilities.nfv.AssociableVdu		Specifies Vdu.OsContainerDeployableUnit nodes that are associated to the Mciop. If the associated VDUs belong to a deployable module, all associated VDUs shall belong to that deployable module. See note.
NOTE: This means that one MCIOP shall not combine descriptions of VDUs that do not belong to the same deployable module nor it shall combine descriptions of VDUs that belong to a deployable module and VDUs that do not belong to any deployable module.				

6.8.14.5 Capabilities

The capabilities of the Mciop node type shall comply with the provisions set out in table 6.8.14.5-1.

Table 6.8.14.5-1: Capabilities

Name	Type	Constraints	Description
deployable_module_member	tosca.capabilities.nfv.DeployableModuleMember		Defines ability of being member of a deployable module.

6.8.14.6 Definition

The syntax of the Mciop node type shall comply with the following definition:

```
tosca.nodes.nfv.Mciop:
  derived_from: tosca.nodes.Root
  description: Representation of the object described by the mciop artifact, capable
of being profiled by the properties of the MciopProfile information element defined in
ETSI GS NFV-IFA 011.
  capabilities:
    deployable_module_member:
      type: tosca.capabilities.nfv.DeployableModuleMember
      occurrences: [ 1, UNBOUNDED ]
  requirements:
    - associatedVdu:
      capability: tosca.capabilities.nfv.AssociableVdu
      relationship: tosca.relationships.nfv.MciopAssociates
      node: tosca.nodes.nfv.Vdu.OsContainerDeployableUnit
      occurrences: [1, UNBOUNDED]
```

6.8.14.7 Additional Requirements

The dependency requirement as defined in TOSCA-Simple-Profile-YAML-v1.3 [20] may be used towards other Mciop nodes to express the order of deployment.

Node templates of type `tosca.nodes.nfv.Mciop` may contain an artifact definition of type `tosca.artifacts.nfv.HelmChart`. There shall be a maximum number of one such artifact definition in a `tosca.nodes.nfv.Mciop` node template.

Node templates of type `tosca.nodes.nfv.Mciop` may contain an artifact definition of type `tosca.artifacts.nfv.HelmParamMappingScript` and an artifact of type `HelmParamMappingRule`. There shall be a maximum number of one such artifact definition of each type in a `tosca.nodes.nfv.Mciop` node template. If there is no artifact definition of type `HelmParamMappingScript` there shall be no artifact definition of type `HelmParamMappingRule`.

6.8.14.8 Example

See clause A.18.

6.8.15 `tosca.nodes.nfv.VirtualCp`

6.8.15.1 Description

A `VirtualCp` node type represents the `VirtualCpd` information element as defined in ETSI GS NFV-IFA 011 [1], which describes a requirement to create a virtual connection point allowing the access to a number of VNFC instances (based on their respective VDUs). Table 6.8.15.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.15.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VirtualCp</code>
Type Qualified Name	<code>toscanfv:VirtualCp</code>
Type URI	<code>tosca.nodes.nfv.VirtualCp</code>

6.8.15.2 Properties

The properties of the `VirtualCp` node type shall comply with the provisions set out in table 6.8.15.2-1.

Table 6.8.15.2-1: Properties

Name	Required	Type	Constraints	Description
<code>additionalServiceData</code>	no	List of <code>tosca.datatypes.nfv.AdditionalServiceData</code>		Additional service identification data of the <code>VirtualCp</code> exposed to NFV-MANO.
<code>address_pool_name_configurable</code>	yes	boolean	default: false	Indicates whether the corresponding MCIO supports configuration of an address pool name. See notes 1 and 2.
NOTE 1: The property is only relevant if the <code>virtualCp</code> is instantiated in a cluster that supports configuration of IP address pools for <code>virtualCps</code> . Otherwise it shall be ignored. MetalLB is an example of a solution for Kubernetes® that supports configuration of address pools for load balancer services.				
NOTE 2: The value of this property and the value of the <code>ip_address_assignment</code> property (see clause 9.2.2) are uncorrelated. In particular, a value false of the <code>ip_address_assignment</code> property does not prevent a value true of the <code>address_pool_name_configurable</code> property.				

6.8.15.3 Attributes

None.

6.8.15.4 Requirements

The requirements of the VirtualCp node type shall comply with the provisions set out in table 6.8.15.4-1.

Table 6.8.15.4-1: Requirements

Name	Required	Capability type	Constraints	Description
target	yes	tosca.capabilities.Node		Target shall be Vdu.OsContainerDeployableUnit
virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Describes the requirements for linking to virtual link

6.8.15.5 Definition

The syntax of the VirtualCp node type shall comply with the following definition:

```

tosca.nodes.nfv.VirtualCp:
  derived_from: tosca.nodes.nfv.Cp
  description: Describes a virtual connection point allowing access to a number of
VNFC instances (based on their respective VDUs).
  properties:
    additionalServiceData:
      type: list
      entry_schema:
        type: tosca.datatypes.nfv.AdditionalServiceData
      description: Additional service identification data of the VirtualCp exposed
to NFV-MANO
      required: false
      address_pool_name_configurable:
        type: boolean
        description: Indicates whether the corresponding MCIO supports configuration
of an address pool name.
        required: true
        default: false
      requirements:
        - target:
            capability: tosca.capabilities.Node
            relationship: tosca.relationships.DependsOn
            node: tosca.nodes.nfv.Vdu.OsContainerDeployableUnit
            occurrences: [ 1, UNBOUNDED ]
        - virtual_link:
            capability: tosca.capabilities.nfv.VirtualLinkable
            relationship: tosca.relationships.nfv.VirtualLinksTo
            occurrences: [0, 1]

```

6.8.15.5a Additional requirements

In the node type "VirtualCp", the value "mac_address" is not applicable in the "address_type" property inherited from the "Cp" node type.

In a VNFD service template, if this VirtualCp represents a Kubernetes® load balancer service realized by one or a set of OS containers:

- if the IP address is configurable in the declarative descriptor of the corresponding MCIO, the property ip_address_assignment shall be set to true in the L3AddressData;
- if the IP address is not configurable in the declarative descriptor of the corresponding MCIO, the property ip_address_assignment shall be set to false and the ip_address_assignment_subtype shall be set to dynamic.

If this VirtualCp represents a Kubernetes® NodePort service:

- the property `ip_address_assignment` shall be set to false and the `ip_address_assignment_subtype` shall be set to dynamic.

Indicating which Kubernetes® service type VirtualCp represents is not supported in the present document.

6.8.15.6 Example

```
VirtCp1:
  type: toska.nodes.nfv.VirtualCp
  properties:
    layer_protocols: [ tcp ]
    protocol:
      - address_data:
          - address_type: ip_address
            l3_address_data:
                floating_ip_activated: true
                ip_address_assignment: false
                ip_address_type: ipv6
                number_of_ip_address: 1
            associated_layer_protocol: ipv6
      additionalServiceData:
          - portData:
              - name: AccessService
                protocol: tcp
                port: 8001
                portConfigurable: false
              - name: ConfigService
                protocol: tcp
                port: 443
                portConfigurable: false
    requirements:
      - target: Vdu_1

Vdu_1:
  type: toska.nodes.nfv.Vdu.OsContainerDeployableUnit
  properties:
    name: "SWLB"
    description: "Software Load Balancer"
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 4
  requirements:
    - container: Vdu_1_Container_1
```

6.8.16 toska.nodes.nfv.DeployableModule

6.8.16.1 Description

A DeployableModule node type represents the DeployableModule information element as defined in ETSI GS NFV-IFA 011 [1], which describes a set of optional VDUs. Table 6.8.16.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.16.1-1: Type name, shorthand, and URI

Shorthand Name	DeployableModule
Type Qualified Name	toscanfv:DeployableModule
Type URI	tosca.nodes.nfv.DeployableModule

6.8.16.2 Properties

The properties of the DeployableModule node type shall comply with the provisions set out in table 6.8.16.2-1.

Table 6.8.16.2-1: Properties

Name	Required	Type	Constraints	Description
name	yes	string		Name of the deployable module.
description	no	string		Describes the DeployableModule, e.g. in terms of the function performed by the VNFCs deployed with their associated VDUs.

6.8.16.3 Attributes

None.

6.8.16.4 Requirements

The requirements of the DeployableModule node type shall comply with the provisions set out in table 6.8.16.4-1.

Table 6.8.16.4-1: Requirements

Name	Required	Capability type	Constraints	Description
member	yes	tosca.capabilities.nfv. DeployableModuleMember		Describes the requirements for deploying VDUs

6.8.16.5 Capabilities

None.

6.8.16.6 Definition

The syntax of the DeployableModule node type shall comply with the following definition:

```

tosca.nodes.nfv.DeployableModule:
  derived_from: toska.nodes.nfv.Root
  description: describes a set of optional VDUs.
  properties:
    name:
      type: string
      description: Name of the deployable module
      required: true
    description:
      type: string
      description: Describes the DeployableModule, e.g. in terms of the function
performed by the VNFCs deployed with their associated VDUs.
      required: false
  requirements:
    - member:
      capability: toska.capabilities.nfv.DeployableModuleMember
      relationship: toska.relationships.nfv.DeployableModuleAssociates
      occurrences: [1, UNBOUNDED]

```

6.8.16.7 Example

See clause A.21.

6.8.17 toska.nodes.nfv.PaaSServiceRequest

6.8.17.1 Description

The PaaSServiceRequest node type describes the information about and requirements by the VNF on the PaaS Services (e.g. VNF Common/Dedicated Services) that the VNF needs for its operation as specified in ETSI GS NFV-IFA 011 [1].

Table 6.8.17.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.17.1-1: Type name, shorthand, and URI

Shorthand Name	PaaSServiceRequest
Type Qualified Name	tosca.nfv.PaaSServiceRequest
Type URI	tosca.nodes.nfv.PaaSServiceRequest

6.8.17.2 Properties

The properties of the PaaSServiceRequest node type shall comply with the provisions set out in table 6.8.17.2-1.

Table 6.8.17.2-1: Properties

Name	Required	Type	Constraints	Description
name	no	string		Provides the human readable name of the requested PaaS Service.
is_mandatory	yes	boolean	default: false	Indicates whether the requested PaaS Service is mandatory for successful VNF operation. Default is FALSE, i.e. the PaaS Service is optional for successful VNF operation.
type	yes	string		The type of PaaS Service. The value of this property is expected to be matched against values of the registered PaaS Services in the PaaS Services Repository (PSR).
min_version	no	string		In the case the PaaS Service is versioned, it specifies the minimum version of the requested PaaS Service. See note.
preferred_version	no	string		In the case the PaaS Service is versioned, it specifies the preferred version of the requested PaaS Service. See note.
usage_format	yes	string	Valid values: See YAML definition constraints	Describes the usage format of the PaaS Service, as specified in [1].
specific_parameters	no	map of string		Provides attributes specific to the type of the PaaS Service. This property is reserved for future use in the present document.
NOTE: It is assumed that some form of "semantic versioning" is used to define the version of PaaS Services, so that minimum and preferred PaaS Service versions can be matched by the NFV-MANO.				

6.8.17.3 Attributes

None.

6.8.17.4 Requirements

None.

6.8.17.5 Capabilities

The capabilities of the PaasServiceRequest node type shall comply with the provisions set out in table 6.8.17.5-1.

Table 6.8.17.5-1: Capabilities

Name	Type	Constraints	Description
paas_service_requesting	tosca.capabilities.nfv.PaasServiceRequestable		Defines ability of PaasServiceRequestable.

6.8.17.6 Definition

The syntax of the PaasServiceRequest node type shall comply with the following definition:

```

tosca.nodes.nfv.PaasServiceRequest:
  derived_from: tosca.nodes.Root
  description: describes the information about and requirements by the VNF on the
PaaS Services that the VNF needs for its operation as specified in [1].
  properties:
    name:
      type: string
      description: Provides the human readable name of the requested PaaS Service.
      required: false
    is_mandatory:
      type: boolean
      description: Indicates whether the PaaS Service is mandatory for successful VNF
operation.
      required: true
      default: false
    type:
      type: string
      description: The type of PaaS Service.
      required: true
    min_version:
      type: string
      description: In the case the PaaS Service is versioned, it specifies the
minimum version of the requested PaaS Service.
      required: false
    preferred_version:
      type: string
      description: In the case the PaaS Service is versioned, it specifies the
preferred version of the requested PaaS Service.
      required: false
    usage_format:
      type: string
      description: Describes the usage format of the PaaS Service, as specified in
[1].
      required: true
      constraints:
        - valid_values: [ common, dedicated, undefined ]
    specific_parameters:
      type: map
      description: Provides attributes specific to the type of the PaaS Service. This
property is reserved for future use in the present document.
      required: false
      entry_schema:
        type: string
      capabilities:

```

```

paas_service_requesting:
  type: toska.capabilities.nfv.PaaSServiceRequestable
  occurrences: [ 1, UNBOUNDED ]

```

6.8.17.7 Additional requirements

None.

6.8.17.8 Example

None.

6.8.18 toska.nodes.nfv.PaaSServiceProfile

6.8.18.1 Description

The PaaSServiceProfile node type provides additional information about a PaaS Service request by the VNF as specified in ETSI GS NFV-IFA 011 [1].

Table 6.8.18.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.18.1-1: Type name, shorthand, and URI

Shorthand Name	PaasServiceProfile
Type Qualified Name	toscanfv:PaasServiceProfile
Type URI	tosca.nodes.nfv.PaaSServiceProfile

6.8.18.2 Properties

The properties of the PaaSServiceProfile node type shall comply with the provisions set out in table 6.8.18.2-1.

Table 6.8.18.2-1: Properties

Name	Required	Type	Constraints	Description
number_of_instances	yes	integer	greater_or_equal: 0	Number of instances of the PaaS Service that is requested for this profile.

6.8.18.3 Attributes

None.

6.8.18.4 Requirements

The requirements of the PaaSServiceProfile node type shall comply with the provisions set out in table 6.8.18.4-1.

Table 6.8.18.4-1: Requirements

Name	Required	Capability type	Constraints	Description
paas_service_request	yes	tosca.capabilities.nfv.PaaSServiceRequestable		Describes the requirement for requesting PaaS Service.

6.8.18.5 Capabilities

The capabilities of the PaaSServiceProfile node type shall comply with the provisions set out in table 6.8.18.5-1.

Table 6.8.18.5-1: Capabilities

Name	Type	Constraints	Description
associable_paas_service	tosca.capabilities.nfv.AssociablePaasService		Defines ability of AssociablePaasService.

6.8.18.6 Definition

The syntax of the PaasServiceProfile node type shall comply with the following definition:

```

tosca.nodes.nfv.PaaSServiceProfile:
  derived_from: tosca.nodes.Root
  description: Provides additional information about a PaaS Service request by the
VNF as defined in ETSI GS NFV-IFA 011.
  properties:
    number_of_instances:
      type: integer
      description: Number of instances of the PaaS Service that is requested for
this profile.
      required: true
      constraints:
        - greater_or_equal: 0
    requirements:
      - paas_service_request:
          capability: tosca.capabilities.nfv.PaaSServiceRequestable
          relationship: tosca.relationships.nfv.PaaSServiceRequestsTo
          occurrences: [1, 1]
    capabilities:
      associable_paas_service:
        type: tosca.capabilities.nfv.AssociablePaasService
        occurrences: [ 1, UNBOUNDED ]

```

6.8.18.7 Additional requirements

None.

6.8.18.8 Example

None.

6.8.19 tosca.nodes.nfv.Certificate

6.8.19.1 Description

The Certificate node type describes the certificate to be used by the VNF in delegation-mode, as defined in ETSI GS NFV-IFA 011 [1].

Table 6.8.19.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.8.19.1-1: Type name, shorthand, and URI

Shorthand Name	Certificate
Type Qualified Name	toscanfv:Certificate
Type URI	tosca.nodes.nfv.Certificate

6.8.19.2 Properties

The properties of the Certificate node type shall comply with the provisions set out in table 6.8.19.2-1.

Table 6.8.19.2-1: Properties

Name	Required	Type	Constraints	Description
name	yes	string		Human readable name of the certificate.
certificate_type	yes	string	Allowed values: • See YAML definition constraints	Type of the certificate.
certificate_base_profile	no	tosca.datatypes.nfv.CertificateBaseProfile		Basic information for this certificate when issuing a CSR. Shall be present in the delegation mode. Otherwise shall be absent.
csr_requirements	no	list of tosca.datatypes.nfv.CSRRRequirements		Requirements for Certificate when issuing CSR.

6.8.19.3 Attributes

None.

6.8.19.4 Requirements

None.

6.8.19.5 Capabilities

The capabilities of the Certificate node type shall comply with the provisions set out in table 6.8.19.5-1.

Table 6.8.19.5-1: Capabilities

Name	Type	Constraints	Description
installable_certificate	tosca.capabilities.nfv.InstallableCertificate		Defines the capabilities of Installable_certificate.

6.8.19.6 Definition

The syntax of the Certificate node type shall comply with the following definition:

```
tosca.nodes.nfv.Certificate:
  derived_from: tosca.nodes.Root
  description: describes the certificate to be used by the VNF
  properties:
    name:
      type: string
      description: Human readable name of the certificate
      required: true
    certificate_type:
      type: string
      description: Type of the certificate.
      required: true
      constraints:
        - valid_values: [ VNFCI_CERT, VNFOAM_CERT ]
    certificate_base_profile:
      type: tosca.datatypes.nfv.CertificateBaseProfile
```



```

    description: Basic information for this certificate when issuing a CSR. Shall
be present in the delegation mode. Otherwise shall be absent.
    required: false
    csr_requirements:
      type: list
      description: Requirements for Certificate when issuing CSR.
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.CSRRequirements
    capabilities:
      installable_certificate:
        type: tosca.capabilities.nfv.InstallableCertificate
        occurrences: [ 1, UNBOUNDED ]

```

6.8.19.7 Additional requirements

The VNF topology template shall only contain node templates of type `tosca.nodes.nfv.Certificate` when it supports the delegation-mode. The "installable_certificate" requirements in `tosca.nodes.nfv.Vdu.Compute` or `tosca.nodes.nfv.Vdu.OsContainerDeployableUnit` node template shall be present in each low level service template when the particular VDU in particular Deployment Flavour supports delegation-mode. If the VNF supports direct mode or does not require a certificate VNF topology template shall not contain any `tosca.nodes.nfv.Certificate` node template.

6.8.19.8 Example

```

VNF_Certificate_1:
  type: tosca.nodes.nfv.Certificate
  properties:
    name: VNF Certificate #1
    certificate_type: VNFCI_CERT
    certificate_base_profile:
      type: tosca.datatypes.nfv.CertificateBaseProfile
      properties:
        id: "aaacb5b681f937671f3738aaa1f9ed2e"
        issuer: "CA1"
        issuer_unique_identifier: "b489d994d1521aa8571c69f8ed4a3620"
        subject:
          type: tosca.datatypes.nfv.CertSubjectData
          properties:
            common_name: "example.mycompany.com"
            organization: "MyCompany"
            email_address: "abc@example.mycompany.com"
        subject_unique_identifier: "a872f93cf9a5661ae11154fa2e6c66e4"
        basic_constraints: "critical,CA:true,pathlen:1"
        subject_alt_name: "DNS name=example.mycompany.com"
        name_constraints: "critical,permitted;DNS:.example.mycompany.com"
    csr_requirements:
      - id: "bc5746547b5d9d29a17d1f625677bc71"
        supported_certificate_version: "X.509_ver3"
        supported_signature: [ "sha256withRSAEncryption" ]
        supported_max_keylength: 4096
        supported_min_keylength: 2048
        maximum_validity_period: 730
        minimum_validity_period: 365
        key_usages: [ "Digital signature", "Key encipherment", "Key agreement" ]
  capabilities:
    - installable_certificate: installable.MyCertificate

```

6.9 Group Types

6.9.1 `tosca.groups.nfv.PlacementGroup`

6.9.1.1 Description

PlacementGroup is used for describing the affinity or anti-affinity relationship applicable between the virtualisation containers to be created based on different VDUs, or between internal VLs to be created based on different VnfVirtualLinkDesc(s) or between the workloads being deployed based on different Mciops. Table 6.9.1.1-1 specifies the declared names for this group type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.9.1.1-1: Type name, shorthand, and URI

Shorthand Name	PlacementGroup
Type Qualified Name	toscanfv:PlacementGroup
Type URI	tosca.groups.nfv.PlacementGroup

6.9.1.2 Properties

The properties of the PlacementGroup group type shall comply with the provisions set out in table 6.9.1.2-1.

Table 6.9.1.2-1: Properties

Name	Required	Type	Constraints	Description
description	yes	string		Human readable description of the group

6.9.1.3 Definition

The syntax of the PlacementGroup group type shall comply with the following definition:

```
tosca.groups.nfv.PlacementGroup:
  derived_from: toska.groups.Root
  description: PlacementGroup is used for describing the affinity or anti-affinity
relationship applicable between the virtualisation containers to be created based on
different VDUs, or between internal VLs to be created based on different
VnfVirtualLinkDesc(s) or between the workloads being deployed based on different
Mciops
  properties:
    description:
      type: string
      description: Human readable description of the group
      required: true
  members: [ toska.nodes.nfv.Vdu.Compute,
tosca.nodes.nfv.Vdu.OsContainerDeployableUnit, toska.nodes.nfv.VnfVirtualLink,
tosca.nodes.nfv.Mciop ]
```

6.9.1.4 Additional Requirements

A group with type `tosca.groups.nfv.PlacementGroup` shall contain more than one member with the same node type when used as the target of an AffinityRule or AntiAffinityRule policy.

6.9.1.5 Examples

See clause 6.10.10.5.

6.10 Policy Types

6.10.1 `tosca.policies.nfv.InstantiationLevels`

6.10.1.1 Description

The `InstantiationLevels` type is a policy type representing all the instantiation levels of resources to be instantiated within a deployment flavour and including default instantiation level in term of the number of VNFC instances to be created as defined in ETSI GS NFV-IFA 011 [1]. This policy concerns the whole VNF (deployment flavour) represented by the `topology_template` and thus has no explicit target list. Table 6.10.1.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.1.1-1: Type name, shorthand, and URI

Shorthand Name	<code>InstantiationLevels</code>
Type Qualified Name	<code>toscanfv:InstantiationLevels</code>
Type URI	<code>tosca.policies.nfv.InstantiationLevels</code>

6.10.1.2 Properties

The properties of the `InstantiationLevels` policy type shall comply with the provisions set out in table 6.10.1.2-1.

Table 6.10.1.2-1: Properties

Name	Required	Type	Constraints	Description
<code>levels</code>	yes	map of <code>tosca.datatypes.nfv.InstantiationLevel</code>		Describes the various levels of resources that can be used to instantiate the VNF using this flavour.
<code>default_level</code>	no	string		The default instantiation level for this flavour.

6.10.1.3 Definition

The syntax of the `InstantiationLevels` policy type shall comply with the following definition:

```
tosca.policies.nfv.InstantiationLevels:
  derived_from: toska.policies.Root
  description: The InstantiationLevels type is a policy type representing all the
instantiation levels of resources to be instantiated within a deployment flavour and
including default instantiation level in term of the number of VNFC instances to be
created as defined in ETSI GS NFV-IFA 011.
  properties:
    levels:
      type: map # key: levelId
      description: Describes the various levels of resources that can be used to
instantiate the VNF using this flavour.
      required: true
      entry_schema:
        type: toska.datatypes.nfv.InstantiationLevel
      constraints:
        - min_length: 1
    default_level:
      type: string # levelId
      description: The default instantiation level for this flavour.
      required: false # required if multiple entries in levels
```

6.10.2 `tosca.policies.nfv.VduInstantiationLevels`

6.10.2.1 Description

The `VduInstantiationLevels` type is a policy type representing all the instantiation levels of resources to be instantiated within a deployment flavour in term of the number of VNFC instances to be created from each `vdu.Compute` as defined in ETSI GS NFV-IFA 011 [1].

Table 6.10.2.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.2.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VduInstantiationLevels</code>
Type Qualified Name	<code>tosca.nfv.VduInstantiationLevels</code>
Type URI	<code>tosca.policies.nfv.VduInstantiationLevels</code>

6.10.2.2 Properties

The properties of the `VduInstantiationLevels` policy type shall comply with the provisions set out in table 6.10.2.2-1.

Table 6.10.2.2-1: Properties

Name	Required	Type	Constraints	Description
<code>levels</code>	yes	map of <code>tosca.datatypes.nfv.VduLevel</code>		Describes the <code>Vdu.Compute</code> levels of resources that can be used to instantiate the VNF using this flavour.

6.10.2.3 Definition

The syntax of the `VduInstantiationLevels` policy type shall comply with the following definition:

```
tosca.policies.nfv.VduInstantiationLevels:
  derived_from: tosca.policies.Root
  description: The VduInstantiationLevels type is a policy type representing all the
instantiation levels of resources to be instantiated within a deployment flavour in
term of the number of VNFC instances to be created from each vdu.Compute. as defined
in ETSI GS NFV-IFA 011
  properties:
    levels:
      type: map # key: levelId
      description: Describes the Vdu.Compute levels of resources that can be used to
instantiate the VNF using this flavour
      required: true
      entry_schema:
        type: tosca.datatypes.nfv.VduLevel
      constraints:
        - min_length: 1
      targets: [ tosca.nodes.nfv.Vdu.Compute,
tosca.nodes.nfv.Vdu.OsContainerDeployableUnit ]
```

6.10.2.4 Additional Requirements

A `VduInstantiationLevels` policy shall contain an entry for each instantiation level (and only for them) defined in the `InstantiationLevels` policy.

6.10.3 tosa.policies.nfv.VirtualLinkInstantiationLevels

6.10.3.1 Description

The VirtualLinkInstantiationLevels type is a policy type representing all the instantiation levels of virtual link resources to be instantiated within a deployment flavour as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.3.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.3.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualLinkInstantiationLevels
Type Qualified Name	toscanfv:VirtualLinkInstantiationLevels
Type URI	tosca.policies.nfv.VirtualLinkInstantiationLevels

6.10.3.2 Properties

The properties of the VirtualLinkInstantiationLevels policy type shall comply with the provisions set out in table 6.10.3.2-1.

Table 6.10.3.2-1: Properties

Name	Required	Type	Constraints	Description
levels	yes	map of tosca.datatypes.nfv.VirtualLinkBitrateLevel		Describes the virtual link levels of resources that can be used to instantiate the VNF using this flavour.

6.10.3.3 Definition

The syntax of the VirtualLinkInstantiationLevels policy type shall comply with the following definition:

```
tosca.policies.nfv.VirtualLinkInstantiationLevels:
  derived_from: tosa.policies.Root
  description: The VirtualLinkInstantiationLevels type is a policy type representing
all the instantiation levels of virtual link resources to be instantiated within a
deployment flavour as defined in ETSI GS NFV-IFA 011.
  properties:
    levels:
      type: map # key: levelId
      description: Describes the virtual link levels of resources that can be used
to instantiate the VNF using this flavour.
      required: true
      entry_schema:
        type: tosa.datatypes.nfv.VirtualLinkBitrateLevel
      constraints:
        - min_length: 1
  targets: [ tosa.nodes.nfv.VnfVirtualLink ]
```

6.10.3.4 Additional Requirements

A VirtualLinkInstantiationLevels policy shall contain an entry for each instantiation level (and only for them) defined in the InstantiationLevels policy.

6.10.4 Void

6.10.5 tosca.policies.nfv.ScalingAspects

6.10.5.1 Description

The ScalingAspects type is a policy type representing the scaling aspects used for horizontal scaling as defined in ETSI GS NFV-IFA 011 [1]. This policy concerns the whole VNF (deployment flavour) represented by the topology_template and thus has no explicit target list. Table 6.10.5.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.5.1-1: Type name, shorthand, and URI

Shorthand Name	ScalingAspects
Type Qualified Name	toscanfv:ScalingAspects
Type URI	tosca.policies.nfv.ScalingAspects

6.10.5.2 Properties

The properties of the ScalingAspects policy type shall comply with the provisions set out in table 6.10.5.2-1.

Table 6.10.5.2-1: Properties

Name	Required	Type	Constraints	Description
aspects	yes	Map of tosca.datatypes.nfv. ScalingAspect		Describe maximum scale level for total number of scaling steps that can be applied to a particular aspect.

6.10.5.3 Definition

The syntax of the ScalingAspects policy type shall comply with the following definition:

```
tosca.policies.nfv.ScalingAspects:
  derived_from: tosca.policies.Root
  description: The ScalingAspects type is a policy type representing the scaling
aspects used for horizontal scaling as defined in ETSI GS NFV-IFA 011
  properties:
    aspects:
      type: map # key: aspectId
      description: Describe maximum scale level for total number of scaling steps
that can be applied to a particular aspect
      required: true
      entry_schema:
        type: tosca.datatypes.nfv.ScalingAspect
      constraints:
        - min_length: 1
```

6.10.5.4 Additional Requirements

A scaling aspect for which only one scaling delta is defined in VduScalingAspectDeltas and VirtualLinkBitrateScalingAspectDeltas policies is called a "uniform aspect". In the case of "uniform aspect", the step_deltas properties of tosca.datatypes.nfv.ScalingAspect is optional. If step_deltas is included, the value shall be a list of entries of step_deltas.

6.10.5.5 Examples

See clause A.6.

6.10.6 toska.policies.nfv.VduScalingAspectDeltas

6.10.6.1 Description

The VduScalingAspectDeltas type is a policy type representing the Vdu.Compute detail of an aspect deltas used for horizontal scaling, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.6.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.6.1-1: Type name, shorthand, and URI

Shorthand Name	VduScalingAspectDeltas
Type Qualified Name	toscanfv.VduScalingAspectDeltas
Type URI	tosca.policies.nfv.VduScalingAspectDeltas

6.10.6.2 Properties

The properties of the VduScalingAspectDeltas policy type shall comply with the provisions set out in table 6.10.6.2-1.

Table 6.10.6.2-1: Properties

Name	Required	Type	Constraints	Description
aspect	yes	string		Represents the scaling aspect to which this policy applies. When two or more policies are related to the same scaling aspect, i.e. have the same value of the aspect property, all the respective targets of the policies (Vdu.Compute or Vdu.OsContainerDeployableUnit), if they belong to a deployable module, shall belong exactly to the same deployable modules. See note.
deltas	yes	map of toska.datatypes.nfv.VduLevel		Describes the Vdu.Compute scaling deltas to be applied for every scaling steps of a particular aspect.
NOTE: This means that one scaling aspect shall not combine VDUs that belong to different deployable modules nor VDUs that belong to a deployable module and VDUs that don't belong to any deployable module.				

6.10.6.3 Definition

The syntax of the VduScalingAspectDeltas policy type shall comply with the following definition:

```
tosca.policies.nfv.VduScalingAspectDeltas:
  derived_from: toska.policies.Root
  description: The VduScalingAspectDeltas type is a policy type representing the
Vdu.Compute detail of an aspect deltas used for horizontal scaling, as defined in ETSI
GS NFV-IFA 011
  properties:
    aspect:
      type: string
      description: Represents the scaling aspect to which this policy applies. When
two or more policies are related to the same scaling aspect, i.e. have the same value
of the aspect property, all the respective targets of the policies (Vdu.Compute or
Vdu.OsContainerDeployableUnit), if they belong to a deployable module, shall belong
exactly to the same deployable modules.
      required: true
    deltas:
      type: map # key: scalingDeltaId
      description: Describes the Vdu.Compute scaling deltas to be applied for every
scaling steps of a particular aspect.
```

```

    required: true
    entry_schema:
      type: toska.datatypes.nfv.VduLevel
    constraints:
      - min_length: 1
    targets: [ toska.nodes.nfv.Vdu.Compute,
tosca.nodes.nfv.Vdu.OsContainerDeployableUnit ]

```

6.10.6.4 Additional Requirements

In the case of "uniform aspect", the deltas properties shall have only one entry.

If a policy definition of this type is included in a service template, a policy definition of the type VduInitialDelta defined in clause 6.10.8 of the present document shall also be included with the same target.

6.10.6.5 Examples

See clause A.6.

6.10.7 toska.policies.nfv.VirtualLinkBitrateScalingAspectDeltas

6.10.7.1 Description

The VirtualLinkBitrateScalingAspectDeltas type is a policy type representing the VnfVirtualLink detail of an aspect deltas used for horizontal scaling, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.7.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.7.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualLinkBitrateScalingAspectDeltas
Type Qualified Name	toscanfv:VirtualLinkBitrateScalingAspectDeltas
Type URI	tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas

6.10.7.2 Properties

The properties of the VirtualLinkBitrateScalingAspectDeltas policy type shall comply with the provisions set out in table 6.10.7.2-1.

Table 6.10.7.2-1: Properties

Name	Required	Type	Constraints	Description
aspect	yes	string		Represents the scaling aspect to which this policy applies.
deltas	yes	map of toska.datatypes.nfv.VirtualLinkBitrateLevel		Describes the VnfVirtualLink scaling deltas to be applied for every scaling steps of a particular aspect.

6.10.7.3 Definition

The syntax of the VirtualLinkBitrateScalingAspectDeltas policy type shall comply with the following definition:

```

tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas:
  derived_from: toska.policies.Root
  description: The VirtualLinkBitrateScalingAspectDeltas type is a policy type
representing the VnfVirtualLink detail of an aspect deltas used for horizontal
scaling, as defined in ETSI GS NFV-IFA 011.
  properties:

```



```

    aspect:
      type: string
      description: Represents the scaling aspect to which this policy applies.
      required: true
    deltas:
      type: map # key: scalingDeltaId
      description: Describes the VnfVirtualLink scaling deltas to be applied for
every scaling steps of a particular aspect.
      required: true
      entry_schema:
        type: toska.datatypes.nfv.VirtualLinkBitrateLevel
      constraints:
        - min_length: 1
    targets: [ toska.nodes.nfv.VnfVirtualLink ]

```

6.10.7.4 Additional Requirements

In the case of "uniform aspect", the deltas properties shall have only one entry.

If a policy definition of this type is included in a service template, a policy definition of the type VirtualLinkBitrateInitialDelta defined in clause 6.10.9 of the present document shall also be included with the same target.

6.10.7.5 Examples

See clause A.6.

6.10.8 toska.policies.nfv.VduInitialDelta

6.10.8.1 Description

The VduInitialDelta type is a policy type representing the Vdu.Compute detail of an initial delta used for horizontal scaling, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.8.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.8.1-1: Type name, shorthand, and URI

Shorthand Name	VduInitialDelta
Type Qualified Name	toscanfv:VduInitialDelta
Type URI	tosca.policies.nfv.VduInitialDelta

6.10.8.2 Properties

The properties of the VduInitialDelta policy type shall comply with the provisions set out in table 6.10.8.2-1.

Table 6.10.8.2-1: Properties

Name	Required	Type	Constraints	Description
initial_delta:	yes	tosca.datatypes.nfv.VduLevel		Represents the initial minimum size of the VNF.

6.10.8.3 Definition

The syntax of the VduInitialDelta policy type shall comply with the following definition:

```

tosca.policies.nfv.VduInitialDelta:
  derived_from: toska.policies.Root

```

```

description: The VduInitialDelta type is a policy type representing the
Vdu.Compute detail of an initial delta used for horizontal scaling, as defined in
ETSI GS NFV-IFA 011.
properties:
  initial_delta:
    type: toska.datatypes.nfv.VduLevel
    description: Represents the initial minimum size of the VNF.
    required: true
  targets: [ toska.nodes.nfv.Vdu.Compute,
toska.nodes.nfv.Vdu.OsContainerDeployableUnit ]

```

6.10.8.4 Examples

See clause A.6.

6.10.9 toska.policies.nfv.VirtualLinkBitrateInitialDelta

6.10.9.1 Description

The VirtualLinkBitrateInitialDelta type is a policy type representing the VnfVirtualLink detail of an initial deltas used for horizontal scaling, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.9.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.9.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualLinkBitrateInitialDelta
Type Qualified Name	toscanfv:VirtualLinkBitrateInitialDelta
Type URI	tosca.policies.nfv.VirtualLinkBitrateInitialDelta

6.10.9.2 Properties

The properties of the VirtualLinkBitrateInitialDelta policy type shall comply with the provisions set out in table 6.10.9.2-1.

Table 6.10.9.2-1: Properties

Name	Required	Type	Constraints	Description
initial_delta:	yes	tosca.datatypes.nfv.VirtualLinkBitrateLevel		Represents the initial minimum size of the VNF.

6.10.9.3 Definition

The syntax of the VirtualLinkBitrateInitialDelta policy type shall comply with the following definition:

```

tosca.policies.nfv.VirtualLinkBitrateInitialDelta:
  derived_from: toska.policies.Root
  description: The VirtualLinkBitrateInitialDelta type is a policy type
representing the VnfVirtualLink detail of an initial deltas used for horizontal
scaling, as defined in ETSI GS NFV-IFA 011.
  properties:
    initial_delta:
      type: toska.datatypes.nfv.VirtualLinkBitrateLevel
      description: Represents the initial minimum size of the VNF.
      required: true
  targets: [ toska.nodes.nfv.VnfVirtualLink ]

```

6.10.9.4 Examples

See clause A.6.

6.10.10 AffinityRule, AntiAffinityRule

6.10.10.1 Description

The AffinityRule or AntiAffinityRule describes the affinity or anti-affinity rules applicable for the defined targets:

- If there is only one node template with node type `tosca.nodes.nfv.Vdu.Compute` or `tosca.nodes.nfv.Vdu.OsContainerDeployableUnit` or `tosca.nodes.nfv.VnfVirtualLink` set as the targets, the AffinityRule or AntiAffinityRule applies between the virtualisation containers to be created based on a particular VDU, or between internal VLs to be created based on a particular VnfVirtualLinkDesc as described in ETSI GS NFV-IFA 011 [1].
- If there are more than one node templates with node type `tosca.nodes.nfv.Vdu.Compute` or `tosca.nodes.nfv.Vdu.OsContainerDeployableUnit` or `tosca.nodes.nfv.VnfVirtualLink` or `tosca.nodes.nfv.Mciop` set as the targets, or a group with type `tosca.groups.nfv.PlacementGroup` which contains more than one members set as targets, the AffinityRule or AntiAffinityRule applies between the virtualisation containers to be created based on different VDUs, or between internal VLs to be created based on different VnfVirtualLinkDesc(s) or between sets of virtualisation containers, realized by OS containers, to be created based on different MCIOPs as described in ETSI GS NFV-IFA 011 [1].

Tables 6.10.10.1-1 and 6.10.10.1-2 specify the declared names for the policy types. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.10.1-1: Type name, shorthand, and URI

Shorthand Name	AffinityRule
Type Qualified Name	toscanfv:AffinityRule
Type URI	tosca.policies.nfv.AffinityRule

Table 6.10.10.1-2: Type name, shorthand, and URI

Shorthand Name	AntiAffinityRule
Type Qualified Name	toscanfv:AntiAffinityRule
Type URI	tosca.policies.nfv.AntiAffinityRule

6.10.10.2 Properties

The properties of the AffinityRule and AntiAffinityRule types shall comply with the provisions set out in table 6.10.10.2-1.

Table 6.10.10.2-1: Properties

Name	Required	Type	Constraints	Description
scope	Yes	String	Possible values are "nfvi_pop", "zone", "zone_group", "nfvi_node", "network_link_and_node", "container_namespace", "cis_node".	Specifies the scope of the affinity or anti-affinity rule. See notes 1, 2 and 3.
nfvi_maintenance_group_info	no	tosca.datatypes.nfv.NfviMaintenanceInfo		Provides information on the impact tolerance and rules to be observed when a group of instances based on the same Vdu.Compute node (for VM based VDU) is impacted during NFVI operation and maintenance (e.g. NFVI resource upgrades). See notes 4 and 5.
NOTE 1: When used in an anti-affinity relationship, the "network_link_and_node" scope is conceptually similar to link and node disjoint paths capabilities used commonly in network Traffic Engineering (TE). For example, as in Fast Reroute Resource Reservation Protocol Traffic Engineering (RSVP-TE) for Label-Switched Path (LSP) tunnels as introduced in IETF RFC 4090 [i.17]. NOTE 2: The "container" namespace is only applicable when the targets of the policy are exclusively nodes of type <code>tosca.nodes.nfv.Mciop</code>. NOTE 3: The "cis-node" scope is only applicable when the targets of the policy are exclusively nodes of type <code>tosca.nodes.nfv.Vdu.OsContainerDeployableUnit</code>. NOTE 4: The <code>nfvi_maintenance_info</code> property may only be present if there is only one node template with node type <code>tosca.nodes.nfv.Vdu.Compute</code> (for VM based VDU) set as the targets. NOTE 5: An NFVI level operation (e.g. restart of a virtual machine) can impact a VNF and the VNF can be able to tolerate only a limited number of such impacts simultaneously. The <code>nfvi_maintenance_group_info</code> provides constraints related to the tolerated simultaneous impacts on a group of resources so that negative impact on VNF functionality can be avoided during NFVI maintenance operations.				

6.10.10.3 targets

The targets of the AffinityRule and AntiAffinityRule policy types shall comply with the provisions set out in table 6.10.10.3-1.

Table 6.10.10.3-1: Targets

Name	Required	Type	Constraints	Description
targets	Yes	string[]	Valid types: <code>tosca.nodes.nfv.Vdu.Compute</code> <code>tosca.nodes.nfv.VnfVirtualLink</code> <code>tosca.groups.nfv.PlacementGroup</code> <code>tosca.nodes.nfv.Mciop</code> , <code>tosca.nodes.nfv.Vdu.OsContainerDeployableUnit</code> , <code>tosca.nodes.nfv.PaaSServiceRequest</code>	In case of LocalAffinityOrAntiAffinityRule as defined in ETSI GS NFV-IFA 011 [1], the valid type of the targets is <code>tosca.nodes.nfv.Vdu.Compute</code> or <code>tosca.nodes.nfv.Vdu.OsContainerDeployableUnit</code> or <code>tosca.nodes.nfv.VnfVirtualLink</code> . In case of affinityOrAntiAffinityGroup as defined in ETSI GS NFV-IFA 011 [1], the valid types of the targets are: <code>tosca.nodes.nfv.Vdu.Compute</code> and <code>tosca.nodes.nfv.Vdu.OsContainerDeployableUnit</code> and <code>tosca.nodes.nfv.VnfVirtualLink</code> and <code>tosca.nodes.nfv.Mciop</code> and <code>tosca.nodes.nfv.PaaSServiceRequest</code> or a <code>tosca.groups.nfv.PlacementGroup</code> .

6.10.10.4 Definition

The syntax of the AffinityRule policy type shall comply with the following definition:

```
tosca.policies.nfv.AffinityRule:
  derived_from: tosca.policies.Placement
  description: The AffinityRule describes the affinity rules applicable for the
defined targets
  properties:
    scope:
      type: string
      description: scope of the rule is an NFVI_node, an NFVI_PoP, etc.
      required: true
      constraints:
        - valid_values: [ nfvi_node, zone, zone_group, nfvi_pop,
network_link_and_node, container_namespace, cis_node ]
    nfvi_maintenance_group_info:
      type: tosca.datatypes.nfv.NfviMaintenanceInfo
      description: Provides information on the impact tolerance and rules to be
observed when a group of instances based on the same Vdu.Compute (for VM based VDU)
node is impacted during NFVI operation and maintenance (e.g. NFVI resource upgrades).
      required: false
    targets: [ tosca.nodes.nfv.Vdu.Compute, tosca.nodes.nfv.VnfVirtualLink,
tosca.groups.nfv.PlacementGroup, tosca.nodes.nfv.Mciop,
tosca.nodes.nfv.Vdu.OsContainerDeployableUnit, tosca.nodes.nfv.PaaSServiceRequest ]
```

The syntax of the AntiAffinityRule policy type shall comply with the following definition:

```
tosca.policies.nfv.AntiAffinityRule:
  derived_from: tosca.policies.Placement
  description: The AntiAffinityRule describes the anti-affinity rules applicable for
the defined targets
  properties:
    scope:
      type: string
      description: scope of the rule is an NFVI_node, an NFVI_PoP, etc.
      required: true
      constraints:
        - valid_values: [ nfvi_node, zone, zone_group, nfvi_pop,
network_link_and_node, container_namespace, cis_node ]
    nfvi_maintenance_group_info:
      type: tosca.datatypes.nfv.NfviMaintenanceInfo
      description: Provides information on the impact tolerance and rules to be
observed when a group of instances based on the same Vdu.Compute node is impacted
during NFVI operation and maintenance (e.g. NFVI resource upgrades).
      required: false
    targets: [ tosca.nodes.nfv.Vdu.Compute, tosca.nodes.nfv.VnfVirtualLink,
tosca.groups.nfv.PlacementGroup, tosca.nodes.nfv.Mciop,
tosca.nodes.nfv.Vdu.OsContainerDeployableUnit, tosca.nodes.nfv.PaaSServiceRequest ]
```

6.10.10.5 Examples

The following example template fragments illustrate the concepts:

```
node_templates:
  VDU_1:
    type: tosca.nodes.nfv.Vdu.Compute

policies:
```

```

policy_affinity_local_VDU_1:
  type: tosa.policies.nfv.AffinityRule
  targets: [ VDU_1 ]
  properties:
    scope: nfvi_node

```

The above example illustrates a local affinity rule for VDU_1.

```

node_template:
  VDU_1:
    type: tosa.nodes.nfv.Vdu.Compute

  VDU_2:
    type: tosa.nodes.nfv.Vdu.Compute

groups:
  affinityOrAntiAffinityGroup_1:
    type: tosa.groups.nfv.PlacementGroup
    members: [ VDU_1, VDU_2 ]

policies:
  policy_antiaffinity_group_1:
    type: tosa.policies.nfv.AntiAffinityRule
    targets: [ affinityOrAntiAffinityGroup_1 ]
    properties:
      scope: nfvi_node

```

The above example illustrates an anti-affinity policy among a group which contains VDU_1 and VDU_2 as members.

6.10.11 tosa.policies.nfv.Abstract.SecurityGroupRule

6.10.11.1 Description

The Abstract.SecurityGroupRule policy type is defined in clause 9.10.1 of the present document.

6.10.12 tosa.policies.nfv.SupportedVnfInterface

6.10.12.1 Description

The SupportedVnfInterface policy type represents interfaces produced by a VNF, the details to access them and the applicable connection points to use to access these interfaces. It corresponds to the VnfInterfaceDetails information element defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.12.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.12.1-1: Type name, shorthand, and URI

Shorthand Name	SupportedVnfInterface
Type Qualified Name	toscanfv:SupportedVnfInterface
Type URI	tosa.policies.nfv.SupportedVnfInterface

6.10.12.2 Properties

The properties of the SupportedVnfInterface policy type shall comply with the provisions set out in table 6.10.12.2-1.

Table 6.10.12.2-1: Properties

Name	Required	Type	Constraints	Description
interface_name	yes	string	Valid values vnf_indicator vnf_configuration vnf_lcm_coordination	Identifies an interface produced by the VNF.
details	no	tosca.datatypes.nfv.InterfaceDetails		Provide additional data to access the interface endpoint (e.g. API URI prefix).

6.10.12.3 Definition

The syntax of the SupportedVnfInterface policy type shall comply with the following definition:

```
tosca.policies.nfv.SupportedVnfInterface:
  derived_from: tosca.policies.Root
  description: this policy type represents interfaces produced by a VNF, the details
to access them and the applicable connection points to use to access these interfaces
  properties:
    interface_name:
      type: string
      description: Identifies an interface produced by the VNF.
      required: true
      constraints:
        - valid_values: [ vnf_indicator, vnf_configuration, vnf_lcm_coordination ]
    details:
      type: tosca.datatypes.nfv.InterfaceDetails
      description: Provide additional data to access the interface endpoint
      required: false
  targets: [ tosca.nodes.nfv.VnfExtCp, tosca.nodes.nfv.VduCp ]
```

6.10.12.4 Additional requirements

The valid targets for this policy type shall be the node templates representing the connection point descriptors from which to instantiate the connection point instances through which the interfaces can be accessed. This may be a VnfExtCp node template or a VduCp node template when an internal connection point is re-exposed externally.

6.10.12.5 Example

```
policies:
  policy_interface_1:
    type: tosca.policies.nfv.SupportedVnfInterface
    targets: [ MyVnfmFacingExtCp ]
    properties:
      interface_name: vnf_indicator
      details:
        uri_components:
          scheme: https
          authority:
            host: myvnf.example.com
```

6.10.13 tosa.policies.nfv.SecurityGroupRule

6.10.13.1 Description

The SecurityGroupRule type is a policy type specifying the matching criteria for the ingress and/or egress traffic to and from visited connection points as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.13.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.13.1-1: Type name, shorthand, and URI

Shorthand Name	SecurityGroupRule
Type Qualified Name	toscanfv:SecurityGroupRule
Type URI	tosca.policies.nfv.SecurityGroupRule

6.10.13.2 Properties

None.

6.10.13.3 targets

The targets of the SecurityGroupRule policy types shall comply with the provisions set out in table 6.10.13.3-1.

Table 6.10.13.3-1: Targets

Name	Required	Type	Constraints	Description
targets	yes	string[]	Valid types: tosca.nodes.nfv.VduCp, tosca.nodes.nfv.VnfExtCp, tosca.nodes.nfv.Vdu.OsContainerDeployableUnit	Target connection points of VduCp and/or VnfExtCp, or target VDUs based on one or set of OS containers. See note.
NOTE: If the OsContainerDeployableUnit is used as the target, the security rule is applicable for all the connection points related to the OsContainerDeployableUnit.				

6.10.13.4 Definition

The syntax of the SecurityGroupRule policy type shall comply with the following definition:

```
tosca.policies.nfv.SecurityGroupRule:
  derived_from: tosa.policies.nfv.Abstract.SecurityGroupRule
  description: The SecurityGroupRule type is a policy type specified the
  matching criteria for the ingress and/or egress traffic to/from visited connection
  points as defined in ETSI GS NFV-IFA 011.
  targets: [ tosa.nodes.nfv.VduCp, tosa.nodes.nfv.VnfExtCp,
  tosa.nodes.nfv.Vdu.OsContainerDeployableUnit ]
```

6.10.13.5 Additional Requirements

None.

6.10.14 tosca.policies.nfv.VnfIndicator

6.10.14.1 Description

The VnfIndicator is a base policy type for defining VNF indicator specific policies that define the conditions to assess and the action to perform when a VNF indicator changes value as defined in ETSI GS NFV-IFA 011 [1].

Table 6.10.14.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.14.1-1: Type name, shorthand, and URI

Shorthand Name	VnfIndicator
Type Qualified Name	toscanfv:VnfIndicator
Type URI	tosca.policies.nfv.VnfIndicator

6.10.14.2 Properties

The properties of the VnfIndicator policy type shall comply with the provisions set out in table 6.10.14.2-1.

Table 6.10.14.2-1: Properties

Name	Required	Type	Constraints	Description
source	yes	string	valid values: See YAML definition constraints	Describe the source of the indicator.

6.10.14.3 Definition

The syntax of the VnfIndicator policy type shall comply with the following definition:

```
tosca.policies.nfv.VnfIndicator:
  derived_from: tosca.policies.Root
  description: The VnfIndicator policy type is a base policy type for defining VNF
  indicator specific policies that define the conditions to assess and the action to
  perform when a VNF indicator changes value as defined in ETSI GS NFV-IFA 011.
  properties:
    source:
      type: string
      description: Describe the source of the indicator.
      required: true
      constraints:
        - valid_values: [ vnf, em, both_vnf_and_em ]
  targets: [ tosca.nodes.nfv.VNF ]
```

6.10.14.4 Additional requirements

The VNFD service template may include VNF indicator specific policies of VnfIndicator type with the following requirements:

- a) it shall include one or more trigger definitions which:
 - shall include an event with a value equal to the full name of a notification in the VnfIndicator interface definition of the VNF node where the policy applies;
 - may include a condition definition which can assert the value of vnf indicator attributes and other node attributes using arbitrary AND and OR combinations of the individual assertions;
 - may include an action invoking one or multiple operations of the Vnflcm interface;

- b) the target shall be set to the node template to which the policy applies, i.e. to the node template of the VNF specific type present in the topology template that represents a particular deployment flavour.

6.10.15 `tosca.policies.nfv.VnfPackageChange`

6.10.15.1 Description

The `VnfPackageChange` type is a policy type specifying the processes and rules to be used for performing the resource related tasks, to change VNF instance to a different VNF Package (destination package) as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.15.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.15.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VnfPackageChange</code>
Type Qualified Name	<code>toscanfv.VnfPackageChange</code>
Type URI	<code>tosca.policies.nfv.VnfPackageChange</code>

6.10.15.2 Properties

The properties of the `VnfPackageChange` policy type shall comply with the provisions set out in table 6.10.15.2-1.

Table 6.10.15.2-1: Properties

Name	Required	Type	Constraints	Description
<code>selectors</code>	yes	list of <code>tosca.datatypes.nfv.VnfPackageChangeSelector</code>		Information to identify the source and destination VNFD for the change, and the related deployment flavours. See note 1.
<code>modification_qualifier</code>	yes	string	up, down	Specifies the type of modification resulting from transitioning from <code>srcVnfId</code> to <code>dstVnfId</code> . The possible values are UP indicating that the destination VNF version is newer than the source version, DOWN indicating that the destination VNF version is older than the source version.
<code>additional_modification_description</code>	no	string		Additional information to qualify further the change between the two versions.
<code>component_mappings</code>	no	list of <code>tosca.datatypes.nfv.VnfPackageChangeComponentMapping</code>		Mapping information related to identifiers of components in source VNFD and destination VNFD that concern to the change process.
<code>destination_flavour_id</code>	yes	string		Identifies the deployment flavour in the destination VNF package for which this change applies. The flavour ID is defined in the destination VNF package.
<code>actions</code>	no	list of string		List of applicable supported LCM coordination action names specified in this VNFD (<code>action_name</code>) as a TOSCA policy of a type derived from <code>tosca.policies.nfv.LcmCoordinationAction</code> .

Name	Required	Type	Constraints	Description
referenced_coordination_actions	no	list of string		List of names of coordination actions not specified within this VNFD as a TOSCA policy of a type derived from <code>tosca.policies.nfv.LcmCoordinationAction</code> . See note 2.
upgrade_type	no	string	Valid values: See YAML definition constraints	Supported upgrade type when change the current VNF Package on which a VNF instance is based.
NOTE 1: If selectors include multiple entries, all other properties apart from the selectors in the <code>VnfPackageChange</code> policy define the package change that is applicable to any change path defined by any of the entries in the selectors. If change paths require e.g. different <code>component_mappings</code> they shall be described by different <code>VnfPackageChange</code> policies. Each triplet (<code>source_descriptor_id</code>, <code>destination_descriptor_id</code>, <code>source_flavour_id</code>) represented by one entry in the selectors shall not appear more than once in a VNFD. If a triplet occurs in both, source and destination package, the content of the identified <code>VnfPackageChange</code> policies (apart from the selectors) shall be the same. NOTE 2: Naming conventions for coordination names are defined in names as specified in clause 10.7 of ETSI GS NFV-SOL 002 [22].				

6.10.15.3 Definition

The syntax of the `VnfPackageChange` policy type shall comply with the following definition:

```
tosca.policies.nfv.VnfPackageChange:
  derived_from: toska.policies.Root
  description: policy type specifying the processes and rules to be used for
performing the resource related tasks, to change VNF instance to a different VNF
Package (destination package)
  properties:
    selector:
      type: list
      entry_schema:
        type: toska.datatypes.nfv.VnfPackageChangeSelector
        description: Information to identify the source and destination VNFD for
the change, and the related deployment flavours.
        required: true
        constraints:
          - min_length: 1
        modification_qualifier:
          type: string
          description: Specifies the type of modification resulting from
transitioning from srcVnfdId to dstVnfdId. The possible values are UP indicating
that the destination VNF version is newer than the source version, DOWN indicating
that the destination VNF version is older than the source version.
          constraints: [ valid_values: [ up, down ] ]
          required: true
        additional_modification_description:
          type: string
          description: Additional information to qualify further the change between
the two versions.
          required: false
        component_mappings:
          type: list
          entry_schema:
            type: toska.datatypes.nfv.VnfPackageChangeComponentMapping
            description: Mapping information related to identifiers of components in
source VNFD and destination VNFD that concern to the change process.
            required: false
```

```

    destination_flavour_id:
      type: string
      description: Identifies the deployment flavour in the destination VNF
package for which this change applies. The flavour ID is defined in the
destination VNF package.
      required: true
    actions:
      type: list
      description: List of applicable supported LCM coordination action names
(action_name) specified in this VNFD as a TOSCA policy of a type derived from
tosca.policies.nfv.LcmCoordinationAction.
      required: false
      entry_schema:
        type: string
    referenced_coordination_actions:
      type: list
      description: List of names of coordination actions not specified within
this VNFD as a TOSCA policy of a type derived from
tosca.policies.nfv.LcmCoordinationAction.
      required: false
      entry_schema:
        type: string
    upgrade_type:
      type: string
      description: Supported upgrade type when change the current VNF Package on
which a VNF instance is based.
      required: false
      constraints:
        - valid_values: [ rolling_upgrade, blue_green ]
    targets: [ tosca.nodes.nfv.VNF ]

```

6.10.15.4 Additional Requirements

The VnfPackageChange specific type policy shall have exactly one trigger with an event and an action.

The event value shall be set to change_current_package_notification notification of Vnflcm interface.

The target shall be set to the node template to which the policy applies, i.e. to the node template of the VNF specific type present in the topology template that represents a particular deployment flavour.

The action value shall be set to either change_current_package operation on the Vnflcm (in case the same LCM script or no LCM script with the same set of "additionalParams" or no "additionalParams" is suitable for all change paths) or one of the VNF-specific operations on the ChangeCurrentVnfPackage interface (in case different change paths require different LCM scripts or no LCM script potentially with different sets of "additionalParams" or no "additionalParams").

The policy shall be applied when the actual values of the descriptor_id and flavour_id of the source VNF type and the descriptor_id of the destination VNF type match the source_descriptor_id, destination_descriptor_id and source_flavour_id properties, respectively, of the selector in the policy.

VNF-specific coordination actions shall be declared with their parameters in data types derived from tosca.datatypes.nfv.VnfLcmOpCoord (see clause 6.2.67), and an interface_name property of the tosca.policies.nfv.SupportedVnfInterface set to "vnf_lcm_coordination" shall be specified in the related deployment flavour to signal that this interface is exposed by the VNF.

NOTE: During and after the VNF Package change all information related to any other deployment flavours in the source VNFD than the source flavour is no longer applicable.

6.10.15.5 Example

The following example template illustrates a VNFD which is supporting VNF package change info via using VnfPackageChange policy, additional_parameters input for each case of the selector and VNF specific interface (tosca.interfaces.nfv.ChangeCurrentVnfPackage), LCM coordination actions, and other data types. A policy definition of change_from_version_1 and change_to_version_1 has a trigger. The action will be taken when the conditions of the selector values are satisfied. The policy has an event associated to the notification of toasca.interfaces.nfv.Vnflcm.change_current_package_notification.

```
tosca_definitions_version: toasca_simple_yaml_1_3

imports:
..

node_types:
  MyCompany.SunshineDB.2_0.2_0:
    derived_from: toasca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: ebc68e34-0cfa-40ba-8b45-9caa31f9dcb5 ]
        default: ebc68e34-0cfa-40ba-8b45-9caa31f9dcb5
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: SunshineDB ]
        default: SunshineDB
      software_version:
        type: string
        constraints: [ equal: '2.0' ]
        default: '2.0'
      descriptor_version:
        type: string
        constraints: [ equal: '2.0' ]
        default: '2.0'
      flavour_id:
        type: string
        constraints: [ equal: default ]
        default: default
    ..
  interfaces:
    MyCompanyChangeCurrentVnfPackage:
      type: MyCompany.interfaces.nfv.ChangeCurrentVnfPackage

interface_types:
  MyCompany.interfaces.nfv.ChangeCurrentVnfPackage
    derived_from: toasca.interfaces.nfv.ChangeCurrentVnfPackage
    operations:
      change_from_version_1:
        description: operation for change from version 1 to 2
        inputs:
          additional_parameters:
            type:
MyCompany.datatypes.nfv.VnfChangeFromVersion1AdditionalParameters
      change_to_version_1:
        description: operation for change from version 2 to 1
        inputs:
          additional_parameters:
```

```

        type: MyCompany.datatypes.nfv.VnfChangeToVersion1AdditionalParameters
data_types:
  ..

  MyCompany.datatypes.nfv.VnfChangeFromVersion1AdditionalParameters:
    derived_from: tosca.datatypes.nfv.VnfOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
      parameter_2:
        type: string

  MyCompany.datatypes.nfv.VnfChangeToVersion1AdditionalParameters:
    derived_from: tosca.datatypes.nfv.VnfOperationAdditionalParameters
    properties:
      parameter_3:
        type: string
      parameter_4:
        type: string

  MyCompany.datatypes.nfv.VnfChangeFromVersion1OpCoord:
    derived_from: tosca.datatypes.nfv.VnfLcmOpCoord
    properties:
      description:
        type: string
        required: true
        constraints: [ equal: 'MyCompany vnf change from version 1 operation
coordination information' ]
        default: 'MyCompany vnf change from version 1 operation coordination
information'
      endpoint_type:
        type: string
        required: true
        constraints: [ equal: 'vnf' ]
        default: 'vnf'
      coordination_stage:
        type: string
        required: true
        constraints: [ equal: 'start' ]
        default: 'start'
      input_parameters:
        type: MyCompany.datatypes.nfv.VnfChangeFromVersion1InputOpCoordParams
        required: true
      output_parameters:
        type: MyCompany.datatypes.nfv.VnfChangeGeneralOutputOpCoordParams
        required: true

  MyCompany.datatypes.nfv.VnfChangeToVersion1OpCoord:
    derived_from: tosca.datatypes.nfv.VnfLcmOpCoord
    properties:
      description:
        type: string
        required: true
        constraints: [ equal: 'MyCompany vnf change to version 1 operation
coordination information' ]
        default: 'MyCompany vnf change to version 1 operation coordination
information'
      endpoint_type:
        type: string
        required: true

```

```

        constraints: [ equal: 'vnf' ]
        default: 'vnf'
    coordination_stage:
        type: string
        required: true
        constraints: [ equal: 'start' ]
        default: 'start'
    input_parameters:
        type: MyCompany.datatypes.nfv.VnfChangeToVersion1InputOpCoordParams
        required: true
    output_parameters:
        type: MyCompany.datatypes.nfv.VnfChangeGeneralOutputOpCoordParams
        required: true

MyCompany.datatypes.nfv.InstantiateOpCoord:
    derived_from: tosca.datatypes.nfv.VnfLcmOpCoord
    properties:
        description:
            type: string
            required: true
            constraints: [ equal: 'MyCompany vnf instantiate operation coordination
information' ]
            default: 'MyCompany vnf instantiate operation coordination information'
        endpoint_type:
            type: string
            required: true
            constraints: [ equal: 'vnf' ]
            default: 'vnf'
        coordination_stage:
            type: string
            required: true
            constraints: [ equal: 'start' ]
            default: 'start'
        input_parameters:
            type: MyCompany.datatypes.nfv.InstantiateInputOpCoordParams
            required: true

MyCompany.datatypes.nfv.InstantiateInputOpCoordParams:
    derived_from: tosca.datatypes.nfv.InputOpCoordParams
    properties:
        data1:
            type: string
            required: true
            constraints: [ equal: 'value_1' ]
            default: 'value_1'

MyCompany.datatypes.nfv.VnfChangeFromVersion1InputOpCoordParams:
    derived_from: tosca.datatypes.nfv.InputOpCoordParams
    properties:
        data2:
            type: string
            required: true
            constraints: [ equal: 'value_2' ]
            default: 'value_2'

MyCompany.datatypes.nfv.VnfChangeToVersion1InputOpCoordParams:
    derived_from: tosca.datatypes.nfv.InputOpCoordParams
    properties:
        data3:
            type: string
            required: true

```

```

        constraints: [ equal: 'value_3' ]
        default: 'value_3'

MyCompany.datatypes.nfv.VnfChangeGeneralOutputOpCoordParams:
  derived_from: tosca.datatypes.nfv.OutputOpCoordParams
  properties:
    data4:
      type: string
      required: true
      constraints: [ equal: 'value_4' ]
      default: 'value_4'

..

policy_types:

  tosca.policies.nfv.LcmCoordinationAction.InstantiateHelloHandshake:
    derived_from: tosca.policies.nfv.LcmCoordinationAction
    properties:
      action:
        type: MyCompany.datatypes.nfv.InstantiateOpCoord

  tosca.policies.nfv.VnfChangeFromVersion1Action1OpCoord:
    derived_from: tosca.policies.nfv.LcmCoordinationAction
    properties:
      action:
        type: MyCompany.datatypes.nfv.VnfChangeFromVersion1OpCoord

  tosca.policies.nfv.VnfChangeToVersion1Action1OpCoord:
    derived_from: tosca.policies.nfv.LcmCoordinationAction
    properties:
      action:
        type: MyCompany.datatypes.nfv.VnfChangeToVersion1OpCoord

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.2_0.2_0
    ..

node_templates:
  sunshine_db:
    type: MyCompany.SunshineDB.2_0.2_0
    ..
  interfaces:
    MyCompanyChangeCurrentVnfPackage:
      operations:
        ..
        change_from_version_1:
          implementation: change-from-version-1.workbook.mistral.yaml
        change_to_version_1:
          implementation: change-to-version-1.workbook.mistral.yaml

server:
  type: MyCompany.nodes.nfv.Vdu.Aux
  ..

volume:
  type: tosca.nodes.nfv.Vdu.VirtualBlockStorage
  ..

```



```

server_internal_cp:
  type: toska.nodes.nfv.VduCp
  ..

internal_vl:
  type: toska.nodes.nfv.VnfVirtualLink
  ..

policies:
  ..

  - instantiation_levels:
    type: toska.policies.nfv.InstantiationLevels
    properties:
      levels:
        single:
          description: ..
        quadruple:
          description: ..
      default_level: single
    ..

  - change_from_version_1_0 and 1_1:
    type: toska.policies.nfv.VnfPackageChange
    properties:
      selectors:
        - source_descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
          destination_descriptor_id: ebc68e34-0cfa-40ba-8b45-9caa31f9dcb5
          source_flavour_id: simple
        - source_descriptor_id: a9c82e92-0007-11ec-9a03-0242ac130003
          destination_descriptor_id: ebc68e34-0cfa-40ba-8b45-9caa31f9dcb5
          source_flavour_id: simple
      modification_qualifier: up
      additional_modification_description: ..
      component_mappings:
        - component_type: vdu
          source_id: dbBackend
          destination_id: server
          description: ..
        - component_type: virtual_storage
          source_id: mariaDbStorage
          destination_id: volume
          description: ..
        - component_type: instantiation_level
          source_id: single
          destination_id: default
          description: ..

      destination_flavour_id: default
      actions:
        - vnd.mycompany.VnfChangeFromVersion1Action1
      triggers:
        change_from_version_1:
          event:
tosca.interfaces.nfv.Vnflcm.change_current_package_notification
          action:
            - call_operation:
MyCompanyChangeCurrentVnfPackage.change_from_version_1
          targets: [sunshine_db]

  - change_to_version_1_0:

```

```

type: toska.policies.nfv.VnfPackageChange
properties:
  selectors:
    - source_descriptor_id: ebc68e34-0cfa-40ba-8b45-9caa31f9dcb5
      destination_descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      source_flavour_id: default
  modification_qualifier: down
  additional_modification_description: ..
  component_mappings:
    - component_type: vdu
      source_id: server
      destination_id: dbBackend
      description: ..
    - component_type: virtual_storage
      source_id: volume
      destination_id: mariaDbStorage
      description: ..
    - component_type: instantiation_level
      source_id: default
      destination_id: single
      description: ..

  destination_flavour_id: simple
  actions:
    - vnd.mycompany.VnfChangeToVersion1Action1
  triggers:
    change_to_version_1:
      event:
tosca.interfaces.nfv.Vnflcm.change_current_package_notification
      action:
        - call_operation:
MyCompanyChangeCurrentVnfPackage.change_to_version_1
      targets: [sunshine_db]

- instantiate_HelloHandshake:
  type: toska.policies.nfv.LcmCoordinationAction.InstantiateHelloHandshake
  properties:
    action_name: vnd.mycompany.hellohandshake

- instantiate_OpCoord:
  type: toska.policies.nfv.LcmCoordinationsForLcmOperation
  properties:
    vnf_lcm_operation: instantiate
  actions:
    - vnd.mycompany.hellohandshake

- terminate_OpCoord:
  type: toska.policies.nfv.LcmCoordinationsForLcmOperation
  properties:
    vnf_lcm_operation: terminate
    referenced_coordination_actions:
      - etsi.nfv.take-vnf-out-of-service

- VnfChangeFromVersion1Action1_OpCoord:
  type: toska.policies.nfv.LcmCoordinationAction
  properties:
    action_name: vnd.mycompany.VnfChangeFromVersion1Action1

- VnfChangeToVersion1Action1_OpCoord:
  type: toska.policies.nfv.LcmCoordinationAction
  properties:

```

```
action_name: vnd.mycompany.VnfChangeToVersion1Action1
```

The following example template illustrates a VNFD which is supporting VNF package change info via using VnfPackageChange policy, without additional_parameters input and Vnflcm interface operation. A policy definition of change_from_version_1 and change_to_version_1 has a trigger. The action will be taken when the conditions of the selector values are satisfied. The policy has an event associated to the notification of tosca.interfaces.nfv.Vnflcm.change_current_package_notification.

```
tosca_definitions_version: tosca_simple_yaml_1_3

imports:
..

node_types:
  MyCompany.SunshineDB.2_0.2_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: ebc68e34-0cfa-40ba-8b45-9caa31f9dcb5 ]
        default: ebc68e34-0cfa-40ba-8b45-9caa31f9dcb5
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: SunshineDB ]
        default: SunshineDB
      software_version:
        type: string
        constraints: [ equal: '2.0' ]
        default: '2.0'
      descriptor_version:
        type: string
        constraints: [ equal: '2.0' ]
        default: '2.0'
      flavour_id:
        type: string
        constraints: [ equal: default ]
        default: default
    ..
  interfaces:
    MyCompanyVnflcm:
      type: tosca.interfaces.nfv.Vnflcm

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.2_0.2_0
    ..

  node_templates:
    sunshine_db:
      type: MyCompany.SunshineDB.2_0.2_0
      ..
    interfaces:
      Vnflcm:
        operations:
          ..
```

```

    change_current_package:

server:
  type: MyCompany.nodes.nfv.Vdu.Aux
  ..

volume:
  type: tosca.nodes.nfv.Vdu.VirtualBlockStorage
  ..

server_internal_cp:
  type: tosca.nodes.nfv.VduCp
  ..

internal_vl:
  type: tosca.nodes.nfv.VnfVirtualLink
  ..

policies:
  ..

  - instantiation_levels:
      type: tosca.policies.nfv.InstantiationLevels
      properties:
        levels:
          single:
            description: ..
          quadruple:
            description: ..
        default_level: single
      ..

  - change_from_version_1_0_and_1_1:
      type: tosca.policies.nfv.VnfPackageChange
      properties:
        selectors:
          - source_descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
            destination_descriptor_id: ebc68e34-0cfa-40ba-8b45-9caa31f9dc5
            source_flavour_id: simple
          - source_descriptor_id: a9c82e92-0007-11ec-9a03-0242ac130003
            destination_descriptor_id: ebc68e34-0cfa-40ba-8b45-9caa31f9dc5
            source_flavour_id: simple
        modification_qualifier: up
        additional_modification_description: ..
        component_mappings:
          - component_type: vdu
            source_id: dbBackend
            destination_id: server
            description: ..
          - component_type: virtual_storage
            source_id: mariaDbStorage
            destination_id: volume
            description: ..
          - component_type: instantiation_level
            source_id: single
            destination_id: default
            description: ..

        destination_flavour_id: default
      triggers:
        change_from_version_1:

```

```

        event:
tosca.interfaces.nfv.Vnflcm.change_current_package_notification
        action:
            - call_operation: Vnflcm.change_current_package
        targets: [ sunshine_db ]

- change_to_version_1_0:
    type: toska.policies.nfv.VnfPackageChange
    properties:
        selectors:
            - source_descriptor_id: ebc68e34-0cfa-40ba-8b45-9caa31f9dcb5
              destination_descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
              source_flavour_id: default
        modification_qualifier: down
        additional_modification_description: ..
        component_mappings:
            - component_type: vdu
              source_id: server
              destination_id: dbBackend
              description: ..
            - component_type: virtual_storage
              source_id: volume
              destination_id: mariaDbStorage
              description: ..
            - component_type: instantiation_level
              source_id: default
              destination_id: single
              description: ..

        destination_flavour_id: simple
    triggers:
        change_to_version_1:
            event:
tosca.interfaces.nfv.Vnflcm.change_current_package_notification
            action:
                - call_operation: Vnflcm.change_current_package
            targets: [ sunshine_db ]

```

6.10.16 toska.policies.nfv.LcmCoordinationAction

6.10.16.1 Description

The LcmCoordinationAction type is a base type for deriving policy types which describe the LCM coordination actions supported by a VNF and/or expected to be supported by its EM. This policy concerns the whole VNF (deployment flavour) represented by the topology_template and thus has no explicit target list.

Table 6.10.16.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.16.1-1: Type name, shorthand, and URI

Shorthand Name	LcmCoordinationAction
Type Qualified Name	toscanfv:LcmCoordinationAction
Type URI	tosca.policies.nfv.LcmCoordinationAction

6.10.16.2 Properties

The properties of the LcmCoordinationAction policy type shall comply with the provisions set out in table 6.10.16.2-1.

Table 6.10.16.2-1: Properties

Name	Required	Type	Constraints	Description
action_name	yes	string		Coordination action name. See note 1.
action	yes	tosca.datatypes.nfv.VnfLcmOpCoord		Describes a set of information needed for coordination action in the VNF LCM operation. See note 2.
NOTE 1: The action_name shall comply with naming conventions as specified in in clause 10.7 of ETSI GS NFV-SOL 002 [22].				
NOTE 2: Represents a place holder for specifying actions of a VNF-specific type derived from toska.datatypes.nfv.VnfLcmOpCoord.				

6.10.16.3 Definition

The syntax of the LcmCoordinationAction policy type shall comply with the following definition:

```
tosca.policies.nfv.LcmCoordinationAction:
  derived_from: toska.policies.Root
  description: The LcmCoordinationAction type is a policy type representing the LCM
  coordination actions supported by a VNF and/or expected to be supported by its EM for
  a particular VNF LCM operation. This policy concerns the whole VNF (deployment
  flavour) represented by the topology_template and thus has no explicit target list.
  properties:
    action_name:
      type: string
      description: Coordination action name.
      required: true
    action: #represents a place holder for specifying actions of a VNF-specific
  type derived from toska.datatypes.nfv.VnfLcmOpCoord
      type: toska.datatypes.nfv.VnfLcmOpCoord
      description: Describes a set of information needed for coordination action in
  the VNF LCM operation.
      required: true
```

6.10.16.4 Additional Requirements

A LcmCoordinationAction policy shall contain only one LCM coordination action. The VNFD shall define a policy type derived from toska.policies.nfv.LcmCoordinationAction for each of VNF LCM coordination action. The LCM coordination action specific derived policy type shall contain a property 'action' of a datatype derived from toska.datatypes.nfv.VnfLcmCoord.

The interface_name property of the toska.policies.nfv.SupportedVnfInterface set to "vnf_lcm_coordination" shall be specified in the related deployment flavour to signal that this interface is exposed by the VNF.

6.10.17 toska.policies.nfv.LcmCoordinationsForLcmOperation

6.10.17.1 Description

The LcmCoordinationsForLcmOperation type is a policy type representing supported LCM coordination actions associated to a VNF LCM operation. This policy concerns the whole VNF (deployment flavour) represented by the topology_template and thus has no explicit target list. Table 6.10.17.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.17.1-1: Type name, shorthand, and URI

Shorthand Name	LcmCoordinationsForLcmOperation
Type Qualified Name	toscafv:LcmCoordinationsForLcmOperation
Type URI	tosca.policies.nfv.LcmCoordinationsForLcmOperation

6.10.17.2 Properties

The properties of the LcmCoordinationsForLcmOperation policy type shall comply with the provisions set out in table 6.10.17.2-1.

Table 6.10.17.2-1: Properties

Name	Required	Type	Constraints	Description
vnf_lcm_operation	yes	string	Valid values: See YAML definition constraints	The VNF LCM operation the LCM coordination actions are associated with.
actions	no	list of string		List of applicable supported LCM coordination action names (action_name) specified in this VNFD as a TOSCA policy of a type derived from toasca.policies.nfv.LcmCoordinationAction. See note 1 and note 2.
referenced_coordination_actions	no	list of string		List of names of coordination actions not specified within this VNFD as a TOSCA policy of a type derived from toasca.policies.nfv.LcmCoordinationAction. See note 1 and note 2.
NOTE 1: At least one of the actions and referenced_coordination_actions properties shall be present.				
NOTE 2: Naming conventions for coordination names are defined in names as specified in clause 10.7 of ETSI GS NFV-SOL 002 [22].				

6.10.17.3 Definition

The syntax of the LcmCoordinationsForLcmOperation policy type shall comply with the following definition:

```
tosca.policies.nfv.LcmCoordinationsForLcmOperation:
  derived_from: toska.policies.Root
  description: The LcmCoordinationsForLcmOperation type is a policy type
representing supported LCM coordination actions associated to a VNF LCM operation.
This policy concerns the whole VNF (deployment flavour) represented by the
topology_template and thus has no explicit target list.
  properties:
    vnf_lcm_operation:
      type: string
      description: The VNF LCM operation the LCM coordination actions are associated
with.
      required: true
      constraints:
        - valid_values: [instantiate, scale, scale_to_level, change_flavour,
terminate, heal, operate, change_ext_conn, modify_info, create_snapshot,
revert_to_snapshot ]
      actions:
        type: list
        description: List of applicable supported LCM coordination action names
(action_name) specified in this VNFD as a TOSCA policy of a type derived from
tosca.policies.nfv.LcmCoordinationAction.
        required: false
        entry_schema:
          type: string
      referenced_coordination_actions:
        type: list
        description: List of names of coordination actions not specified within this
VNFD as a TOSCA policy of a type derived from
tosca.policies.nfv.LcmCoordinationAction.
        required: false
        entry_schema:
          type: string
```

6.10.17.4 Additional Requirements

The VNFD may define one LcmCoordinationsForLcmOperation policy per each VNF LCM operation (vnf_lcm_operation).

VNF lifecycle management coordination actions are invoked by the VNFM towards the VNF instance or towards operation supporting management systems (e.g. EM). They can be standardized or VNF-specific. To distinguish between both categories, clause 10.7 of ETSI GS NFV-SOL 002 [22] defines namespaces for the values of the coordination action names.

6.10.18 toska.policies.nfv.VipCpScalingAspectDeltas

6.10.18.1 Description

The VipCpScalingAspectDeltas type is a policy type representing the VipCp detail of an aspect deltas used for horizontal scaling, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.18.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.18.1-1: Type name, shorthand, and URI

Shorthand Name	VipCpScalingAspectDeltas
Type Qualified Name	tosca.nfv.VipCpScalingAspectDeltas
Type URI	tosca.policies.nfv.VipCpScalingAspectDeltas

6.10.18.2 Properties

The properties of the VipCpScalingAspectDeltas policy type shall comply with the provisions set out in table 6.10.18.2-1.

Table 6.10.18.2-1: Properties

Name	Required	Type	Constraints	Description
aspect	yes	string		Represents the scaling aspect to which this policy applies.
deltas	yes	map of tosca.datatypes .nfv.VipCpLevel		Describes the VipCp scaling deltas to be applied for every scaling steps of a particular aspect.

6.10.18.3 Definition

The syntax of the VipCpScalingAspectDeltas policy type shall comply with the following definition:

```
tosca.policies.nfv.VipCpScalingAspectDeltas:
  derived_from: toska.policies.Root
  description: The VipCpScalingAspectDeltas type is a policy type representing the
  VipCp detail of an aspect deltas used for horizontal scaling, as defined in ETSI GS
  NFV-IFA 011
  properties:
    aspect:
      type: string
      description: Represents the scaling aspect to which this policy applies
      required: true
    deltas:
      type: map # key: scalingDeltaId
      description: Describes the VipCp scaling deltas to be applied for every
scaling steps of a particular aspect.
      required: true
      entry_schema:
        type: toska.datatypes.nfv.VipCpLevel
      constraints:
        - min_length: 1
  targets: [ toska.nodes.nfv.VipCp ]
```

6.10.18.4 Additional Requirements

In the case of "uniform aspect", the deltas properties shall have only one entry.

If a policy definition of this type is included in a service template, a policy definition of the type VipCpInitialDelta defined in clause 6.10.19 of the present document shall also be included with the same target.

6.10.18.5 Examples

None.

6.10.19 tosca.policies.nfv.VipCpInitialDelta

6.10.19.1 Description

The VipCpInitialDelta type is a policy type representing the VipCp detail of an initial delta used for horizontal scaling, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.19.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.19.1-1: Type name, shorthand, and URI

Shorthand Name	VipCpInitialDelta
Type Qualified Name	toscanfv:VipCpInitialDelta
Type URI	tosca.policies.nfv.VipCpInitialDelta

6.10.19.2 Properties

The properties of the VipCpInitialDelta policy type shall comply with the provisions set out in table 6.10.19.2-1.

Table 6.10.19.2-1: Properties

Name	Required	Type	Constraints	Description
initial_delta:	yes	tosca.datatypes.nfv.VipCpLevel		Represents the initial minimum size of the VNF.

6.10.19.3 Definition

The syntax of the VipCpInitialDelta policy type shall comply with the following definition:

```
tosca.policies.nfv.VipCpInitialDelta:
  derived_from: tosca.policies.Root
  description: The VipCpInitialDelta type is a policy type representing the VipCp
detail of an initial delta used for horizontal scaling, as defined in ETSI GS NFV-IFA
011
  properties:
    initial_delta:
      type: tosca.datatypes.nfv.VipCpLevel
      description: Represents the initial minimum size of the VNF.
      required: true
  targets: [ tosca.nodes.nfv.VipCp ]
```

6.10.19.4 Additional Requirements

None.

6.10.19.5 Examples

None.

6.10.20 tosca.policies.nfv.VipCpInstantiationLevels

6.10.20.1 Description

The VipCpInstantiationLevels type is a policy type representing all the instantiation levels of resources to be instantiated within a deployment flavour in term of the number of VipCp instances to be created from each VipCp. as defined in ETSI GS NFV-IFA 011 [1].

Table 6.10.20.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.20.1-1: Type name, shorthand, and URI

Shorthand Name	VipCpInstantiationLevels
Type Qualified Name	toscanfv:VipCpInstantiationLevels
Type URI	tosca.policies.nfv.VipCpInstantiationLevels

6.10.20.2 Properties

The properties of the VipCpInstantiationLevels policy type shall comply with the provisions set out in table 6.10.20.2-1.

Table 6.10.20.2-1: Properties

Name	Required	Type	Constraints	Description
levels	yes	map of tosca.datatypes.nfv. VipCpLevel		Describes the VipCp levels of resources that can be used to instantiate the VNF using this flavour.

6.10.20.3 Definition

The syntax of the VipCpInstantiationLevels policy type shall comply with the following definition:

```
tosca.policies.nfv.VipCpInstantiationLevels:
  derived_from: toscanfv.Root
  description: The VipCpInstantiationLevels type is a policy type representing all
the instantiation levels of resources to be instantiated within a deployment flavour
in term of the number of VipCp instances to be created from each VipCp as defined in
ETSI GS NFV-IFA 011.
  properties:
    levels:
      type: map # key: levelId
      description: Describes the VipCp levels of resources that can be used to
instantiate the VNF using this flavour
      required: true
      entry_schema:
        type: toscanfv.datatypes.VipCpLevel
      constraints:
        - min_length: 1
  targets: [ toscanfv.nodes.VipCp ]
```

6.10.20.4 Additional Requirements

A VipCpInstantiationLevels policy shall contain an entry for each instantiation level (and only for them) defined in the InstantiationLevels policy.

6.10.21 toscanfv.PowerProfiles

6.10.21.1 Description

The PowerProfiles type is a policy type representing all the VNF power profiles within a deployment flavour as defined in ETSI GS NFV-IFA 011 [1]. This policy concerns the whole VNF (deployment flavour) represented by the topology_template and thus has no explicit target list. Table 6.10.21.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 6.10.21.1-1: Type name, shorthand, and URI

Shorthand Name	PowerProfiles
Type Qualified Name	toscanfv:PowerProfiles
Type URI	tosca.policies.nfv.PowerProfiles

6.10.21.2 Properties

The properties of the PowerProfiles policy type shall comply with the provisions set out in table 6.10.21.2-1.

Table 6.10.21.2-1: Properties

Name	Required	Type	Constraints	Description
profiles	yes	map of tosca.datatypes.nfv.PowerProfile		Specifies VNF power profiles for this deployment flavour.

6.10.21.3 Definition

The syntax of the PowerProfiles policy type shall comply with the following definition:

```

tosca.policies.nfv.PowerProfiles:
  derived_from: toscanfv.Root
  description: The PowerProfiles type is a policy type representing all the VNF
power profiles within a deployment flavour as defined in ETSI GS NFV-IFA 011.
  properties:
    profiles:
      type: map # key: powerProfileId
      description: Specifies VNF power profiles for this deployment flavour.
      required: true
      entry_schema:
        type: toscanfv.datatypes.nfv.PowerProfile

```

6.11 VNFD TOSCA service template design

6.11.1 General

The TOSCA service template design for a VNFD in the general case uses two levels of service templates as described in clause 6.11.2. In this design, the top level contains an abstract VNF node template, i.e. without an implementation of the creation operation and is therefore substituted by one of the lower level service templates. This design is applicable regardless of whether the VNF has one or multiple deployment flavours.

In the particular case of a VNF with only one deployment flavour there is an alternative design which is described in clause 6.11.3 and which uses only one service template.

6.11.2 Single or multiple deployment flavour design with two levels of service templates

VNFD shall be implemented as one top-level service template and one or multiple lower level service templates, where each lower level service template represents a deployment flavour. A separate YAML file with a VNF specific node type definition which shall be derived from toscanfv.VNF node type as defined in clause 6.8.1 shall be provided and is also considered as a part of a VNFD. The top level service template shall be the main entry point of the VNF package as specified in ETSI GS NFV-SOL 004 [i.6], i.e. the Entry-definitions file. The file names of all the lower service templates shall be declared as the value of the Other-Definitions key as specified in TOSCA-Simple-Profile-YAML-v1.3 [20] in the TOSCA.meta file of the VNF package.

See clause A.2 for an example of VNFD design with multiple deployment flavours.

The top level service template shall comply with TOSCA-Simple-Profile-YAML-v1.3 [20] and shall include:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.2;
- b) an import statement referencing a yaml file which contains a VNF specific node type definition;
- c) optionally, import statements referencing additional VNF-specific files containing only type definitions used by this TOSCA service template; and
- d) a topology template with a node template of the VNF specific node type, which:
 - shall include the flavour_id and other properties that are marked as required but do not have a default value in the VNF specific node type definition;
 - shall include the requirements as defined in clause 6.8.1;
 - may include other properties specified in the VNF specific node type definition, excluding the vnf_profile property;
 - may include a substitute directive;
- e) optionally, metadata and dsl definitions as specified in section 3.10 of TOSCA-Simple-Profile-YAML-v1.3 [20].

Irrespective of the presence or absence of the substitute directive, the deployment and lifecycle management of instances of this VNF node type shall be done by means of substitution by any of the lower level service templates as declared in the Other-Definitions of the TOSCA.meta file in the VNF package. The VNFD consumer shall silently ignore the substitute directive if explicit directives are not supported.

The lower level service template is an implementable TOSCA service template for the deployment of a specific deployment flavour. The lower level service template shall comply with TOSCA-Simple-Profile-YAML-v1.3 [20] and shall include:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.2;
- b) an import statement referencing a yaml file which contains a VNF specific node type definition which shall be derived from tosa.nodes.nfv.VNF node type as defined in clause 6.8.1;
- c) optionally, additional VNF-specific type definitions and import statements referencing additional VNF-specific files containing type definitions used by this TOSCA service template; and
- d) a topology template describing the internal topology of the VNF with:
 - substitution_mappings indicating:
 - the same node type as defined in the VNF specific node type definition service template;
 - a flavour_id property and its value as defined in substitution_filter which identifies the DF corresponding to this low level template within the VNFD;

NOTE 1: Starting with version 3.3.1 of the present document, the property_mapping grammar was changed to support substitution_filter. For backward compatibility, TOSCA-Simple-Profile-YAML-v1.3 [20], clause 3.8.8.3 specifies the provisions for handling the previous grammar. Support of the Release 2 property_mapping grammar can be removed in subsequent versions of the present document.

- the mapping of the virtual_link requirements on external connection points;
- a node template referencing the VNF specific node type, implementations of the operations of the LCM interface to be executed by the VNFM, if applicable; and
- additional node templates of type Vdu.Compute (or a derived node type), Vdu.VirtualBlockStorage, Vdu.VirtualObjectStorage, Vdu.VirtualFileStorage VduCp, etc. that define the topology and composition of the VNF flavour, the dependency requirements as defined in TOSCA-Simple-Profile-YAML-v1.3 [20] may be used between different node templates of type

Vdu.Compute (or a derived node type) to specify the order in which instances of the VNFCs have to be created;

- additional group definitions, policy definitions and parameter definitions if applicable;
- e) optionally, metadata and dsl definitions as specified in section 3.10 of TOSCA-Simple-Profile-YAML-v1.3 [20].

NOTE 2: The format and structure of a VNF package is defined in ETSI GS NFV-SOL 004 [i.6].

NOTE 3: All the imported type definition files as indicated either in the top level service template or in any of the lower level service template are considered as parts of a VNFD.

When the flavour_id of a VNF has been chosen (e.g. through an input parameter of a VNF instantiation request received by a VNFM) among the values included in the VNF node type imported into the top level service template, it is then used as the filter for selecting a particular lower level TOSCA service template inside the VNF package as described in TOSCA-Simple-Profile-YAML-v1.3 [20].

6.11.3 Single deployment flavour design with one service template

In case of the single deployment flavour scenario with one service template design, the VNFD shall use TOSCA-Simple-Profile-YAML-v1.3 [20] and shall include:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.2;
- b) either a VNF specific node type definition derived from the `tosca.nodes.nfv.VNF` node type, as defined in clause 6.8.1 or an import statement referencing a file that contains such definition;
- c) optionally, additional VNF-specific type definitions and import statements referencing additional VNF-specific files containing type definitions used by this TOSCA service template; and
- d) a topology template describing the internal topology of the VNF with:
 - `substitution_mappings` indicating the same VNF specific node type and the mapping of the `virtual_link` requirements on external connection points;
 - a node template of this VNF specific node type with the `flavour_id` and other properties and, if applicable, implementations of the operations of the LCM interface to be executed by the VNFM; and
 - additional node templates of type `Vdu.Compute` (or a derived node type), `Vdu.VirtualBlockStorage`, `Vdu.VirtualObjectStorage`, `Vdu.VirtualFileStorage`, `VduCp`, etc. that define the topology and composition of the VNF flavour, the dependency requirements as defined in TOSCA-Simple-Profile-YAML-v1.3 [20] may be used between different node templates of type `Vdu.Compute` (or a derived node type) to specify the order in which instances of the VNFCs have to be created;
 - additional group definitions, policy definitions and parameter definitions if applicable;
- e) optionally, metadata and dsl definitions as specified in section 3.10 of TOSCA-Simple-Profile-YAML-v1.3 [20].

See clause A.5 for an example of VNFD design with single deployment flavour.

NOTE 1: The service template is deployed stand-alone, i.e. without performing a substitution. However including the `substitution_mappings` rule indicates its ability to substitute a node template of the VNF specific node type, which may appear in an NSD.

NOTE 2: All the imported type definition files as indicated in the service template are considered as parts of a VNFD.

6.11.4 Package change (handling the Change current VNF Package request)

The Change current VNF Package operation, defined in ETSI GS NFV-SOL 002 [22] and ETSI GS NFV-SOL 003 [25], enables the NFVO to request the VNFM to change the current VNF Package, i.e. the VNF package on which a VNF instance is based, hence enabling VNF software modification. For such a request to be handled properly by the VNFM, the VNFD provides a policy of type `tosca.policies.nfv.VnfPackageChange` with the following characteristics:

- there may be as many instances of this policy type in the VNFD as change paths are supported;
- either the source or the destination of a change path shall be the current VNFD i.e. the one where the policy instance is present;
- a given instance of this policy type is triggered when the combination of the `source_descriptor_id`, `destination_descriptor_id`, `source_flavour_id` values in its selector property matches with the following:
 - `source_descriptor_id`: "vnfdId": attribute in `VnfInstance`
 - `destination_descriptor_id`: "vnfdId" attribute in `ChangeCurrentVnfPkgRequest`
 - `source_flavour_id`: "flavour" attribute in `VnfInstance`
- the operation designated by the trigger action in the policy definition is invoked upon receiving the Change current VNF Package request on the API (ETSI GS NFV-SOL 003 [25] or ETSI GS NFV-SOL 002 [22]);
- if "additionalParams" are to be received in the Change current VNF Package request on the API, then the input signature (`additional_parameters`) of the designated operation defines what "additionalParams" can be submitted as part of the operation request.

7 NSD TOSCA model

7.1 Introduction

The NSD information model specified by ETSI GS NFV-IFA 014 [2] is mapped to the TOSCA concepts. NSD occurrences are represented as TOSCA service templates, as defined in the TOSCA-Simple-Profile-YAML-v1.3 [20], to be used by the NFVO for managing the lifecycle of NS instances.

Table 7.1-1 shows an overview of the mapping between the main NSD information elements defined in ETSI GS NFV-IFA 014 [2] and TOSCA types defined in the present document. The definition of all TOSCA types for representing all information elements is described in the following clauses.

Table 7.1-1: Mapping of ETSI GS NFV-IFA 014 [2] information elements with TOSCA types

ETSI NFV Information Element ETSI GS NFV-IFA 014 [2]	TOSCA type	Derived from
NSD	<code>tosca.nodes.nfv.NS</code>	<code>tosca.nodes.Root</code>
Sapd	<code>tosca.nodes.nfv.Sap</code>	<code>tosca.nodes.Root</code>
NsVirtualLinkDesc	<code>tosca.nodes.nfv.NsVirtualLink</code>	<code>tosca.nodes.Root</code>
Pnfd	<code>tosca.nodes.nfv.PNF</code>	<code>tosca.nodes.Root</code>
Vnfd	<code>tosca.nodes.nfv.VNF</code>	<code>tosca.nodes.Root</code>
Vnffgd	<code>tosca.groups.nfv.VNFFG</code>	<code>tosca.groups.Root</code>
PaasServiceRequest	<code>tosca.nodes.nfv.NsPaasServiceRequest</code>	<code>tosca.nodes.Root</code>
PaasServiceProfile	<code>tosca.nodes.nfv.NsPaasServiceProfile</code>	<code>tosca.nodes.Root</code>

7.2 Data Types

7.2.1 Void

7.2.2 `tosca.datatypes.nfv.VnfProfile`

7.2.2.1 Description

The VnfProfile data type is defined in clause 9.2.8 of the present document.

7.2.3 `tosca.datatype.nfv.NsVIPProfile`

7.2.3.1 Description

The NsVIPProfile data type describes additional instantiation data for a given NsVirtualLink used in a specific NS deployment flavour. Table 7.2.3.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.3.1-1: Type name, shorthand, and URI

Shorthand Name	NsVIPProfile
Type Qualified Name	toscanfv:NsVIPProfile
Type URI	tosca.datatypes.nfv.NsVIPProfile

7.2.3.2 Properties

The properties of the NsVIPProfile data type shall comply with the provisions set out in table 7.2.3.2-1.

Table 7.2.3.2-1: Properties

Name	Required	Type	Constraints	Description
max_bitrate_requirements	yes	tosca.datatypes.nfv.LinkBitrateRequirements		Specifies the maximum bitrate requirements for a VL instantiated according to this profile.
min_bitrate_requirements	yes	tosca.datatypes.nfv.LinkBitrateRequirements		Specifies the minimum bitrate requirements for a VL instantiated according to this profile.
qos	no	tosca.datatypes.nfv.NsVirtualLinkQos		Specifies the QoS requirements of a VL instantiated according to this profile.
service_availability_level	no	integer	greater_or_equal: 1	If present, specifies the service availability level for the VL instance created from this profile. See note 2.
virtual_link_protocol_data	no	list of toscanfv.NsVirtualLinkProtocolData		Specifies the protocol data for a virtual link. If more than 1 values are present, the order shall be the same as the order of the layer_protocols occurrences in the connectivity_type property of the same NsVirtualLink node, i.e. the first occurrence of the virtual_link_protocol_data represents the highest layer protocol data, and the last occurrence represents the lowest layer protocol data.
data_flow_mirroring	no	list of toscanfv.NsDataFlowMirroring		If present, it describes the data flow mirroring to be associated to the NS instance containing the NS VL created from this NsVIPProfile.
NOTE 1: A virtualLinkDescId property, which exists in ETSI GS NFV-IFA 014 [2] is not needed, as the NsVIPProfile is contained in the NsVirtualLink node.				

Name	Required	Type	Constraints	Description
NOTE 2: The value '1' expresses the highest service availability level.				

7.2.3.3 Definition

The syntax of the NsVlProfile data type shall comply with the following definition:

```

tosca.datatypes.nfv.NsVlProfile:
  derived_from: tosca.datatypes.Root
  description: Describes additional instantiation data for a given NsVirtualLink
used in a specific NS deployment flavour.
  properties:
    max_bitrate_requirements:
      type: tosca.datatypes.nfv.LinkBitrateRequirements
      description: Specifies the maximum bitrate requirements for a VL instantiated
according to this profile.
      required: true
    min_bitrate_requirements:
      type: tosca.datatypes.nfv.LinkBitrateRequirements
      description: Specifies the minimum bitrate requirements for a VL instantiated
according to this profile.
      required: true
    qos:
      type: tosca.datatypes.nfv.NsVirtualLinkQos
      description: Specifies the QoS requirements of a VL instantiated according to
this profile.
      required: false
    service_availability_level:
      type: integer
      description: Specifies the service availability level for the VL instance
created from this profile
      required: false
      constraints:
        - greater_or_equal: 1
    virtual_link_protocol_data:
      type: list
      description: Specifies the protocol data for a virtual link.
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.NsVirtualLinkProtocolData
    data_flow_mirroring:
      type: list
      description: It describes the data flow mirroring to be associated to the NS
instance containing the NS VL created from this NsVlProfile..
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.NsDataFlowMirroring

```

7.2.3.4 Examples

None.

7.2.3.5 Additional Requirements

None.

7.2.4 `tosca.datatypes.nfv.ConnectivityType`

7.2.4.1 Description

The `ConnectivityType` data type is defined in clause 9.2.4 of the present document.

7.2.5 `tosca.datatypes.nfv.NsVirtualLinkQos`

7.2.5.1 Description

The `NsVirtualLinkQoS` describes QoS data type a given `NsVirtualLink` used in an NS deployment flavour. Table 7.2.5.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.5.1-1: Type name, shorthand, and URI

Shorthand Name	<code>NsVirtualLinkQos</code>
Type Qualified Name	<code>toscanfv:NsVirtualLinkQos</code>
Type URI	<code>tosca.datatypes.nfv.NsVirtualLinkQos</code>

7.2.5.2 Properties

The properties of the `NsVirtualLinkQos` data type shall comply with the provisions set out in table 7.2.5.2-1.

Table 7.2.5.2-1: Properties

Name	Required	Type	Constraints	Description
<code>priority</code>	no	integer	<code>greater_or_equal: 0</code>	Specifies the priority level in case of congestion on the underlying physical links.

7.2.5.3 Definition

The syntax of the `NsVirtualLinkQos` data type shall comply with the following definition:

```
tosca.datatypes.nfv.NsVirtualLinkQos:
  derived_from: toska.datatypes.nfv.Qos
  description: describes QoS data for a given VL used in a VNF deployment flavour
  properties:
    priority:
      type: integer
      constraints:
        - greater_or_equal: 0
      description: Specifies the priority level in case of congestion on the
underlying physical links
      required: false
```

7.2.5.4 Examples

None.

7.2.5.5 Additional Requirements

None.

7.2.6 `tosca.datatypes.nfv.LinkBitrateRequirements`

7.2.6.1 Description

The `LinkBitrateRequirements` data type is defined in clause 9.2.5 of the present document.

7.2.7 Void

7.2.8 Void

7.2.9 Void

7.2.10 Void

7.2.11 `tosca.datatypes.nfv.CpProtocolData`

7.2.11.1 Description

The `CpProtocolData` data type is defined in clause 9.2.6 of the present document.

7.2.12 `tosca.datatypes.nfv.AddressData`

7.2.12.1 Description

The `AddressData` data type is defined in clause 9.2.3 of the present document.

7.2.13 `tosca.datatypes.nfv.L2AddressData`

7.2.13.1 Description

The `L2AddressData` data type is defined in clause 9.2.1 of the present document.

7.2.14 `tosca.datatypes.nfv.L3AddressData`

7.2.14.1 Description

The `L3AddressData` data type is defined in clause 9.2.2 of the present document.

7.2.15 `tosca.datatypes.nfv.Qos`

7.2.15.1 Description

The `Qos` data type is defined in clause 9.2.7 of the present document.

7.2.16 `tosca.datatypes.nfv.NsProfile`

7.2.16.1 Description

The `NsProfile` data type describes a profile for instantiating nested NSs which are constituents of an NS with a particular NS DF as defined in ETSI GS NFV-IFA 014 [2]. Table 7.2.16.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.16.1-1: Type name, shorthand, and URI

Shorthand Name	NsProfile
Type Qualified Name	toscanfv:NsProfile
Type URI	tosca.datatypes.nfv.NsProfile

7.2.16.2 Properties

The properties of the NsProfile data type shall comply with the provisions set out in table 7.2.16.2-1.

Table 7.2.16.2-1: Properties

Name	Required	Type	Constraints	Description
ns_instantiation_level	no	string		Identifier of the instantiation level of the NS DF to be used for instantiation. If not present, the default instantiation level as declared in the NSD shall be used.
target_ns_scale_level_info	no	map of tosca.datatypes.nfv.NsScaleInfo		This attribute is applicable for NS target scale level instantiation. For each NS scaling aspect of the deployment flavour applicable to this NS instance, defines the target NS scale level to which the NS instance to be instantiated. If the property is present it shall contain all NS scaling aspects. See note 2.
min_number_of_instances	yes	integer	greater_or_equal: 0	Minimum number of instances of the NS based on this NSD that is permitted to exist for this NsProfile.
max_number_of_instances	yes	integer	greater_or_equal: 0	Maximum number of instances of the NS based on this NSD that is permitted to exist for this NsProfile.
version_dependency	no	list of tosca.datatypes.nfv.VersionDependency		Identifies versions of descriptors of other constituents in the NSD upon which the nested NS depends. The dependencies may be described for the NSD referenced with descriptor_id in the NS node where this profile is defined and for NSDs with the same ext_invariant_id. There shall not be more than one version_dependency present with the same dependent_constituent_id. See note.
deployable_modules_in_constituent_vnf	no	map of tosca.datatypes.nfv.SelectedDeployableModules	min_length: 1	Indicates the selected deployable modules of a VNF instance which is a constituent of the NS instances created from this profile.
NOTE 1: This property does not indicate any ordering requirement, i.e. the dependent nested NS instance and the other constituents upon which this nested NS instance depends may be created or incorporated to the NS in any order, unless specified otherwise by the 'dependency' requirement in the node templates in the NSD. NOTE 2: The target size for NS instantiation may be provided as NS instantiation level, or alternatively as target_ns_scale_level_info, but not both. If none of the two attributes (ns_instantiation_level or target_ns_scale_level_info) are present, the default NS instantiation level as declared in the deployment flavour shall be used.				

7.2.16.3 Definition

The syntax of the NsProfile data type shall comply with the following definition:

```

tosca.datatypes.nfv.NsProfile:
  derived_from: tosca.datatypes.Root
  description: describes a profile for instantiating NSs of a particular NS DF
  according to a specific NSD and NS DF.
  properties:
    ns_instantiation_level:
      type: string
      description: Identifier of the instantiation level of the NS DF to be used for
  instantiation. If not present, the default instantiation level as declared in the NSD
  shall be used.
      required: false
    target_ns_scale_level:
      type: map
      description: For each NS scaling aspect of the deployment flavour applicable
  to this NS instance, defines the target NS scale level to which the NS instance to be
  instantiated. If the property is present it shall contain all NS scaling aspects.
      required: false
    entry_schema:
      type: tosca.datatypes.nfv.NsScaleInfo
    min_number_of_instances:
      type: integer
      description: Minimum number of instances of the NS based on this NSD that is
  permitted to exist for this NsProfile.
      required: true
      constraints:
        - greater_or_equal: 0
    max_number_of_instances:
      type: integer
      description: Maximum number of instances of the NS based on this NSD that is
  permitted to exist for this NsProfile.
      required: true
      constraints:
        - greater_or_equal: 0
    version_dependency:
      type: list
      description: Identifies versions of descriptors of other constituents in the
  NSD upon which the nested NS depends. The dependencies may be described for the NSD
  referenced with descriptor_id in the NS node where this profile is defined and for
  NSDs with the same ext_invariant_id.
      required: false
    entry_schema:
      type: tosca.datatypes.nfv.VersionDependency
    deployable_modules_in_constituent_vnf:
      type: map #key: node template name of a constituent VNF
      description: Indicates the selected deployable modules of a VNF instance which
  is a constituent of the NS instances created from this profile.
      entry_schema:
        type: tosca.datatypes.nfv.SelectedDeployableModules
      constraints:
        - min_length: 1

```

7.2.16.4 Example

None.

7.2.16.5 Additional Requirements

None.

7.2.17 tosca.datatypes.nfv.Mask

7.2.17.1 Description

The Mask data type describes the value to be matched for a sequence of bits at a particular location in a frame. Table 7.2.17.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.17.1-1: Type name, shorthand, and URI

Shorthand Name	Mask
Type Qualified Name	toscanfv:Mask
Type URI	tosca.datatypes.nfv.Mask

7.2.17.2 Properties

The properties of the Mask data type shall comply with the provisions set out in table 7.2.17.2-1.

Table 7.2.17.2-1: Properties

Name	Required	Type	Constraints	Description
starting_point	yes	integer		Indicates the offset between the last bit of the source mac address and the first bit of the sequence of bits to be matched.
length	yes	integer		Indicates the number of bits to be matched.
value	yes	string		Provides the sequence of bit values to be matched.

7.2.17.3 Definition

The syntax of the Mask data type shall comply with the following definition:

```
tosca.datatypes.nfv.Mask:
  derived_from: tosca.datatypes.Root
  properties:
    starting_point:
      description: Indicates the offset between the last bit of the source mac
address and the first bit of the sequence of bits to be matched.
      type: integer
      constraints:
        - greater_or_equal: 1
      required: true
    length:
      description: Indicates the number of bits to be matched.
      type: integer
      constraints:
        - greater_or_equal: 1
      required: true
    value:
      description: Provide the sequence of bit values to be matched.
      type: string
      required: true
```

7.2.17.4 Examples

None.

7.2.18 tosa.datatypes.nfv.NsOperationAdditionalParameters

7.2.18.1 Description

The NsOperationAdditionalParameters data type is an empty base type for deriving data types for describing NS specific additional parameters that affect the invocation of NS Lifecycle Management operations, as defined in ETSI GS NFV-IFA 014 [2]. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.18.1-1 specifies the declared names for this data type.

Table 7.2.18.1-1: Type name, shorthand, and URI

Shorthand Name	NsOperationAdditionalParameters
Type Qualified Name	toscanfv:NsOperationAdditionalParameters
Type URI	tosca.datatypes.nfv.NsOperationAdditionalParameters

7.2.18.2 Properties

None.

7.2.18.3 Definition

The syntax of the NsOperationAdditionalParameters data type shall comply with the following definition:

```
tosca.datatypes.nfv.NsOperationAdditionalParameters:
  derived_from: tosa.datatypes.Root
  description: Is an empty base type for deriving data types for describing NS-
specific additional parameters to be passed when invoking NS lifecycle management
operations
  #properties:
```

7.2.18.4 Examples

```
tosca_definitions_version: tosa_simple_yaml_1_3

node_types:
  tosa.example_NS:
    derived_from: tosa.nodes.nfv.NS
    properties:
      ..
    interfaces:
      Nslcm:
        operations:
          instantiate:
            inputs:
              additional_parameters:
                type: MyCompany.datatypes.nfv.NsInstantiateAdditionalParameters
          scale:
            inputs:
              additional_parameters:
                type: MyCompany.datatypes.nfv.NsScaleAdditionalParameters
          heal:
            inputs:
              additional_parameters:
                type: MyCompany.datatypes.nfv.NsHealAdditionalParameters

data_types:
  MyCompany.datatypes.nfv.NsInstantiateAdditionalParameters:
```

```

derived_from: toska.datatypes.nfv.NsOperationAdditionalParameters
properties:
  parameter_1:
    type: string
    required: true
  parameter_2:
    type: string
    required: true
    default: value_2

MyCompany.datatypes.nfv.NsScaleAdditionalParameters:
derived_from: toska.datatypes.nfv.NsOperationAdditionalParameters
properties:
  parameter_1:
    type: string
    required: true
  parameter_2:
    type: string
    required: true

MyCompany.datatypes.nfv.NsHealAdditionalParameters:
derived_from: toska.datatypes.nfv.NsOperationAdditionalParameters
properties:
  parameter_1:
    type: string
    required: true
  parameter_2:
    type: string
    required: true
    default: value_2

```

7.2.19 toska.datatypes.nfv.NsMonitoringParameter

7.2.19.1 Description

This data type is used to specify information on virtualised resource related performance metrics to be monitored at the NS level. Table 7.2.19.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.19.1-1: Type name, shorthand, and URI

Shorthand Name	NsMonitoringParameter
Type Qualified Name	toscanfv:NsMonitoringParameter
Type URI	tosca.datatypes.nfv.NsMonitoringParameter

7.2.19.2 Properties

The properties of the NsMonitoringParameter data type shall comply with the provisions set out in table 7.2.19.2-1.

Table 7.2.19.2-1: Properties

Name	Required	Type	Constraints	Description
name	no	string		Human readable name of the monitoring parameter.
performance_metric	yes	string	valid values: See YAML definition constraints	Identifies a performance metric to be monitored. Performance metric values shall be either set to: - A measurement name defined in clause 7.3 of ETSI GS NFV-IFA 027 [7]. In this case the NFVO computes these measurements from lower-level metrics collected from the VIM. - A measurement name defined in clause 7.1 of ETSI GS NFV-IFA 027 [7], without appending a sub-counter. In this case the NFVO collects these metrics from the VIM for all network resources allocated to all NS virtual links.
collection_period	no	scalar-unit.time		Describes the periodicity at which to collect the performance information.

7.2.19.3 Definition

The syntax of the NsMonitoringParameter data type shall comply with the following definition:

```

tosca.datatypes.nfv.NsMonitoringParameter:
  derived_from: toska.datatypes.Root
  description: Represents information on virtualised resource related performance
metrics applicable to the NS.
  properties:
    name:
      type: string
      description: Human readable name of the monitoring parameter
      required: true
    performance_metric:
      type: string
      description: Identifies a performance metric to be monitored, according to
ETSI GS NFV-IFA 027.
      required: true
      constraints:
        - valid_values: [byte_incoming_sap, byte_outgoing_sap,
packet_incoming_sap, packet_outgoing_sap, byte_incoming, byte_outgoing,
packet_incoming, packet_outgoing, energy_consumption_ns, energy_consumption_mean_ns ]
    collection_period:
      type: scalar-unit.time
      description: Describes the periodicity at which to collect the performance
information.
      required: false

```

7.2.19.4 Examples

None.

7.2.19.5 Additional Requirements

None.

7.2.20 `tosca.datatypes.nfv.VnfMonitoringParameter`

The `VnfMonitoringParameter` data type is defined in clause 9.2.9 of the present document.

7.2.21 `tosca.datatypes.nfv.NsVirtualLinkProtocolData`

7.2.21.1 Description

The `NsVirtualLinkProtocolData` data type describes one protocol layer and associated protocol data for a given virtual link used in a specific NS deployment flavour. Table 7.2.21.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.21.1-1: Type name, shorthand, and URI

Shorthand Name	<code>NsVirtualLinkProtocolData</code>
Type Qualified Name	<code>tosca.nfv.NsVirtualLinkProtocolData</code>
Type URI	<code>tosca.datatypes.nfv.NsVirtualLinkProtocolData</code>

7.2.21.2 Properties

The properties of the `NsVirtualLinkProtocolData` data type shall comply with the provisions set out in table 7.2.21.2-1.

Table 7.2.21.2-1: Properties

Name	Required	Type	Constraints	Description
<code>associated_layer_protocol</code>	yes	string	Valid values: See YAML definition constraints	Identifies one of the protocols a virtualLink gives access to (ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire) as specified by the <code>connectivity_type</code> property.
<code>l2_protocol_data</code>	no	<code>tosca.datatype.s.nfv.NsL2ProtocolData</code>		Specifies the L2 protocol data for a virtual link. Shall be present when the <code>associatedLayerProtocol</code> attribute indicates a L2 protocol and shall be absent otherwise.
<code>l3_protocol_data</code>	no	<code>tosca.datatype.s.nfv.NsL3ProtocolData</code>		Specifies the L3 protocol data for this virtual link. Shall be present when the <code>associatedLayerProtocol</code> attribute indicates a L3 protocol and shall be absent otherwise.

7.2.21.3 Definition

The syntax of the `NsVirtualLinkProtocolData` data type shall comply with the following definition:

```
tosca.datatypes.nfv.NsVirtualLinkProtocolData:
  derived_from: toska.datatypes.Root
  description: describes one protocol layer and associated protocol data for a
  given virtual link used in a specific NS deployment flavour
  properties:
    associated_layer_protocol:
      type: string
      description: Identifies one of the protocols a virtualLink gives access
      to (ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire) as specified by the
      connectivity_type property.
      required: true
      constraints:
        - valid_values: [ ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire ]
    l2_protocol_data:
      type: toska.datatypes.nfv.NsL2ProtocolData
```

```

    description: Specifies the L2 protocol data for a virtual link. Shall be
    present when the associatedLayerProtocol attribute indicates a L2 protocol and
    shall be absent otherwise.
    required: false
    l3_protocol_data:
      type: toska.datatypes.nfv.NsL3ProtocolData
      description: Specifies the L3 protocol data for this virtual link. Shall
      be present when the associatedLayerProtocol attribute indicates a L3 protocol and
      shall be absent otherwise.
      required: false

```

7.2.21.4 Examples

None.

7.2.21.5 Additional Requirements

None.

7.2.22 toska.datatypes.nfv.NsL2ProtocolData

7.2.22.1 Description

The NsL2ProtocolData data type describes L2 protocol data for a given virtual link used in a specific NS deployment flavour. Table 7.2.22.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.22.1-1: Type name, shorthand, and URI

Shorthand Name	NsL2ProtocolData
Type Qualified Name	toscanfv:NsL2ProtocolData
Type URI	tosca.datatypes.nfv.NsL2ProtocolData

7.2.22.2 Properties

The properties of the NsL2ProtocolData data type shall comply with the provisions set out in table 7.2.22.2-1.

Table 7.2.22.2-1: Properties

Name	Required	Type	Constraints	Description
name	no	string		Identifies the network name associated with this L2 protocol.
network_type	no	string	Valid values: See YAML definition constraints	Specifies the network type for this L2 protocol. The value may be overridden at run-time.
vlan_transparent	no	boolean	default: false	Specifies whether to support VLAN transparency for this L2 protocol or not.
mtu	no	integer	greater_than: 0	Specifies the Maximum Transmission Unit (MTU) value for this L2 protocol.
segmentation_id	no	string		If present, specifies a specific virtualised network segment, which depends on the network type. For e.g. VLAN ID for VLAN network type and tunnel ID for GRE/VXLAN network types. See note.
NOTE: If this property is included in the NSD, the property value shall be provided at run-time, unless a default value is provided at design time in the NSD. If a default value is provided at design-time, this value may be overridden at run-time.				

7.2.22.3 Definition

The syntax of the NsL2ProtocolData data type shall comply with the following definition:

```

tosca.datatypes.nfv.NsL2ProtocolData:
  derived_from: toska.datatypes.Root
  description: describes L2 protocol data for a given virtual link used in a
specific NS deployment flavour.
  properties:
    name:
      type: string
      description: Identifies the network name associated with this L2 protocol.
      required: false
    network_type:
      type: string
      description: Specifies the network type for this L2 protocol. The value
may be overridden at run-time.
      required: false
      constraints:
        - valid_values: [ flat, vlan, vxlan, gre ]
    vlan_transparent:
      type: boolean
      description: Specifies whether to support VLAN transparency for this L2
protocol or not.
      required: false
      default: false
    mtu:
      type: integer
      description: Specifies the maximum transmission unit (MTU) value for this
L2 protocol.
      required: false
      constraints:
        - greater_than: 0
    segmentation_id:
      type: string
      description: Specifies a specific virtualised network segment, which
depends on the network type. For e.g. VLAN ID for VLAN network type and tunnel ID
for GRE/VXLAN network types
      required: false

```

7.2.22.4 Examples

None.

7.2.22.5 Additional Requirements

None.

7.2.23 `tosca.datatypes.nfv.NsL3ProtocolData`

7.2.23.1 Description

The `NsL3ProtocolData` data type describes L3 protocol data for a given virtual link used in a specific NS deployment flavour. Table 7.2.23.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.23.1-1: Type name, shorthand, and URI

Shorthand Name	<code>NsL3ProtocolData</code>
Type Qualified Name	<code>tosca.nfv.NsL3ProtocolData</code>
Type URI	<code>tosca.datatypes.nfv.NsL3ProtocolData</code>

7.2.23.2 Properties

The properties of the `NsL3ProtocolData` data type shall comply with the provisions set out in table 7.2.23.2-1.

Table 7.2.23.2-1: Properties

Name	Required	Type	Constraints	Description
<code>name</code>	no	string		Identifies the network name associated with this L3 protocol.
<code>ip_version</code>	yes	string	Valid values: See YAML definition constraints	Specifies IP version of this L3 protocol. The value of the <code>ip_version</code> property shall be consistent with the value of the <code>layer_protocol</code> in the <code>connectivity_type</code> property of the virtual link node.
<code>cidr</code>	yes	string		Specifies the CIDR (Classless Inter-Domain Routing) of this L3 protocol. The value may be overridden at run-time.
<code>ip_allocation_pools</code>	no	list of <code>tosca.datatypes.nfv.NsIpAllocationPool</code>		Specifies the allocation pools with start and end IP addresses for this L3 protocol. The value may be overridden at run-time.

7.2.23.3 Definition

The syntax of the `NsL3ProtocolData` data type shall comply with the following definition:

```
tosca.datatypes.nfv.NsL3ProtocolData:
  derived_from: toska.datatypes.Root
  description: describes L3 protocol data for a given virtual link used in a
  specific NS deployment flavour.
  properties:
    name:
      type: string
      description: Identifies the network name associated with this L3 protocol.
```

```

    required: false
    ip_version:
      type: string
      description: Specifies IP version of this L3 protocol. The value of the
ip_version property shall be consistent with the value of the layer_protocol in
the connectivity_type property of the virtual link node.
      required: true
      constraints:
        - valid_values: [ ipv4, ipv6 ]
    cidr:
      type: string
      description: Specifies the CIDR (Classless Inter-Domain Routing) of this
L3 protocol. The value may be overridden at run-time.
      required: true
    ip_allocation_pools:
      type: list
      description: Specifies the allocation pools with start and end IP
addresses for this L3 protocol. The value may be overridden at run-time.
      required: false
    entry_schema:
      type: tosca.datatypes.nfv.NsIpAllocationPool

```

7.2.23.4 Examples

None.

7.2.23.5 Additional Requirements

None.

7.2.24 tosca.datatypes.nfv.NsIpAllocationPool

7.2.24.1 Description

The NsIpAllocationPool data type specifies a range of IP addresses. Table 7.2.24.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.24.1-1: Type name, shorthand, and URI

Shorthand Name	NsIpAllocationPool
Type Qualified Name	tosca.nfv.NsIpAllocationPool
Type URI	tosca.datatypes.nfv.NsIpAllocationPool

7.2.24.2 Properties

The properties of the NsIpAllocationPool data type shall comply with the provisions set out in table 7.2.24.2-1.

Table 7.2.24.2-1: Properties

Name	Required	Type	Constraints	Description
start_ip_address	yes	string		The IP address to be used as the first one in a pool of addresses derived from the cidr block full IP range
end_ip_address	yes	string		The IP address to be used as the last one in a pool of addresses derived from the cidr block full IP range

7.2.24.3 Definition

The syntax of the `NsIpAllocationPool` data type shall comply with the following definition:

```
tosca.datatypes.nfv.NsIpAllocationPool:
  derived_from: toska.datatypes.Root
  description: Specifies a range of IP addresses
  properties:
    start_ip_address:
      type: string
      description: The IP address to be used as the first one in a pool of
addresses derived from the cidr block full IP range
      required: true
    end_ip_address:
      type: string
      description: The IP address to be used as the last one in a pool of
addresses derived from the cidr block full IP range
      required: true
```

7.2.24.4 Examples

None.

7.2.24.5 Additional Requirements

None.

7.2.25 toska.datatypes.nfv.NsScalingAspect

7.2.25.1 Description

The `NsScalingAspect` data type describes the details of an aspect used for horizontal scaling. Table 7.2.25.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.25.1-1: Type name, shorthand, and URI

Shorthand Name	<code>NsScalingAspect</code>
Type Qualified Name	<code>toscanfv:NsScalingAspect</code>
Type URI	<code>tosca.datatypes.nfv.NsScalingAspect</code>

7.2.25.2 Properties

The properties of the `NsScalingAspect` data type shall comply with the provisions set out in table 7.2.25.2-1.

Table 7.2.25.2-1: Properties

Name	Required	Type	Constraints	Description
name	yes	string		Human readable name of the NS scaling aspect.
description	yes	string		Human readable description of the NS scaling aspect.
ns_scale_levels	yes	map of toska.datatypes.nfv.NsLevels		Description of the NS levels for this scaling aspect.

7.2.25.3 Definition

The syntax of the NsScalingAspect data type shall comply with the following definition:

```

tosca.datatypes.nfv.NsScalingAspect:
  derived_from: tosc.datatypes.Root
  description: describes the details of an aspect used for horizontal scaling
  properties:
    name:
      type: string
      description: Human readable name of the aspect
      required: true
    description:
      type: string
      description: Human readable description of the aspect
      required: true
    ns_scale_levels:
      type: map
      description: Description of the NS levels for this scaling aspect.
      required: true
      key_schema:
        type: integer # Integer type in order to number the levels. First level is
level 0.
      constraints:
        - greater_or_equal: 0
      entry_schema:
        type: tosc.datatypes.nfv.NsLevels

```

7.2.25.4 Examples

See clause A.17.

7.2.25.5 Additional Requirements

None.

7.2.26 tosc.datatypes.nfv.NsLevels

7.2.26.1 Description

The NsLevels data type describes the Ns levels. Table 7.2.26.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.26.1-1: Type name, shorthand, and URI

Shorthand Name	NsLevels
Type Qualified Name	toscanfv:NsLevels
Type URI	tosca.datatypes.nfv.NsLevels

7.2.26.2 Properties

The properties of the NsLevels data type shall comply with the provisions set out in table 7.2.26.2-1.

Table 7.2.26.2-1: Properties

Name	Required	Type	Constraints	Description
description	yes	string		Human readable description of the NS level.

7.2.26.3 Definition

The syntax of the NsLevels data type shall comply with the following definition:

```
tosca.datatypes.nfv.NsLevels:
  derived_from: toska.datatypes.Root
  description: describes the Ns levels
  properties:
    description:
      type: string
      description: Human readable description of the Ns level
      required: true
```

7.2.26.4 Examples

See clause A.17.

7.2.26.5 Additional Requirements

None.

7.2.27 toska.datatypes.nfv.ScaleNsByStepsData

7.2.27.1 Description

The ScaleNsByStepsData data type describes the information needed to scale an NS instance by one or more scaling steps, with respect to a particular NS scaling aspect as defined in ETSI GS NFV-IFA 013 [i.8]. Table 7.2.27.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.27.1-1: Type name, shorthand, and URI

Shorthand Name	ScaleNsByStepsData
Type Qualified Name	toscanfv:ScaleNsByStepsData
Type URI	tosca.datatypes.nfv.ScaleNsByStepsData

7.2.27.2 Properties

The properties of the ScaleNsByStepsData data type shall comply with the provisions set out in table 7.2.27.2-1.

Table 7.2.27.2-1: Properties

Name	Required	Type	Constraints	Description
scaling_direction	yes	string	Valid values: See YAML definition constraints	Indicates the type of the scale operation requested.
aspect	yes	string		Identifier of the scaling aspect.
number_of_steps	yes	integer		Number of scaling steps to be executed.

7.2.27.3 Definition

The syntax of the ScaleNsByStepsData data type shall comply with the following definition:

```
tosca.datatypes.nfv.ScaleNsByStepsData:
  derived_from: toska.datatypes.Root
  description: describes the information needed to scale an NS instance by one or
more scaling steps, with respect to a particular NS scaling aspect
```

```

properties:
  scaling_direction:
    type: string
    description: Indicates the type of the scale operation requested.
    required: true
    constraints:
      - valid_values: [ scale_out, scale_in ]
  aspect:
    type: string
    description: Identifier of the scaling aspect.
    required: true
  number_of_steps:
    type: integer
    description: Number of scaling steps to be executed.
    required: true
    constraints:
      - greater_than: 0
    default: 1

```

7.2.27.4 Examples

None.

7.2.27.5 Additional Requirements

None.

7.2.28 `tosca.datatypes.nfv.ScaleNsToLevelData`

7.2.28.1 Description

The `ScaleNsByStepsData` data type describes the information needed to scale an NS instance to a target size as defined in ETSI GS NFV-IFA 013 [i.8]. Table 7.2.28.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.28.1-1: Type name, shorthand, and URI

Shorthand Name	<code>ScaleNsToLevelData</code>
Type Qualified Name	<code>toscanfv:ScaleNsToLevelData</code>
Type URI	<code>tosca.datatypes.nfv.ScaleNsToLevelData</code>

7.2.28.2 Properties

The properties of the `ScaleNsToLevelData` data type shall comply with the provisions set out in table 7.2.28.2-1.

Table 7.2.28.2-1: Properties

Name	Required	Type	Constraints	Description
<code>instantiation_level</code>	no	string		Identifier of the target instantiation level of the current deployment flavour to which the NS is requested to be scaled. Either <code>instantiation_level</code> or <code>ns_scale_info</code> shall be provided.
<code>ns_scale_info</code>	no	map of integer		For each scaling aspect of the current deployment flavour, indicates the target scale level to which the NS is to be scaled. Either <code>instantiation_level</code> or <code>ns_scale_info</code> shall be provided.

7.2.28.3 Definition

The syntax of the ScaleNsToLevelData data type shall comply with the following definition:

```

tosca.datatypes.nfv.ScaleNsToLevelData:
  derived_from: toska.datatypes.Root
  description: describes the information needed to scale an NS instance to a target
size.
  properties:
    instantiation_level:
      type: string
      description: Identifier of the target instantiation level of the current
deployment flavour to which the NS is requested to be scaled. Either
instantiation_level or ns_scale_info shall be provided.
      required: false
    ns_scale_info:
      type: map # key: aspectId
      description: For each scaling aspect of the current deployment flavour,
indicates the target scale level to which the NS is to be scaled. Either
instantiation_level or ns_scale_info shall be provided.
      required: false
    entry_schema:
      type: integer
      constraints:
        - greater_or_equal: 0

```

7.2.28.4 Examples

None.

7.2.28.5 Additional Requirements

None.

7.2.29 toska.datatypes.nfv.NsDataFlowMirroring

7.2.29.1 Description

The NsDataFlowMirroring data describes the data flow mirroring information to be associated to an NS instance as defined in ETSI GS NFV-IFA 014 [2].

Table 7.2.29.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.29.1-1: Type name, shorthand, and URI

Shorthand Name	NsDataFlowMirroring
Type Qualified Name	toscanfv:NsDataFlowMirroring
Type URI	tosca.datatypes.nfv.NsDataFlowMirroring

7.2.29.2 Properties

The properties of the NsDataFlowMirroring data type shall comply with the provisions set out in table 7.2.29.2-1.

Table 7.2.29.2-1: Properties

Name	Required	Type	Constraints	Description
mirroring_job_name	yes	string		Name of Data Flow Mirroring Job.
description	no	string		Information description of Data Flow Mirroring Job.

7.2.29.3 Definition

The syntax of the NsDataFlowMirroring data type shall comply with the following definition:

```

tosca.datatypes.nfv.NsDataFlowMirroring:
  derived_from: toska.datatypes.Root
  description: describes the data flow mirroring information to be associated to
the NS instance.
  properties:
    mirroring_job_name:
      type: string
      description: Name of Data Flow Mirroring Job
      required: true
    description:
      type: string
      description: Information description of Data Flow Mirroring Job
      required: false

```

7.2.29.4 Example

None.

7.2.29.5 Additional Requirements

None.

7.2.30 toska.datatypes.nfv.NsScaleInfo

7.2.30.1 Description

The NsScaleInfo data type indicates for a given nsScalingAspect the corresponding scaleLevel. Table 7.2.30.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.2.30.1-1: Type name, shorthand and URI

Shorthand Name	NsScaleInfo
Type Qualified Name	toscanfv:NsScaleInfo
Type URI	tosca.datatypes.nfv.NsScaleInfo

7.2.30.2 Properties

The properties of the NsScaleInfo data type shall comply with the provisions set out in table 7.2.30.2-1.

Table 7.2.30.2-1: Properties

Name	Required	Type	Constraints	Description
nsScale_level	yes	integer	greater_or_equal: 0	The NS scale level for a particular aspect.

7.2.30.3 Definition

The syntax of the NsScaleInfo data type shall comply with the following definition:

```
tosca.datatypes.nfv.NsScaleInfo:
  derived_from: toska.datatypes.Root
  description: Indicates for a given nsScalingAspect the corresponding NS scaleLevel
  properties:
    scale_level:
      type: integer
      description: The scale level for a particular aspect
      required: true
      constraints:
        - greater_or_equal: 0
```

7.2.30.4 Examples

None.

7.2.30.5 Additional Requirements

None.

7.3 Artifact Types

None.

7.4 Capability Types

7.4.1 toska.capabilities.nfv.VirtualLinkable

7.4.1.1 Description

The VirtualLinkable capability type is defined in clause 9.4.1 of the present document.

7.4.2 toska.capabilities.nfv.Forwarding

7.4.2.1 Description

The Forwarding capability type describes the capabilities related to nodes which can be pointed by toska.relationships.nfv.ForwardTo relationship type. Table 7.4.2.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

NOTE: The forwarding capability represents the ability of a CP or SAP to receive and forward traffic flows. Traffic flows can be received by a CP from an NS Virtual Link and forwarded to the VNF or PNF to which the CP is attached. Symmetrically, traffic flows can be received by an external CP of the VNF or PNF and forwarded to an NS Virtual Link. Traffic flows can be received by a SAP from an external link and forwarded to the NS to which the SAP is attached. Symmetrically, traffic flows can be received from the NS to which the SAP is attached and forwarded to an external link. An ingress CP is an external CP that forwards traffic to a VNF, PNF or NS while and egress CP is an external CP that forwards traffic outside a VNF, PNF or NS. The same CP may but need not play both roles.

Table 7.4.2.1-1: Type name, shorthand, and URI

Shorthand Name	Forwarding
Type Qualified Name	toscanfv:Forwarding
Type URI	tosca.capabilities.nfv.Forwarding

7.4.2.2 Properties

None.

7.4.2.3 Definition

The syntax of the Forwarding capability type shall comply with the following definition:

```
tosca.capabilities.nfv.Forwarding:
  derived_from: toska.capabilities.Root
```

7.4.3 toska.capabilities.nfv.NsPaasServiceRequestable

7.4.3.1 Description

The NsPaasServiceRequestable capability indicates that the node that includes it can be pointed by a toska.relationships.nfv.NsPaasServiceRequestsTo relationship type. Table 7.4.3.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.4.3.1-1: Type name, shorthand, and URI

Shorthand Name	NsPaasServiceRequestable
Type Qualified Name	toscanfv:NsPaasServiceRequestable
Type URI	tosca.capabilities.nfv.NsPaasServiceRequestable

7.4.3.2 Properties

None.

7.4.3.3 Definition

The syntax of the NsPaasServiceRequestable capability type shall comply with the following definition:

```
tosca.capabilities.nfv.NsPaasServiceRequestable:
  derived_from: toska.capabilities.Node
  description: Indicates that the node that includes it can be pointed by a
tosca.relationships.nfv.NsPaasServiceRequestsTo relationship type.
```

7.4.4 toska.capabilities.nfv.AssociablePaasService

7.4.4.1 Description

The AssociablePaasService capability type is defined in clause 9.4.3 of the present document.

7.4.5 `tosca.capabilities.nfv.VirtualLinkCapabilityConfigurable`

7.4.5.1 Description

The `VirtualLinkCapabilityConfigurable` capability indicates that the node that includes it can be pointed by a `tosca.relationships.nfv.VirtualLinkCapabilityConfiguresTo` relationship type. Table 7.4.5.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.4.5.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VirtualLinkCapabilityConfigurable</code>
Type Qualified Name	<code>toscanfv:VirtualLinkCapabilityConfigurable</code>
Type URI	<code>tosca.capabilities.nfv.VirtualLinkCapabilityConfigurable</code>

7.4.5.2 Properties

None.

7.4.5.3 Definition

The syntax of the `VirtualLinkCapabilityConfigurable` capability type shall comply with the following definition:

```
tosca.capabilities.nfv.VirtualLinkCapabilityConfigurable:
  derived_from: toska.capabilities.Node
  description: Indicates that the node that includes it can be pointed by a
tosca.relationships.nfv.VirtualLinkCapabilityConfiguresTo relationship type.
```

7.5 Requirement Types

None.

7.6 Relationship Types

7.6.1 `tosca.relationships.nfv.VirtualLinksTo`

7.6.1.1 Description

The `VirtualLinksTo` relationship type is defined in clause 9.6.1 of the present document representing an association relationship between the VNF or PNF or Sap of a Nested NS and `NsVirtualLink` node types when used in an NSD.

7.6.2 `tosca.relationships.nfv.ForwardTo`

7.6.2.1 Description

The `ForwardTo` relationship type represents an association between two node types which are a part of NFP. Table 7.6.2.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.6.2.1-1: Type name, shorthand, and URI

Shorthand Name	<code>ForwardTo</code>
Type Qualified Name	<code>toscanfv:ForwardTo</code>
Type URI	<code>tosca.nodes.relationships.ForwardTo</code>

7.6.2.2 Properties

None.

7.6.2.3 Definition

The syntax of the ForwardTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.ForwardTo:
  derived_from: toska.relationships.Root
  valid_target_types: [ toska.capabilities.nfv.Forwarding ]
```

7.6.3 toska.relationships.nfv.NsPaasServiceRequestsTo

7.6.3.1 Description

This relationship type represents an association between NsPaasServiceProfile or NsVirtualLinkAdditionalCapability and NsPaasServiceRequest node types. Table 7.6.3.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.6.3.1-1: Type name, shorthand, and URI

Shorthand Name	NsPaasServiceRequestsTo
Type Qualified Name	toscanfv:NsPaasServiceRequestsTo
Type URI	tosca.relationships.nfv.NsPaasServiceRequestsTo

7.6.3.2 Properties

None.

7.6.3.3 Definition

The syntax of the NsPaasServiceRequestsTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.NsPaasServiceRequestsTo:
  derived_from: toska.relationships.DependsOn
  description: Represents an association relationship between NsPaasServiceProfile
or NsVirtualLinkAdditionalCapability and NsPaasServiceRequest node types
  valid_target_types: [ toska.capabilities.nfv.NsPaasServiceRequestable ]
```

7.6.4 toska.relationships.nfv.PaaSServiceAssociates

7.6.4.1 Description

The PaasServiceAssociates relationship type is defined in clause 9.6.4 of the present document.

7.6.5 toska.relationships.nfv.VirtualLinkCapabilityConfiguresTo

7.6.5.1 Description

This relationship type represents an association between NsVirtualLink and NsVirtualLinkAdditionalCapability node types. Table 7.6.5.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.6.5.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualLinkCapabilityConfiguresTo
Type Qualified Name	toscanfv:VirtualLinkCapabilityConfiguresTo
Type URI	tosca.relationships.nfv.VirtualLinkCapabilityConfiguresTo

7.6.5.2 Properties

None.

7.6.5.3 Definition

The syntax of the VirtualLinkCapabilityConfiguresTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.VirtualLinkCapabilityConfiguresTo:
  derived_from: toska.relationships.DependsOn
  description: Represents an association relationship between NsVirtualLink and
NsVirtualLinkAdditionalCapability node types
  valid_target_types: [ toska.capabilities.nfv.VirtualLinkCapabilityConfigurable ]
```

7.7 Interface Types

7.7.1 toska.interfaces.nfv.Nslcm

7.7.1.1 Description

The toska.interfaces.nfv.Nslcm interface type contains a set of TOSCA operations corresponding to the following NS LCM operations defined in ETSI GS NFV-IFA 013 [i.8]:

- Instantiate NS
- Scale NS
- Update NS
- Heal NS
- Terminate NS

The interface also contains TOSCA operations corresponding to preamble and postamble to the execution of the aforementioned base operations. The name of these operations is constructed according to the following pattern:

<base_operation_name>_start for a preamble

<base_operation_name>_end for a postamble

The designations ("_start", "_end") in the name of TOSCA operations are postfixes so that related operations are adjacent in an alphabetical listing.

Table 7.7.1.1-1 specifies the declared names for this interface type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.7.1.1-1: Type name, shorthand, and URI

Shorthand Name	Nslcm
Type Qualified Name	toscanfv:Nslcm
Type URI	tosca.interfaces.nfv.Nslcm

7.7.1.2 Definition

The syntax of the Nslcm interface type shall comply with the following definition:

```

tosca.interfaces.nfv.Nslcm:
  derived_from: toska.interfaces.Root
  description: This interface encompasses a set of TOSCA operations corresponding to
  NS LCM operations defined in ETSI GS NFV-IFA 013. as well as to preamble and postamble
  procedures to the execution of the NS LCM operations.
  operations:
    instantiate_start:
      description: Preamble to execution of the instantiate operation
    instantiate:
      description: Base procedure for instantiating an NS, corresponding to the
  Instantiate NS operation defined in ETSI GS NFV-IFA 013.
      inputs:
        additional_parameters:
          type: toska.datatypes.nfv.NsOperationAdditionalParameters
          required: false
      instantiate_end:
        description: Postamble to the execution of the instantiate operation
    terminate_start:
      description: Preamble to execution of the terminate operation
    terminate:
      description: Base procedure for terminating an NS, corresponding to the
  Terminate NS operation defined in ETSI GS NFV-IFA 013.
      terminate_end:
        description: Postamble to the execution of the terminate operation
    update_start:
      description: Preamble to execution of the update operation
    update:
      description: Base procedure for updating an NS, corresponding to the Update NS
  operation defined in ETSI GS NFV-IFA 013.
      update_end:
        description: Postamble to the execution of the update operation
    scale_start:
      description: Preamble to execution of the scale operation
    scale:
      description: Base procedure for scaling an NS, corresponding to the Scale NS
  operation defined in ETSI GS NFV-IFA 013.
      inputs:
        additional_parameters:
          type: toska.datatypes.nfv.NsOperationAdditionalParameters
          required: false
        scale_ns_by_steps_data:
          type: toska.datatypes.nfv.ScaleNsByStepsData
          description: Describes the information needed to scale an NS instance by
  one or more scaling steps, with respect to a particular NS scaling aspect as defined
  in ETSI GS NFV-IFA 013. Either scale_ns_by_steps_data or scale_ns_to_level_data shall
  be provided.
          required: false
        scale_ns_to_level_data:
          type: toska.datatypes.nfv.ScaleNsToLevelData
          description: Describes the information needed to scale an NS instance to a
  target size as defined in ETSI GS NFV-IFA 013. Either scale_ns_by_steps_data or
  scale_ns_to_level_data shall be provided.
          required: false
      scale_end:
        description: Postamble to the execution of the scale operation
    heal_start:
      description: Preamble to execution of the heal operation

```

```

    heal:
      description: Base procedure for healing an NS, corresponding to the Heal NS
operation defined in ETSI GS NFV-IFA 013.
      inputs:
        additional_parameters:
          type: toska.datatypes.nfv.NsOperationAdditionalParameters
          required: false
      heal_end:
        description: Postamble to the execution of the heal operation

```

7.7.1.3 Additional Requirements

The implementation and inputs keynames specified in TOSCA-Simple-Profile-YAML-v1.3 [20] for an operation definition may be included for each operation listed in the Nslcm interface definition.

When a TOSCA operation representing an NS LCM operation does not have an associated implementation keyname, the default implementation provided by the NFVO for this NS LCM operation applies.

The NSD consumer shall make available all parameters from the message invoking the NS LCM operation as inputs to the corresponding TOSCA interface operations. The `inputs` keyname can be used to specify additional input parameters for executing the operation.

In the operation definitions on the Nslcm interface, the `additional_parameters` (NS-specific extension of the `tosca.datatypes.nfv.NsOperationAdditionalParameters`) of inputs section describes the name and type of the additional parameters that can be submitted in the NS LCM operation request. Refer example in clause 7.2.17.

The implementation of preamble and postamble TOSCA operations (e.g. `instantiate_start`), if present, is invoked with the same parameters as the corresponding base TOSCA operation (e.g. `instantiate`). The inputs of the preamble and postamble operations shall not be defined in the NSD.

If an implementation is associated to a TOSCA operation that represents a preamble or a postamble to an NS LCM operation, the implementation logic is executed before or after the execution of the NS LCM operation implementation, respectively.

Starting with version 3.3.1 [i.19] of the present document, the Nslcm interface type definition grammar was changed to support notifications and operations. For backward compatibility, TOSCA-Simple-Profile-YAML-v1.3 [20], clause 3.7.5.5 specifies the provisions for handling the previous grammar. Support of the Release 2 Nslcm interface type definition grammar can be removed in subsequent versions of the present document.

7.7.1.4 Support of LCM scripts

In ETSI GS NFV-IFA 014 [2], the definition of the "LifeCycleManagementScript" information element of the NSD associates LCM scripts with events, where an event can be an external or an internal stimulus. These events are mapped to TOSCA operations of the NS node type in the following way:

- external stimuli are mapped to TOSCA operations corresponding to the NS LCM operations defined in ETSI GS NFV-IFA 013 [i.8];
- internal stimuli are mapped to preamble and postamble of these TOSCA operations.

LCM scripts can be regarded as artifacts that provide an NS-specific implementation of the TOSCA operation corresponding to the stimulus.

The script input parameters shall be provided to the script according to the declaration in the inputs field of the operation definition. The artifact type definition shall enable identifying the DSL used by the script. The artifact type definition for Python is provided in section 5.4.4.1 of TOSCA-Simple-Profile-YAML-v1.3 [20]. The artifact definition for Mistral is provided in clause A.7.2 of the present document.

7.7.1.5 Examples

The following example template fragments illustrate the concept. An LCM script is associated with the `instantiate_end` operation. As no LCM script is associated to the `instantiate` operation, its default implementation runs and before

running the post-instantiate-script. The inputs section of the instantiate_end operation definition provides additional input values to the post-instantiate-script, and the TOSCA artifacts definition conveys the type of DSL used as a scripting language.

```
tosca_definitions_version: tosca_simple_yaml_1_3

imports:
- ..

node_types:
  MyCompany.SunshineVPN.1_0.1_0:
    derived_from: tosca.nodes.nfv.NS
    ..

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineVPN.1_0.1_0
    ..

node_templates:
  SunshineVPN:
    type: MyCompany.SunshineVPN.1_0.1_0
    ..
  interfaces:
    Nslcm:
      operations:
        instantiate_end:
          implementation: post-instantiate-script
          inputs:
            script_input_1: value_1
            script_input_2: value_2

artifacts:
  post-instantiate-script:
    description: Instantiate workflow script
    type: tosca.artifacts.Implementation.Python
    file: instantiate.py
    #repository: ..
    #deploy_path: ..
    ..
```

7.7.2 tosca.interfaces.nfv.NsVnfIndicator

7.7.2.1 Description

The tosca.interfaces.nfv.NsVnfIndicator is an empty base interface type for deriving NS specific interface types that include VNF indicator specific notifications which will be used in a NS.

Table 7.7.2.1-1 specifies the declared names for this interface type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.7.2.1-1: Type name, shorthand, and URI

Shorthand Name	NsVnfIndicator
Type Qualified Name	tosca.nfv.NsVnfIndicator
Type URI	tosca.interfaces.nfv.NsVnfIndicator

7.7.2.2 Definition

The syntax of the VnfIndicator interface type shall comply with the following definition:

```
tosca.interfaces.nfv.NsVnfIndicator:
  derived_from: toska.interfaces.Root
  description: This interface is an empty base interface type for deriving NS
specific interface types that include VNF indicator specific notifications which will
be used in a NS.
```

7.7.2.3 Examples

See clause A.15.3.

7.8 Node Types

7.8.1 toska.nodes.nfv.NS

7.8.1.1 Description

The NFV Network Service (NS) node type describes an NS in terms of deployment, operational behaviour, and requirements, as defined by ETSI GS NFV-IFA 014 [2]. Table 7.8.1.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.8.1.1-1: Type name, shorthand, and URI

Shorthand Name	NS
Type Qualified Name	toscanfv:NS
Type URI	tosca.nodes.nfv.NS

7.8.1.2 Properties

The properties of the NS node type shall comply with the provisions set out in table 7.8.1.2-1.

Table 7.8.1.2-1: Properties

Name	Required	Type	Constraints	Description
descriptor_id	yes	string		Identifier of this NS descriptor. See note 2.
designer	yes	string		Identifies the designer of the NSD.
version	yes	string		Identifies the version of the NSD.
name	yes	string		Provides the human readable name of the NSD.
invariant_id	yes	string		Identifies an NSD in a version independent manner. This attribute is invariant across versions of NSD with no constraints on the changes across versions. See note 2.
ext_invariant_id	no	string		Identifies an NSD in a version independent manner. This attribute is invariant across versions of the NSD that fulfil certain conditions related to the external connectivity and management of the NS. See notes 2 and 6. When used in a nested NS node template in a composite NSD it allows for nested NS instances during NS LCM the use of an NSD different from the one referenced by the descriptor_id property, provided they

Name	Required	Type	Constraints	Description
				have the same ext_invariant_id. See note 7.
flavour_id	yes	string		Identifier of this NS DF within the NSD.
ns_profile	no	tosca.datatypes.nfv.NsProfile		Specifies a profile of an NS, when this NS is used as nested NS within another NS. See note 1.
service_availability_level	no	integer	greater_or_equal: 1	If present, specifies the service availability level for the NS instance. See note 3.
priority	no	integer	greater_or_equal: 0	Specifies the priority for the NS instance. Examples for the usage of priority include conflict resolution in case of resource shortage. See notes 4 and 5.
strict_nsd_constituent_onboarding	yes	boolean	default: true	Indicates if all NSD constituents referenced by this NSD shall be onboarded before onboarding the NSD identified by this information element. If true all referenced constituents shall be onboarded.
NOTE 1: This property is only used in an NS node template, when it is representing a nested NS within another NS. NOTE 2: The value of the descriptor_id string shall comply with an UUID format as specified in section 3 of IETF RFC 4122 [9]. NOTE 3: The value '1' expresses the highest service availability level. NOTE 4: The value '0' expresses the highest priority and the fact that the NS instance based on this NS DF cannot be pre-empted during resource allocation. NOTE 5: A NSD specific node type definition can further constrain the range of valid values by indicating an upper bound. NOTE 6: Different versions of an NSD have different descriptor_ids but can have the same ext_invariant_id. Different versions of the NSD with the same ext_invariant_id shall have the same number and name of NS deployment flavours, where each of them exposes: • same external connectivity, i.e. same number and name of the requirements for VirtualLinkable capability that represent SAPs • same number and name of NS instantiation levels. NOTE 6a: The content of each NS instantiation level may change: • same NS scaling aspects and same number of levels per aspect. NOTE 6b: The constituents of each scaling aspect and the deltas between levels may change. This condition implies that NSDs with the same ext_invariant_id preserve external invariancy. Therefore, fulfilling this condition allows to use a different version of a nested NSD in a composite NS instance without modification of the NSD on which the composite NS instance is based. The use of a different version is ultimately under the control of the service provider and it should consider if the composite NSD fulfils the requirements of the SAPs (e.g. bitrate, IP version, etc.). NOTE 7: When the NS node is used as a nested NS in a composite NSD, this property may only be included if the NS node type definition in the nested NSD includes the property with a value. If the property is supported in the NS node in the nested NSD, it should be included in the nested NS node in the composite NSD to avoid changes in the composite NSD caused by version changes in the nested NSD.				

7.8.1.3 Attributes

The attribute of the NS node type shall comply with the provisions set out in table 7.8.1.3-1.

Table 7.8.1.3-1: Attributes

Name	Required	Type	Constraints	Description
scale_status	no	map of integer		Scale status of the NS, one entry per aspect. Represents for every scaling aspect how "big" the NS has been scaled with reference to that aspect.

If the NS supports VNF indicators as the monitoring data as described in ETSI GS NFV-IFA 014 [2], the NS specific node type definition shall include one TOSCA attribute of a primitive type for each monitored VNF indicator.

NOTE 1: The value of the attribute for each monitored VNF indicator can be retrieved by passing a path to the `get_attribute` function: { `get_attribute`: [`<VNF_node_template_name>`, `<attribute_name_defined_in_the_corresponding_VNFD>`] }.

NOTE 2: The rule for naming the attribute for each monitored VNF indicator should be:

" VNF node template name "_" attribute name defined in the corresponding VNFD ". See example in clause A.15.3.

NOTE 3: In this version of the present document, the type of VNF indicators is constrained to primitive types. This is due to the limitations in the TOSCA-Simple-Profile-YAML-v1.3 [20] to define conditions on attributes of complex types.

If the `scale_status` attribute is used in a NS auto-scale policy, the NS specific node type definitions may include additional attribute definitions of type integer, one for each scaling aspect. See example in clause A.15.3.

NOTE 4: As the `scale_status` attribute is complex, the value of the individual scaling aspects can be retrieved by passing a path to the `get_attribute` function: { `get_attribute`: [`SELF`, `scale_status`, {`scaling_aspect`}] } . If the value of the `scale_status` property is needed in a constraint (`tosca.policies.nfv.NsAutoScale`), then its value can be retrieved in an indirect way by accessing the aforementioned additional attributes. This is due to the limitation mentioned in the note 3.

7.8.1.4 Requirements

The requirements of the NS node type shall comply with the provisions set out in table 7.8.1.4-1.

Table 7.8.1.4-1: Requirements

Name	Required	Capability type	Constraints	Description
virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Describes the requirements for linking to virtual link
associated_paas_service	no	tosca.capabilities.AssociablePaasService		Describes the requirements for associating to a PaaS Service

7.8.1.5 Capabilities

None.

7.8.1.6 Definition

The syntax of the NS node type shall comply with the following definition:

```
tosca.nodes.nfv.NS:
  derived_from: toska.nodes.Root
  properties:
    descriptor_id:
      type: string # UUID
      description: Identifier of this NS descriptor
      required: true
    designer:
      type: string
      description: Identifies the designer of the NSD.
      required: true
    version:
      type: string
      description: Identifies the version of the NSD.
      required: true
    name:
      type: string
      description: Provides the human readable name of the NSD.
      required: true
```

```

    invariant_id: # UUID
      type: string
      description: Identifies an NSD in a version independent manner. This
attribute is invariant across versions of NSD with no constraints on the changes
across versions.
      required: true
    ext_invariant_id: # UUID
      type: string
      description: Identifies an NSD in a version independent manner. This
attribute is invariant across versions of the NSD that fulfil certain conditions
related to the external connectivity and management of the NS. When used in a
nested NS node template in a composite NSD it allows for nested NS instances
during NS LCM the use of an NSD different from the one referenced by the
descriptor_id property, provided they have the same ext_invariant_id.
      required: false
    flavour_id:
      type: string
      description: Identifier of the NS Deployment Flavour within the NSD
      required: true
    ns_profile:
      type: tosca.datatypes.nfv.NsProfile
      description: Specifies a profile of a NS, when this NS is used as nested
NS within another NS.
      required: false
    service_availability_level:
      type: integer
      description: Specifies the service availability level for the NS instance.
      required: false
      constraints:
        - greater_or_equal: 1
    priority:
      type: integer
      description: Specifies the priority for the NS instance. Examples for the
usage of priority include conflict resolution in case of resource shortage.
      required: false
      constraints:
        - greater_or_equal: 0
    strict_nsd_constituent_onboarding:
      type: boolean
      description: Indicates if all NSD constituents referenced by this NSD
shall be onboarded before onboarding the NSD identified by this information
element. If true all referenced constituents shall be onboarded.
      required: true
      default: true
    attributes:
      scale_status:
        type: map # key: aspectId
        description: Scale status of the NS, one entry per aspect. Represents for
every scaling aspect how "big" the NS has been scaled w.r.t. that aspect.
        entry_schema:
          type: integer
          constraints:
            - greater_or_equal: 0
      requirements:
        - virtual_link:
            capability: tosca.capabilities.nfv.VirtualLinkable
            relationship: tosca.relationships.nfv.VirtualLinksTo
            node: tosca.nodes.nfv.NsVirtualLink
            occurrences: [ 0, 1 ]
        - associated_paas_service:
            capability: tosca.capabilities.nfv.AssociablePaasService

```



```

relationship: toska.relationships.nfv.PaaSServiceAssociates
node: toska.nodes.nfv.NsPaasServiceProfile
occurrences: [ 0, UNBOUNDED ]
interfaces:
  Nslcm:
    type: toska.interfaces.nfv.Nslcm

```

7.8.1.7 Artifact

None.

7.8.1.8 Additional requirements

For a given NSD, a new NS node type shall be defined following the below requirements:

- a) The node type shall be derived from: toska.nodes.nfv.NS.
- b) All properties listed in toska.nodes.nfv.NS where the "required:" field is set to "true" shall be included with their values indicated as constraints or assigned as final fixed values if only one value is permitted.
- c) Properties listed in in toska.nodes.nfv.NS where the "required:" field is set to "false" may be included.
- d) The capabilities, requirements, interfaces of toska.nodes.nfv.NS shall be preserved.
- e) Depending on the number of SAPs of the NS, additional requirements for VirtualLinkable capability shall be defined with the occurrences set to [0, 1]. In this case, it is the NSD author's choice to use the requirement for VirtualLinkable capability defined in the toska.nodes.nfv.NS node type or use only the additional requirements defined in the derived NS specific node type. In the latter case, the virtual_link requirement should be included in the node type definition with occurrences [0, 0].
- f) If the NSD supports external invariancy the NS node type definition shall include the ext_invariant_id property with its value indicated as constraint.

7.8.2 toska.nodes.nfv.Sap

7.8.2.1 Description

The Service Access Point (SAP) node type describes a connection point where an NS can be accessed, as defined by ETSI GS NFV-IFA 014 [2]. Table 7.8.2.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.8.2.1-1: Type name, shorthand, and URI

Shorthand Name	Sap
Type Qualified Name	toscanfv:Sap
Type URI	tosca.nodes.nfv.Sap

7.8.2.2 Properties

The properties applied to Sap node are derived from Cp node type as defined in clause 9.8.1 of the present document.

7.8.2.3 Attributes

None.

7.8.2.4 Requirements

The requirements of the Sap node type shall comply with the provisions set out in table 7.8.2.4-1.

Table 7.8.2.4-1: Requirements

Name	Required	Capability type	Constraints	Description
external_virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Specifies that CP instances require to be connected to a node that has a VirtualLinkable capability
internal_virtual_link	yes	tosca.capabilities.nfv.VirtualLinkable		Specifies that CP instances require to be connected to a node that has a VirtualLinkable capability

7.8.2.5 Capabilities

The capabilities of the Sap node type shall comply with the provisions set out in table 7.8.2.5-1.

Table 7.8.2.5-1: Capabilities

Name	Type	Constraints	Description
forwarding	tosca.capabilities.nfv.Forwarding		The forwarding capability exposed by the node.

7.8.2.6 Definition

The syntax of the Sap node type shall comply with the following definition:

```

tosca.nodes.nfv.Sap:
  derived_from: toska.nodes.nfv.Cp
  description: node definition of SAP.
  capabilities:
    forwarding:
      type: toska.capabilities.nfv.Forwarding
  requirements:
    - external_virtual_link:
        capability: toska.capabilities.nfv.VirtualLinkable
        relationship: toska.relationships.nfv.VirtualLinksTo
        occurrences: [0, 1]
    - internal_virtual_link:
        capability: toska.capabilities.nfv.VirtualLinkable
        relationship: toska.relationships.nfv.VirtualLinksTo
        occurrences: [1, 1]

```

7.8.2.7 Additional requirements

A node template of this type is used to represent a SAP only in the case the Sap is connected to an NsVirtualLink inside an NSD. The node template has the following requirements:

- internal_virtual_link requirement to allow to connect it to an NsVirtualLink inside an NSD;
- external_virtual_link requirement to allow to connect it to an NsVirtualLink outside an NSD.

In the case where a Sap is exposed by a VNF external connection point, a PNF external connection point or a Sap of the nested NS, the Sap node type does not apply.

7.8.2.8 Example

In a typical scenario, the Sap node template will be part of a service template representing a certain NS deployment flavour. The service template substitutes for a NS specific node type. In this substitution, the virtual_link requirement is mapped to the external_virtual_link requirement of the Sap node. This example is illustrated in clause A.7.3.

When a Sap re-exposes a VNF external connection point, the service template does not require an explicit node template of type Sap in a typical scenario where a NS specific node type is substituted by a service template representing a certain NS deployment flavour. In this substitution, the virtual_link requirement is mapped to the virtual_link requirement of the VNF node or a corresponding Forwarding node. The first case is illustrated in clause A.7.2 while the second case is illustrated in clause A.14.

7.8.3 toska.nodes.nfv.NsVirtualLink

7.8.3.1 Description

The NsVirtualLink node type represents the NsVirtualLinkDesc information element as defined in ETSI GS NFV-IFA 014 [2], which describes the requirements for a virtual link of a network service. Table 7.8.3.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.8.3.1-1: Type name, shorthand, and URI

Shorthand Name	NsVirtualLink
Type Qualified Name	toscanfv:NsVirtualLink
Type URI	tosca.nodes.nfv.NsVirtualLink

7.8.3.2 Properties

The properties of the NsVirtualLink node type shall comply with the provisions set out in table 7.8.3.2-1.

Table 7.8.3.2-1: Properties

Name	Required	Type	Constraints	Description
vl_profile	yes	tosca.datatype.nfv.NsVIProfile		Specifies instantiation parameters for a virtual link of a particular NS deployment flavour.
connectivity_type	yes	tosca.datatypes.nfv.ConnectivityType		Network service virtual link connectivity type.
test_access	no	list of string	Valid values: See YAML definition constraints	Test access facilities available on the VL.
description	no	string		Human readable information on the purpose of the virtual link (e.g. VL for control plane traffic).

7.8.3.3 Attributes

None.

7.8.3.4 Requirements

The requirements of the NsVirtualLink node type shall comply with the provisions set out in table 7.8.3.4-1.

Table 7.8.3.4-1: Requirements

Name	Required	Capability type	Constraints	Description
associated_paas_service	no	tosca.capabilities.AssociablePaasService		Describes the requirements for associating to a PaaS Service
additional_capability	no	tosca.capabilities.VirtualLinkCapabilityConfigurable		Describes the requirements for a virtual link to be connected to a node that has a VirtualLinkCapabilityConfigurable capability

7.8.3.5 Capabilities

The capabilities of the NsVirtualLink node type shall comply with the provisions set out in table 7.8.3.5-1.

Table 7.8.3.5-1: Capabilities

Name	Type	Constraints	Description
virtual_linkable	tosca.capabilities.nfv.VirtualLinkable		VirtualLinkable capability

7.8.3.6 Definition

The syntax of the NsVirtualLink node type shall comply with the following definition:

```

tosca.nodes.nfv.NsVirtualLink:
  derived_from: tosca.nodes.Root
  description: node definition of Virtual Links
  properties:
    vl_profile:
      type: tosca.datatypes.nfv.NsVlProfile # only covers min/max bitrate
  requirements
    description: Specifies instantiation parameters for a virtual link of a
particular NS deployment flavour.
    required: true
    connectivity_type:
      type: tosca.datatypes.nfv.ConnectivityType
      required: true
    test_access:
      type: list
      description: Test access facilities available on the VL
      required: false
    entry_schema:
      type: string
      constraints:
        - valid_values: [ passive_monitoring, active_loopback ]
    description:
      type: string
      required: false
      description: Human readable information on the purpose of the virtual link
(e.g. VL for control plane traffic).
    capabilities:
      virtual_linkable:
        type: tosca.capabilities.nfv.VirtualLinkable
  requirements:
    - associated_paas_service:
        capability: tosca.capabilities.nfv.AssociablePaasService
        relationship: tosca.relationships.nfv.PaaSServiceAssociates
        node: tosca.nodes.nfv.NsPaasServiceProfile
        occurrences: [ 0, UNBOUNDED ]
    - additional_capability:
        capability: tosca.capabilities.nfv.VirtualLinkCapabilityConfigurable
        relationship: tosca.relationships.nfv.VirtualLinkCapabilityConfiguresTo
        node: tosca.nodes.nfv.NsVirtualLinkAdditionalCapability
        occurrences: [ 0, UNBOUNDED ]

```

7.8.3.7 Artifact

None.

7.8.3.8 Additional Requirements

None.

7.8.3.9 Example

None.

7.8.4 `tosca.nodes.nfv.Cp`

7.8.4.1 Description

The Cp node type is defined in clause 9.8.1 of the present document.

7.8.5 `tosca.nodes.nfv.NfpPositionElement`

7.8.5.1 Description

The NfpPositionElement node type represents the NfpPositionElement information element as defined in ETSI GS NFV-IFA 014 [2], which describes one or two CPD(s) or SAPD(s) for a given Vnf, Pnf or Ns. Table 7.8.5.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

A NfpPositionElement node type has a requirement for a forwarding capability to be exposed by the VNFs, PNFs or NSs, in order to re-expose this capability to an NfpPosition node type.

NOTE: The NfpPosition and NfpPositionElement node types of the VNFFG model describe the entities in VIM for enabling packets/frames to traverse the constituent VNFs, PNFs or Nested NSs of the Network Forwarding Path.

Table 7.8.5.1-1: Type name, shorthand, and URI

Shorthand Name	NfpPositionElement
Type Qualified Name	toscanfv:NfpPositionElement
Type URI	tosca.nodes.nfv.NfpPositionElement

7.8.5.2 Properties

None.

7.8.5.3 Attributes

None.

7.8.5.4 Requirements

The requirements of the NfpPositionElement node type shall comply with the provisions set out in table 7.8.5.4-1.

Table 7.8.5.4-1: Requirements

Name	Required	Capability Type	Constraints	Description
profile_element	yes	tosca.capabilities.nfv.Forwarding		Describes the requirement for the constituent of the NfpPositionElement.

7.8.5.5 Capabilities

The capabilities of the NfpPositionElement node type shall comply with the provisions set out in table 7.8.5.5-1.

Table 7.8.5.5-1: Capabilities

Name	Type	Constraints	Description
forwarding	tosca.capabilities.nfv.Forwarding		NfpPositionElement forwarding capability

7.8.5.6 Definition

The syntax of the NfpPositionElement node type shall comply with the following definition:

```
tosca.nodes.nfv.NfpPositionElement:
  derived_from: tosca.nodes.Root
  description: node definition of NfpPositionElement
  capabilities:
    forwarding:
      type: tosca.capabilities.nfv.Forwarding
  requirements:
    - profile_element:
        capability: tosca.capabilities.nfv.Forwarding
        relationship: tosca.relationships.nfv.ForwardTo
        occurrences: [ 1, 2 ] #When the number of occurrences is 1, the ingress
and egress traffic is associated to a single VnfExtCp or Sap; When the number of
occurrences is 2, the ingress VnfExtCp or Sap is associated to the first value and
the egress VnfExtCp or Sap is associated to the second value.
```

7.8.5.7 Artifact

None.

7.8.5.8 Additional Requirements

The valid node types for the "profile_element" requirements shall be limited to tosca.nodes.nfv.Forwarding and tosca.nodes.nfv.Sap.

7.8.5.9 Example

See clause A.14.

7.8.6 tosca.nodes.nfv.NFP

7.8.6.1 Description

The NFP node type associates traffic flow criteria to a list of descriptors associated to the connection points and service access points to be visited by traffic flows matching these criteria. Table 7.8.6.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.8.6.1-1: Type name, shorthand, and URI

Shorthand Name	NFP
Type Qualified Name	toscanfv:NFP
Type URI	tosca.nodes.nfv.NFP

7.8.6.2 Properties

None.

7.8.6.3 Attributes

None.

7.8.6.4 Requirements

The requirements of the NFP node type shall comply with the provisions set out in table 7.8.6.4-1.

Table 7.8.6.4-1: Requirements

Name	Required	Capability type	Constraints	Description
nfp_position	yes	tosca.capabilities.nfv.Forwarding		

7.8.6.5 Capabilities

None.

7.8.6.6 Definition

The syntax of the NFP node type shall comply with the following definition:

```
tosca.nodes.nfv.NFP:
  derived_from: tosca.nodes.Root
  description: node definition of NFP
  requirements:
    - nfp_position:
        capability: tosca.capabilities.nfv.Forwarding
        node: tosca.nodes.nfv.NfpPosition
        relationship: tosca.relationships.nfv.ForwardTo
        occurrences: [ 1, UNBOUNDED ]
```

7.8.7 tosca.nodes.nfv.NfpPosition

7.8.7.1 Description

The NfpPosition node type describes the reference of one or more NfpPositionElements and rules on how to route traffic flows among VnfExtCp or SAP instances corresponding to these elements. Table 7.8.7.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

NOTE: The NfpPosition and NfpPositionElement node types of the VNFFG model describe the entities in VIM for enabling packets/frames to traverse the constituent VNFs, PNFs or Nested NSs of the Network Forwarding Path.

Table 7.8.7.1-1: Type name, shorthand, and URI

Shorthand Name	NfpPosition
Type Qualified Name	toscanfv:NfpPosition
Type URI	tosca.nodes.nfv.NfpPosition

7.8.7.2 Properties

The properties of the NfpPosition node type shall comply with the provisions set out in table 7.8.7.2-1.

Table 7.8.7.2-1: Properties

Name	Required	Type	Constraints	Description
forwarding_behaviour	no	string	Possible values: "all", "lb", "ff"	Identifies a rule to apply to forward traffic to CP or SAP instances corresponding to the referenced NfpPositionElement(s). The minimum list of rules to be supported shall include: all = Traffic flows shall be forwarded simultaneously to all CP or SAP instances created from the referenced CP profile(s). lb (load balancing) = Traffic flows shall be forwarded to one CP or SAP instance created from the referenced CP profile(s) selected based on a load-balancing algorithm. The following value may be used as well: ff (fast failover) = Traffic flows shall be forwarded to the next CP or SAP in case they cannot be forwarded to a CP or SAP instance created from the referenced CP profile(s). See note.
forwarding_behaviour_input_parameters	no	map of string		Provides input parameters to configure the forwarding behaviour (e.g. identifies a load balancing algorithm). This property is reserved for future use in the present document.
NOTE: When no rules are provided and there are multiple CP or SAP instances corresponding to the referenced CP profile(s), the VIM and/or the NFVI are expected to apply NFP-independent rules determined by means outside the scope of the present document.				

7.8.7.3 Attributes

None.

7.8.7.4 Requirements

The requirements of the NfpPosition node type shall comply with the provisions set out in table 7.8.7.4-1.

Table 7.8.7.4-1: Requirements

Name	Required	Capability type	Constraints	Description
element	yes	tosca.capabilities.nfv.Forwarding		Specifies that an NfpPosition requires a node that has a forwarding capability.

7.8.7.5 Capabilities

The capabilities of the NfpPosition node type shall comply with the provisions set out in table 7.8.7.5-1.

Table 7.8.7.5-1: Capabilities

Name	Type	Constraints	Description
forwarding	tosca.capabilities.nfv.Forwarding		NfpPosition forwarding capability

7.8.7.6 Definition

The syntax of the NfpPosition node type shall comply with the following definition:

```
tosca.nodes.nfv.NfpPosition:
  derived_from: tosca.nodes.Root
  description: node definition of NFP position
  properties:
    forwarding_behaviour:
      type: string
      description: Identifies a rule to apply to forward traffic to CP or SAP
instances corresponding to the referenced NfpPositionElement(s).
      constraints:
        - valid_values: [ all, lb, ff ]
      required: false
    forwarding_behaviour_input_parameters:
      description: Provides input parameters to configure the forwarding
behaviour.
      type: map
      required: false
      entry_schema:
        type: string
  capabilities:
    forwarding:
      type: tosca.capabilities.nfv.Forwarding
  requirements:
    - element:
      capability: tosca.capabilities.nfv.Forwarding
      node: tosca.nodes.nfv.NfpPositionElement
      relationship: tosca.relationships.nfv.ForwardTo
      occurrences: [ 1, UNBOUNDED ]
```

7.8.7.7 Artifact

None.

7.8.7.8 Additional Requirements

None.

7.8.7.9 Example

See clause A.14.

7.8.8 tosca.nodes.nfv.Forwarding

7.8.8.1 Description

The Forwarding node type represents a point in the NS topology that is inserted between a virtual link (VirtualLinkable) requirement of a VNF/PNF node template (in effect, an external connection point of the VNF/PNF) or a nested NS node template and the virtual link template satisfying this requirement. Table 7.8.8.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.8.8.1-1: Type name, shorthand, and URI

Shorthand Name	Forwarding
Type Qualified Name	toscanfv:Forwarding
Type URI	tosca.nodes.nfv.Forwarding

7.8.8.2 Properties

None.

7.8.8.3 Attributes

None.

7.8.8.4 Requirements

The requirements of the Forwarding node type shall comply with the provisions set out in table 7.8.8.4-1.

Table 7.8.8.4-1: Requirements

Name	Required	Capability Type	Constraints	Description
virtual_link	No	tosca.capabilities.nfv.VirtualLinkable		Describes the requirement for linking to a virtual linkable node type.

7.8.8.5 Capabilities

The capabilities of the Forwarding node type shall comply with the provisions set out in table 7.8.8.5-1.

Table 7.8.8.5-1: Capabilities

Name	Type	Constraints	Description
forwarding	tosca.capabilities.nfv.Forwarding		The forwarding capability exposed by the node.
virtual_linkable	tosca.capabilities.nfv.VirtualLinkable		The virtual linkable capability exposed by the node.

7.8.8.6 Definition

The syntax of the Forwarding node type shall comply with the following definition:

```
tosca.nodes.nfv.Forwarding:
  derived_from: tosca.nodes.Root
  capabilities:
    virtual_linkable:
      type: tosca.capabilities.nfv.VirtualLinkable
    forwarding:
      type: tosca.capabilities.nfv.Forwarding
  requirements:
    - virtual_link:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
        occurrences: [0, 1]
```

7.8.8.7 Artifact

None.

7.8.8.8 Additional Requirements

A node template with Forwarding node type when present in a NSD service template shall only be used for the following design:

- a) in the VNFFGD design, at least one node template of NfpPositionElement node type shall be included in the same NSD service template and the Forwarding node template shall be used as the node target of the profile_element requirement;
- b) in the dataFlowMirroring design, at least one policy with DataFlowInfo policy type shall be included in the same NSD service template and the Forwarding node template shall be used as the target of this policy;
- c) in the NsAffinityRule or NsAntiAffinityRule policy rule design with "l2_network" as the scope, at least one such policy shall be included in the same NSD service template and the Forwarding node template shall be used as the target of this policy.

7.8.8.9 Example

See clauses A.14, A.19 and A.20.

7.8.9 toska.nodes.nfv.NsPaasServiceRequest

7.8.9.1 Description

The NsPaasServiceRequest node type describes the information and requirements of PaaS Services that the NS needs for its operation as specified in ETSI GS NFV-IFA 014 [2].

Table 7.8.9.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.8.9.1-1: Type name, shorthand, and URI

Shorthand Name	NsPaasServiceRequest
Type Qualified Name	toscanfv:NsPaasServiceRequest
Type URI	tosca.node.nfv.NsPaasServiceRequest

7.8.9.2 Properties

The properties of the NsPaasServiceRequest node type shall comply with the provisions set out in table 7.8.9.2-1.

Table 7.8.9.2-1: Properties

Name	Required	Type	Constraints	Description
name	no	string		Provides the human readable name of the requested PaaS Service.
is_mandatory	yes	boolean	default: false	Indicates whether the requested PaaS Service is mandatory for successful NS operation. Default is FALSE, i.e. the PaaS Service is optional for successful NS operation.
type	yes	string		The type of PaaS Service. The value of this property is expected to be matched against values of the registered PaaS Services in the PaaS Services Repository (PSR).
min_version	no	string		In the case the PaaS Service is versioned, it specifies the minimum version of the requested PaaS Service. See note.
preferred_version	no	string		In the case the PaaS Service is versioned, it specifies the preferred version of the requested PaaS Service. See note.
usage_format	yes	string	Valid values: See YAML definition constraints	Describes the usage format of the PaaS Service, as specified in ETSI GS NFV-IFA 014 [2].
specific_parameters	no	map of string		Provides attributes specific to the type of the PaaS Service. This property is reserved for future use in the present document.
NOTE: It is assumed that some form of "semantic versioning" is used to define the version of PaaS Services, so that minimum and preferred PaaS Service versions can be matched by the NFV-MANO.				

7.8.9.3 Attributes

None.

7.8.9.4 Requirements

None.

7.8.9.5 Capabilities

The capabilities of the NsPaasServiceRequest node type shall comply with the provisions set out in table 7.8.9.5-1.

Table 7.8.9.5-1: Capabilities

Name	Type	Constraints	Description
ns_paas_service_requesting	tosca.capabilities.nfv.NsPaasServiceRequestable		Defines ability of NsPaasServiceRequestable.

7.8.9.6 Definition

The syntax of the NsPaasServiceRequest node type shall comply with the following definition:

```

tosca.nodes.nfv.NsPaasServiceRequest:
  derived_from: tosca.nodes.Root
  description: describes the information about and requirements by the NS on the PaaS
  Services that the NS needs for its operation as specified in [2].
  properties:
    name:
      type: string
      description: Provides the human readable name of the requested PaaS Service.
      required: false
    is_mandatory:

```

```

    type: boolean
    description: Indicates whether the PaaS Service is mandatory for successful NS
operation.
    required: true
    default: false
  type:
    type: string
    description: The type of PaaS Service.
    required: true
  min_version:
    type: string
    description: In the case the PaaS Service is versioned, it specifies the
minimum version of the requested PaaS Service.
    required: false
  preferred_version:
    type: string
    description: In the case the PaaS Service is versioned, it specifies the
preferred version of the requested PaaS Service.
    required: false
  usage_format:
    type: string
    description: Describes the usage format of the PaaS Service, as specified in
[2].
    required: true
  constraints:
    - valid_values: [ vnf_common, vnf_dedicated, ns, undefined ]
  specific_parameters:
    type: map
    description: Provides attributes specific to the type of the PaaS Service. This
property is reserved for future use in the present document.
    required: false
  entry_schema:
    type: string
  capabilities:
    ns_paas_service_requesting:
      type: toska.capabilities.nfv.NsPaasServiceRequestable
      occurrences: [ 1, UNBOUNDED ]

```

7.8.9.7 Artifact

None.

7.8.9.8 Additional requirements

None.

7.8.9.9 Example

None.

7.8.10 toska.nodes.nfv.NsPaasServiceProfile

7.8.10.1 Description

The NsPaasServiceProfile node type describes how many instances of a PaaS Service is requested, as well as the dependencies between the NS constituents and the requested PaaS Service as specified in ETSI GS NFV-IFA 014 [2].

Table 7.8.10.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.8.10.1-1: Type name, shorthand, and URI

Shorthand Name	NsPaasServiceProfile
Type Qualified Name	tosca.nfv.NsPaasServiceProfile
Type URI	tosca.nodes.nfv.NsPaasServiceProfile

7.8.10.2 Properties

The properties of the NsPaasServiceProfile node type shall comply with the provisions set out in table 7.8.10.2-1.

Table 7.8.10.2-1: Properties

Name	Required	Type	Constraints	Description
number_of_instances	yes	integer	greater_or_equal: 0	Number of instances of the PaaS Service that is requested for this profile.

7.8.10.3 Attributes

None.

7.8.10.4 Requirements

The requirements on the NsPaasServiceProfile node type shall comply with the provisions set out in table 7.8.10.4-1.

Table 7.8.10.4-1: Requirements

Name	Required	Capability type	Constraints	Description
ns_paas_service_request	yes	tosca.capabilities.nfv.NsPaasServiceRequestable		Describes the requirement for requesting PaaS Service.

7.8.10.5 Capabilities

The capabilities of the NsPaasServiceProfile node type shall comply with the provisions set out in table 7.8.10.5-1.

Table 7.8.10.5-1: Capabilities

Name	Type	Constraints	Description
associable_paas_service	tosca.capabilities.nfv.AssociablePaasService		Defines ability of AssociablePaasService

7.8.10.6 Definition

The syntax of the `NsPaasServiceProfile` node type shall comply with the following definition:

```
tosca.nodes.nfv.NsPaasServiceProfile:
  derived_from: toska.nodes.Root
  description: Describes how many instances of a PaaS Service is requested, as well
as the dependencies between the NS constituents and the requested PaaS Service as
defined in ETSI GS NFV-IFA 014.
  properties:
    number_of_instances:
      type: integer
      description: Number of instances of the PaaS Service that is requested for
this profile.
      required: true
      constraints:
        - greater_or_equal: 0
  requirements:
    - ns_paas_service_request:
      capability: toska.capabilities.nfv.NsPaasServiceRequestable
      relationship: toska.relationships.nfv.NsPaasServiceRequestsTo
      occurrences: [ 1, 1]
  capabilities:
    associable_paas_service:
      type: toska.capabilities.nfv.AssociablePaasService
      occurrences: [ 1, UNBOUNDED ]
```

7.8.10.7 Additional requirements

None.

7.8.10.8 Example

None.

7.8.11 toska.nodes.nfv.NsVirtualLinkAdditionalCapability

7.8.11.1 Description

The `NsVirtualLinkAdditionalCapability` specifies additional capabilities and/or services for an NS VL. Table 7.8.11.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.8.11.1-1: Type name, shorthand, and URI

Shorthand Name	<code>NsVirtualLinkAdditionalCapability</code>
Type Qualified Name	<code>toscanfv:NsVirtualLinkAdditionalCapability</code>
Type URI	<code>tosca.node.nfv.NsVirtualLinkAdditionalCapability</code>

7.8.11.2 Properties

The properties of the `NsVirtualLinkAdditionalCapability` node type shall comply with the provisions set out in table 7.8.11.2-1.

Table 7.8.11.2-1: Properties

Name	Required	Type	Constraints	Description
capability_type	yes	string		Defines the type of VL additional capability. Examples of additional capabilities include "name resolution" and "load-balancing".
capability_configurable_parameters	no	map of string		Defines the configurable parameters related to the VL additional capability. See note.
NOTE: Examples of parameterization can include a) for a "name resolution" capability, IP address of the naming service and name resolution entries (FQDN) associated to CPs in the NS b) for a "load-balancing" capability references to identifiers of CPs in the NS (e.g. a VNF external CP) and the rules of traffic to load balance.				

7.8.11.3 Attributes

None.

7.8.11.4 Requirements

The requirements on the NsVirtualLinkAdditionalCapability node type shall comply with the provisions set out in table 7.8.11.4-1.

Table 7.8.11.4-1: Requirements

Name	Required	Capability type	Constraints	Description
ns_paas_service_request	no	tosca.capabilities.nfv.NsPaasServiceRequestable		Describes the requirement for requesting PaaS Service.

7.8.11.5 Capabilities

The capabilities of the NsVirtualLinkAdditionalCapability node type shall comply with the provisions set out in table 7.8.11.5-1.

Table 7.8.11.5-1: Capabilities

Name	Type	Constraints	Description
virtual_link_capability_configurable	tosca.capabilities.nfv.VirtualLinkCapabilityConfigurable		VirtualLinkCapabilityConfigurable capability

7.8.11.6 Definition

The syntax of the NsVirtualLinkAdditionalCapability node type shall comply with the following definition:

```

tosca.nodes.nfv.NsVirtualLinkAdditionalCapability:
  derived_from: tosca.nodes.Root
  description: specifies additional capabilities and/or services for an NS VL.
  properties:
    capability_type:
      type: string
      description: Defines the type of VL additional capability.
      required: true
    capability_configurable_parameters:
      type: map
      description: Defines the configurable parameters related to the VL.
      required: false
    entry_schema: string

```



```

capabilities:
  virtual_link_capability_configurable:
    type: tosca.capabilities.nfv.VirtualLinkCapabilityConfigurable
requirements:
  - ns_paas_service_request:
    capability: tosca.capabilities.nfv.NsPaasServiceRequestable
    relationship: tosca.relationships.nfv.NsPaasServiceRequestsTo
    occurrences: [ 0, 1]

```

7.8.11.7 Artifact

None.

7.8.11.8 Additional requirements

None.

7.8.11.9 Example

None.

7.9 Group Types

7.9.1 tosca.groups.nfv.NsPlacementGroup

7.9.1.1 Description

The NsPlacementGroup group type is used for describing the affinity or anti-affinity relationship applicable between VNF instances created using different VNFDs, Virtual Link instances created using different VLDs, nested NS instances created using different NSDs when used in an NSD or connections created using different NsVirtualLinkConnectivity information elements as described in ETSI GS NFV-IFA 014 [2].

NOTE: The mapping between ETSI GS NFV-IFA 014 [2] and TOSCA concept is illustrated in clause A.9.3.

Table 7.9.1.1-1 specifies the declared names for this group type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.9.1.1-1: Type name, shorthand, and URI

Shorthand Name	NsPlacementGroup
Type Qualified Name	toscanfv:NsPlacementGroup
Type URI	tosca.groups.nfv.NsPlacementGroup

7.9.1.2 Properties

The properties of the NsPlacementGroup group type shall comply with the provisions set out in table 7.9.1.2-1.

Table 7.9.1.2-1: Properties

Name	Required	Type	Constraints	Description
description	yes	string		Human readable description of the group

7.9.1.3 Definition

The syntax of the `NsPlacementGroup` group type shall comply with the following definition:

```
tosca.groups.nfv.NsPlacementGroup:
  derived_from: toska.groups.Root
  description: NsPlacementGroup is used for describing the affinity or anti-affinity
  relationship applicable between VNF instances created using different VNFDs, the
  Virtual Link instances created using different VLDs or the nested NS instances created
  using different NSDs when used in a NSD.
  properties:
    description:
      type: string
      description: Human readable description of the group
      required: true
  members: [tosca.nodes.nfv.VNF, toska.nodes.nfv.NsVirtualLink, toska.nodes.nfv.NS,
  toska.nodes.nfv.Forwarding, toska.nodes.nfv.Sap ]
```

7.9.1.4 Additional Requirements

A group with type `tosca.groups.nfv.NsPlacementGroup` shall contain more than one member when used as the target of an `AffinityRule` or `AntiAffinityRule` policy with the following rules:

- all of the same node type as defined as members in clause 7.9.1.3 except `tosca.nodes.nfv.Sap`; or
- a combination of `tosca.nodes.nfv.VNF` and `tosca.nodes.nfv.NS` node types; or
- a combination of `tosca.nodes.nfv.Forwarding` and `tosca.nodes.nfv.Sap` node types.

7.9.2 toska.groups.nfv.VNFFG

7.9.2.1 Description

The VNF Forwarding Graph (VNFFG) group type describes a topology of the NS or a portion of the NS and optionally forwarding rules, applicable to the traffic conveyed over this topology, as defined by ETSI GS NFV-IFA 014 [2].

Table 7.9.2.1-1 specifies the declared names for this group type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.9.2.1-1: Type name, shorthand, and URI

Shorthand Name	VNFFG
Type Qualified Name	toscanfv:VNFFG
Type URI	tosca.groups.nfv.VNFFG

7.9.2.2 Properties

The properties of the VNFFG group type shall comply with the provisions set out in table 7.9.2.2-1.

Table 7.9.2.2-1: properties

Name	Required	Type	Constraints	Description
description	yes	string		Human readable description of the group

7.9.2.3 Definition

The syntax of the VNFFG group type shall comply with the following definition:

```
tosca.groups.nfv.VNFFG:
  derived_from: toska.groups.Root
  description: the VNFFG group type describes a topology of the NS or a portion of
the NS, and optionally forwarding rules, applicable to the traffic conveyed over
this topology
  properties:
    description:
      type: string
      description: Human readable description of the group
      required: true
    members: [ toska.nodes.nfv.NFP, toska.nodes.nfv.VNF, toska.nodes.nfv.PNF,
tosca.nodes.nfv.NS, toska.nodes.nfv.NsVirtualLink,
tosca.nodes.nfv.NfpPositionElement ]
```

7.9.2.4 Additional Requirements

None.

7.9.2.5 Example

See clause A.14.

7.10 Policy Types

7.10.1 NsAffinityRule, NsAntiAffinityRule

7.10.1.1 Description

The NsAffinityRule and NsAntiAffinityRule policy describes the affinity or anti-affinity rules applicable for the defined target.

If there is only one node template with node type toska.nodes.nfv.VNF or toska.nodes.nfv.NsVirtualLink or toska.nodes.nfv.NS set as the targets, the NsAffinityRule or NsAntiAffinityRule applies between the instances to be created based on the same VNFD, or between VLs to be created based on the same VLD, or between nested NS instances to be created based on the same NSD, as described in ETSI GS NFV-IFA 014 [2].

If there are more than one node templates with node type toska.nodes.nfv.VNF or toska.nodes.nfv.NsVirtualLink or toska.nodes.nfv.NS or toska.nodes.nfv.Sap or toska.nodes.nfv.Forwarding set as the targets, or a group with type toska.groups.nfv.PlacementGroup which contains more than one member set as the targets, the NsAffinityRule or NsAntiAffinityRule applies between VNF instances created using different VNFDs, Virtual Link instances created using different VLDs, nested NS instances created using different NSDs, or connections created using NsVirtualLinkConnectivity information elements as described in ETSI GS NFV-IFA 014 [2].

NOTE: The mapping between ETSI GS NFV-IFA 014 [2] and TOSCA concept is illustrated in clause A.9.3.

Tables 7.10.1.1-1 and 7.10.1.1-2 specify the declared names for the policy types. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.1.1-1: Type name, shorthand, and URI

Shorthand Name	NsAffinityRule
Type Qualified Name	toscanfv:NsAffinityRule
Type URI	tosca.policies.nfv.NsAffinityRule

Table 7.10.1.1-2: Type name, shorthand, and URI

Shorthand Name	NsAntiAffinityRule
Type Qualified Name	toscanfv:NsAntiAffinityRule
Type URI	tosca.policies.nfv.NsAntiAffinityRule

7.10.1.2 Properties

The properties of the NsAffinityRule and NsAntiAffinityRule types shall comply with the provisions set out in table 7.10.1.2-1.

Table 7.10.1.2-1: Properties

Name	Required	Type	Constraints	Description
scope	Yes	String	Valid values : "nfvi_pop", "zone", "zone_group", "nfvi_node", "network_link_and_node", "container_namespace", "l2_network".	Specifies the scope of the local affinity rule. See notes 1, 2, 3 and 4.
NOTE 1: When used in an anti-affinity relationship, the "network_link_and_node" scope is conceptually similar to link and node disjoint paths capabilities used commonly in network Traffic Engineering (TE). For example, as in Fast Reroute Resource Reservation Protocol Traffic Engineering (RSVP-TE) for Label-Switched Path (LSP) tunnels as introduced in IETF RFC 4090 [i.17].				
NOTE 2: The "container_namespace" is only applicable when the targets of the policy are exclusively nodes of type toasca.nodes.nfv.VNF and these are realized exclusively by sets of OS containers. Groups of type toasca.groups.nfv.NsPlacementGroup that consists exclusively of VNFs realized exclusively by set of OS containers may also be target of the policy when the "container_namespace" scope is indicated.				
NOTE 3: When used in an affinity or anti-affinity relationship, the "l2_network" scope represents that the targets belonging to the same policy shall or shall not, respectively, be connected to the same layer 2 network.				
NOTE 4: The "l2_network" scope shall only be used when targets are toasca.nodes.nfv.Sap and/or toasca.nodes.nfv.Forwarding.				

7.10.1.3 Targets

The targets of the NsAffinityRule and NsAntiAffinityRule policy types shall comply with the provisions set out in table 7.10.1.3-1 when used in an NSD.

Table 7.10.1.3-1: Targets

Name	Required	Type	Constraints	Description
targets	Yes	string[]	Valid types: tosca.nodes.nfv.VNF, tosca.nodes.nfv.NsVirtualLink, tosca.nodes.nfv.NS, tosca.groups.nfv.NsPlacementGroup, tosca.nodes.nfv.Sap, tosca.nodes.nfv.Forwarding, tosca.nodes.nfv.NsPaasServiceRequest	In case of LocalAffinityOrAntiAffinityRule as defined in ETSI GS NFV-IFA 014 [2], the valid type of the targets is toasca.nodes.nfv.VNF or toasca.nodes.nfv.NsVirtualLink, or toasca.nodes.nfv.NS, or toasca.nodes.nfv.NsPaasServiceRequest. In case of affinityOrAntiAffinityGroup as defined in ETSI GS NFV-IFA 014 [2], the valid types of the targets are: toasca.nodes.nfv.VNF, toasca.nodes.nfv.NsVirtualLink and toasca.nodes.nfv.NS or a toasca.groups.nfv.NsPlacementGroup or toasca.nodes.nfv.Sap or toasca.nodes.nfv.Forwarding or toasca.nodes.nfv.NsPaasServiceRequest.

7.10.1.4 Definition

The syntax of the `NsAffinityRule` policy type shall comply with the following definition:

```
tosca.policies.nfv.NsAffinityRule:
  derived_from: tosca.policies.Placement
  description: The NsAffinityRule describes the affinity rules applicable for the
defined targets
  properties:
    scope:
      type: string
      description: Specifies the scope of the local affinity rule.
      required: true
      constraints:
        - valid_values: [ nfvi_node, zone, zone_group, nfvi_pop,
network_link_and_node, container_namespace, l2_network ]
    targets: [tosca.nodes.nfv.VNF, tosca.nodes.nfv.NsVirtualLink, tosca.nodes.nfv.NS,
tosca.groups.nfv.NsPlacementGroup, tosca.nodes.nfv.Sap, tosca.nodes.nfv.Forwarding,
tosca.nodes.nfv.NsPaasServiceRequest ]
```

The syntax of the `NsAntiAffinityRule` policy type shall comply with the following definition:

```
tosca.policies.nfv.NsAntiAffinityRule:
  derived_from: tosca.policies.Placement
  description: The NsAntiAffinityRule describes the anti-affinity rules applicable
for the defined targets
  properties:
    scope:
      type: string
      description: Specifies the scope of the local affinity rule..
      required: true
      constraints:
        - valid_values: [ nfvi_node, zone, zone_group, nfvi_pop,
network_link_and_node, container_namespace, l2_network ]
    targets: [tosca.nodes.nfv.VNF, tosca.nodes.nfv.NsVirtualLink, tosca.nodes.nfv.NS,
tosca.groups.nfv.NsPlacementGroup, tosca.nodes.nfv.Sap, tosca.nodes.nfv.Forwarding,
tosca.nodes.nfv.NsPaasServiceRequest ]
```

7.10.1.5 Examples

The following example template fragments illustrate the concepts:

```
node_templates:
  VNF_1:
    type: tosca.nodes.nfv.exampleVNF

policies:
  policy_affinity_local_VNF_1:
    type: tosca.policies.nfv.NsAffinityRule
    targets: [ VNF_1 ]
    properties:
      scope: nfvi_node
```

The above example illustrates a local affinity rule for all the instances of `VNF_1`.

```
node_template:
  VNF_1:
    type: tosca.nodes.nfv.exampleVNF_1
```

```

VNF_2:
  type: toska.nodes.nfv.exampleVNF_2

groups:
  affinityOrAntiAffinityGroup_1:
    type: toska.groups.nfv.NsPlacementGroup
    members: [ VNF_1, VNF_2 ]

policies:
  policy_antiaffinity_group_1:
    type: toska.policies.nfv.NsAntiAffinityRule
    targets: [ affinityOrAntiAffinityGroup_1 ]
    properties:
      scope: nfvi_node

```

The above example illustrates an anti-affinity policy among a group which contains VNF_1 and VNF_2 as members.

7.10.2 toska.policies.nfv.NsSecurityGroupRule

7.10.2.1 Description

The NsSecurityGroupRule policy type when used in an NSD specifies the matching criteria for the ingress and/or egress traffic to and from visited SAPs. Table 7.10.2.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.2.1-1: Type name, shorthand, and URI

Shorthand Name	NsSecurityGroupRule
Type Qualified Name	tosca.nfv.NsSecurityGroupRule
Type URI	tosca.policies.nfv.NsSecurityGroupRule

7.10.2.2 Properties

None.

7.10.2.3 targets

The targets of the SecurityGroupRule policy types shall comply with the provisions set out in table 7.10.2.3-1.

Table 7.10.2.3-1: Targets

Name	Required	Type	Constraints	Description
targets	yes	string[]	Valid types: tosca.nodes.nfv.Sap.	Target connection points of Sap.

7.10.2.4 Definition

The syntax of the NsSecurityGroupRule policy type shall comply with the following definition:

```

tosca.policies.nfv.NsSecurityGroupRule:
  derived_from: toska.policies.nfv.Abstract.SecurityGroupRule
  description: The NsSecurityGroupRule type is a policy type specified the
  matching criteria for the ingress and/or egress traffic to/from visited SAPs.
  targets: [ toska.nodes.nfv.Sap ]

```

7.10.2.5 Additional Requirements

None.

7.10.3 `tosca.policies.nfv.NfpRule`

7.10.3.1 Description

The `NfpRule` policy type represents the NFP rule attribute of the `Nfpd` information element as defined in ETSI GS NFV-IFA 014 [2], which describes the conditions that shall be met in order for the NFP to be applicable to the packet. Table 7.10.3.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.3.1-1: Type name, shorthand, and URI

Shorthand Name	<code>NfpRule</code>
Type Qualified Name	<code>tosca.nfv.NfpRule</code>
Type URI	<code>tosca.policies.nfv.NfpRule</code>

7.10.3.2 Properties

The properties of the NFP policy type shall comply with the provisions set out in table 7.10.3.2-1.

Table 7.10.3.2-1: Properties

Name	Required	Type	Constraints	Description
ether_destination_address	no	string		Indicates a destination Mac address.
ether_source_address	no	string		Indicates a source Mac address.
ether_type	no	string	ipv4, ipv6	Indicates the protocol carried over the Ethernet layer.
vlan_tag	no	list of string		Indicates a VLAN identifier in an IEEE 802.1Q-2018 tag [16]. Multiple tags can be included for QinQ stacking.
protocol	no	string		Indicates the L4 protocol, For IPv4 [17] this corresponds to the field called "Protocol" to identify the next level protocol. For IPv6 [18] this corresponds to the field called the "Next Header" field. Permitted values: Any keyword defined in the IANA [19] protocol registry, e.g.: • TCP • UDP • ICMP
dscp	no	string		For IPv4 [17] a string of "0" and "1" digits that corresponds to the 6-bit Differentiated Services Code Point (DSCP) field of the IP header. For IPv6 [18] a string of "0" and "1" digits that corresponds to the 6 differentiated services bits of the traffic class header field.
source_port_range	no	range	0 - 65 535	Indicates a range of source ports.
destination_port_range	no	range	0 - 65 535	Indicates a range of destination ports.
source_ip_address_prefix	no	string		Indicates the source IP address range in CIDR format.
destination_ip_address_prefix	no	string		Indicates the destination IP address range in CIDR format.
extended_criteria	no	list of tosca.data types.nfv.Mask		Indicates values of specific bits in a frame.

7.10.3.3 Targets

The targets of the NfpRule policy types shall comply with the provisions set out in table 7.10.3.3-1.

Table 7.10.3.3-1: Targets

Name	Required	Type	Constraints	Description
targets	yes	string[]	Valid types: tosca.nodes.nfv.NFP.	The NFPs to which the rule applies.

7.10.3.4 Definition

The syntax of the NfpRule policy type shall comply with the following definition:

```
tosca.policies.nfv.NfpRule:
  derived_from: tosca.policies.Root
  description: policy definition of NfpRule
  properties:
    ether_destination_address:
      description: Indicates a destination Mac address.
      type: string
      required: false
```



```

ether_source_address:
  description: Indicates a source Mac address.
  type: string
  required: false
ether_type:
  description: Indicates the protocol carried over the Ethernet layer.
  type: string
  constraints:
    - valid_values: [ ipv4, ipv6 ]
  required: false
vlan_tag:
  description: Indicates a VLAN identifier in an IEEE 802.1Q-2018 tag.
Multiple tags can be included for QinQ stacking.
  type: list
  entry_schema:
    type: string
  required: false
protocol:
  description: 'Indicates the L4 protocol, For IPv4 this corresponds to the
field called "Protocol" to identify the next level protocol. For IPv6 this
corresponds to the field is called the "Next Header" field. Permitted values: Any
keyword defined in the IANA protocol registry.'
  type: string
  required: false
dscp:
  description: For IPv4 a string of "0" and "1" digits that corresponds to
the 6-bit Differentiated Services Code Point (DSCP) field of the IP header. For
IPv6 [18] a string of "0" and "1" digits that corresponds to the 6 differentiated
services bits of the traffic class header field.
  type: string
  required: false
source_port_range:
  description: Indicates a range of source ports.
  type: range
  required: false
  constraints:
    - in_range: [ 0, 65535 ]
destination_port_range:
  description: Indicates a range of destination ports.
  type: range
  required: false
  constraints:
    - in_range: [ 0, 65535 ]
source_ip_address_prefix:
  description: Indicates the source IP address range in CIDR format.
  type: string
  required: false
destination_ip_address_prefix:
  description: Indicates the destination IP address range in CIDR format.
  type: string
  required: false
extended_criteria:
  description: Indicates values of specific bits in a frame.
  type: list
  entry_schema:
    type: toasca.datatypes.nfv.Mask
  required: false
targets: [ toasca.nodes.nfv.NFP ]

```

7.10.3.5 Example

None.

7.10.4 toska.policies.nfv.NsMonitoring

7.10.4.1 Description

The NsMonitoring policy type is a policy type representing the virtualised resource related performance metrics to be monitored during the lifetime of network service instance as defined in ETSI GS NFV-IFA 014 [2] and ETSI GS NFV-IFA 027 [7]. Table 7.10.4.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.4.1-1: Type name, shorthand, and URI

Shorthand Name	NsMonitoring
Type Qualified Name	toscanfv:NsMonitoring
Type URI	tosca.policies.nfv.NsMonitoring

7.10.4.2 Properties

The properties of the NsMonitoring policy type shall comply with the provisions set out in table 7.10.4.2-1.

Table 7.10.4.2-1: Properties

Name	Required	Type	Constraints	Description
ns_monitoring_parameters	yes	map of toska.datatype.nfv.NsMonitoringParameter		Specifies a virtualised resource related performance metric to be monitored on the NS level.

7.10.4.3 targets

The targets of the NsMonitoring policy types shall comply with the provisions set out in table 7.10.4.3-1.

Table 7.10.4.3-1: Targets

Name	Required	Type	Constraints	Description
targets	yes	string	Valid types: toska.nodes.nfv.NS	Specifies the services node type(s) to which the monitoring policy applies.

7.10.4.4 Definition

The syntax of the NsMonitoring policy type shall comply with the following definition:

```
tosca.policies.nfv.NsMonitoring:
  derived_from: toska.policies.Root
  description: Policy type is used to identify information to be monitored
during the lifetime of a network service instance as defined in ETSI
GS NFV-IFA 014.
  properties:
    ns_monitoring_parameters:
      type: map #key: id
      description: Specifies a virtualised resource related performance metric
to be monitored on the NS level.
      required: true
```

```

entry_schema:
  type: toska.datatypes.nfv.NsMonitoringParameter
constraints:
  - min_length: 1
targets: [ toska.nodes.nfv.NS ]

```

7.10.4.5 Additional Requirements

When a policy of this type is specified in an NS service template, the targets set shall only include NS node template names that correspond to this NS or to a nested NS.

7.10.5 toska.policies.nfv.VnfMonitoring

7.10.5.1 Description

The VnfMonitoring policy type is a policy type representing the virtualised resource related performance metrics to be monitored during the lifetime of VNF instance as defined in ETSI GS NFV-IFA 014 [2] and ETSI GS NFV-IFA 027 [7]. Table 7.10.5.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.5.1-1: Type name, shorthand, and URI

Shorthand Name	VnfMonitoring
Type Qualified Name	toscanfv:VnfMonitoring
Type URI	tosca.policies.nfv.VnfMonitoring

7.10.5.2 Properties

The properties of the VnfMonitoring policy type shall comply with the provisions set out in table 7.10.5.2-1.

Table 7.10.5.2-1: Properties

Name	Required	Type	Constraints	Description
vnf_monitoring_parameters	yes	map of toska.datatypes.nfv.VnfMonitoringParameter		Specifies a virtualised resource related performance metric to be monitored on the VNF level.

7.10.5.3 targets

The targets of the VnfMonitoring policy types shall comply with the provisions set out in table 7.10.5.3-1.

Table 7.10.5.3-1: Targets

Name	Required	Type	Constraints	Description
targets	yes	string	Valid types: toska.nodes.nfv.VNF	Specifies the VNF node type(s) to which the monitoring policy applies.

7.10.5.4 Definition

The syntax of the VnfMonitoring policy type shall comply with the following definition:

```

tosca.policies.nfv.VnfMonitoring:
  derived_from: toska.policies.Root

```

```

description: Policy type is used to identify information to be monitored
during the lifetime of a VNF instance as defined in ETSI GS NFV-IFA 014.
properties:
  vnf_monitoring_parameters:
    type: map #key: id
    description: Specifies a virtualised resource related performance metric
to be monitored on the NS level.
    required: true
    entry_schema:
      type: toska.datatypes.nfv.VnfMonitoringParameter
    constraints:
      - min_length: 1
  targets: [ toska.nodes.nfv.VNF ]

```

7.10.5.5 Additional Requirements

When a policy of this type is specified in an NS service template, the targets set shall only include VNF node template names representing constituent VNFs for the NS deployment flavour corresponding to this NS.

7.10.6 toska.policies.nfv.Abstract.SecurityGroupRule

7.10.6.1 Description

The Abstract.SecurityGroupRule policy type is defined in clause 9.10.1 of the present document.

7.10.7 toska.policies.nfv.NsScalingAspects

7.10.7.1 Description

The NsScalingAspects type is a policy type representing the scaling aspects used for horizontal scaling as defined in ETSI GS NFV-IFA 014 [2]. This policy concerns the whole NS (deployment flavour) represented by the topology_template and thus has no explicit target list. Table 7.10.7.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.7.1-1: Type name, shorthand, and URI

Shorthand Name	NsScalingAspects
Type Qualified Name	toscanfv:NsScalingAspects
Type URI	tosca.policies.nfv.NsScalingAspects

7.10.7.2 Properties

The properties of the NsScalingAspects policy type shall comply with the provisions set out in table 7.10.7.2-1.

Table 7.10.7.2-1: Properties

Name	Required	Type	Constraints	Description
aspects	yes	Map of tosca.datatypes.nfv. NsScalingAspect		Describe the details of a particular aspect including the corresponding NS levels.

7.10.7.3 Definition

The syntax of the NsScalingAspects policy type shall comply with the following definition:

```

tosca.policies.nfv.NsScalingAspects:
  derived_from: toska.policies.Root
  description: The NsScalingAspects type is a policy type representing the scaling
aspects used for horizontal scaling as defined in ETSI GS NFV-IFA 014
  properties:
    aspects:
      type: map # key: aspectId
      description: Describe the details of a particular aspect including the
corresponding NS levels.
      required: true
      entry_schema:
        type: toska.datatypes.nfv.NsScalingAspect
      constraints:
        - min_length: 1

```

7.10.7.4 Examples

See clause A.17.

7.10.8 toska.policies.nfv.VnfToLevelMapping

7.10.8.1 Description

The VnfToLevelMapping type is a policy type representing the number of VNF instances that correspond to every NS level of a particular aspect, as defined in ETSI GS NFV-IFA 014 [2]. Table 7.10.8.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.8.1-1: Type name, shorthand, and URI

Shorthand Name	VnfToLevelMapping
Type Qualified Name	toscanfv: VnfToLevelMapping
Type URI	tosca.policies.nfv.VnfToLevelMapping

7.10.8.2 Properties

The properties of the VnfToLevelMapping policy type shall comply with the provisions set out in table 7.10.8.2-1.

Table 7.10.8.2-1: Properties

Name	Required	Type	Constraints	Description
aspect:	yes	string		Represents the scaling aspect to which this policy applies.
number_of_instances	yes	map of integer		Describes the number of VNF instances to be deployed for each NS level of a particular aspect. The first level is level 0.

7.10.8.3 Definition

The syntax of the VnfToLevelMapping policy type shall comply with the following definition:

```

tosca.policies.nfv.VnfToLevelMapping:
  derived_from: tosca.policies.Root
  description: The VnfToLevelMapping type is a policy type representing the number
of VNF instances to be deployed at each NS level, as defined in ETSI GS NFV-IFA 014
  properties:
    aspect:
      type: string
      description: Represents the scaling aspect to which this policy applies
      required: true
    number_of_instances:
      type: map # key: Ns level
      description: Number of VNF instances to be deployed for each NS level.
      required: true
      key_schema:
        type: integer # First level is level 0.
        constraints:
          - greater_or_equal: 0
      entry_schema:
        type: integer
        constraints:
          - greater_or_equal: 0
      constraints:
        - min_length: 1
  targets: [ tosca.nodes.nfv.VNF ]

```

7.10.8.4 Examples

See clause A.17.

7.10.9 tosca.policies.nfv.NsToLevelMapping

7.10.9.1 Description

The NsToLevelMapping type is a policy type representing the number of instances of a nested NS that correspond to every NS level of the composite NS of a particular aspect, as defined in ETSI GS NFV-IFA 014 [2]. Table 7.10.9.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.9.1-1: Type name, shorthand, and URI

Shorthand Name	NsToLevelMapping
Type Qualified Name	toscanfv: NsToLevelMapping
Type URI	tosca.policies.nfv.NsToLevelMapping

7.10.9.2 Properties

The properties of the NsToLevelMapping policy type shall comply with the provisions set out in table 7.10.9.2-1.

Table 7.10.9.2-1: Properties

Name	Required	Type	Constraints	Description
aspect:	yes	string		Represents the scaling aspect to which this policy applies.
number_of_instances	yes	map of integer		Describes the number of NS instances of a nested NS to be deployed for each NS level of the composite NS of a particular aspect. The first level is level 0.

7.10.9.3 Definition

The syntax of the NsToLevelMapping policy type shall comply with the following definition:

```

tosca.policies.nfv.NsToLevelMapping:
  derived_from: toscapolicies.Root
  description: The NsToLevelMapping type is a policy type representing the number of
  NS instances of a nested NS to be deployed at each NS level of the composite NS, as
  defined in ETSI GS NFV-IFA 014
  properties:
    aspect:
      type: string
      description: Represents the scaling aspect to which this policy applies
      required: true
    number_of_instances:
      type: map # key: Ns level
      description: Number of NS instances of a nested NS to be deployed for each NS
  level of the composite NS.
      required: true
      key_schema:
        type: integer # First level is level 0.
        constraints:
          - greater_or_equal: 0
      entry_schema:
        type: integer
        constraints:
          - greater_or_equal: 0
      constraints:
        - min_length: 1
  targets: [ toscanodes.nfv.NS ]

```

7.10.9.4 Examples

See clause A.17.

7.10.10 toscapolicies.nfv.VirtualLinkToLevelMapping

7.10.10.1 Description

The VirtualLinkToLevelMapping type is a policy type representing the bitrate requirements of a VL for every NS level of a particular aspect, as defined in ETSI GS NFV-IFA 014 [2]. Table 7.10.10.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.10.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualLinkToLevelMapping
Type Qualified Name	toscanfv: VirtualLinkToLevelMapping
Type URI	tosca.policies.nfv.VirtualLinkToLevelMapping

7.10.10.2 Properties

The properties of the VirtualLinkToLevelMapping policy type shall comply with the provisions set out in table 7.10.10.2-1.

Table 7.10.10.2-1: Properties

Name	Required	Type	Constraints	Description
aspect:	yes	string		Represents the scaling aspect to which this policy applies.
bit_rate_requirements	yes	map of tosca.datatypes .nfv. LinkBitrateRequirements		Describes the bitrate requirements of a VL for each NS level of a particular aspect. First level is level 0.

7.10.10.3 Definition

The syntax of the VirtualLinkToLevelMapping policy type shall comply with the following definition:

```
tosca.policies.nfv.VirtualLinkToLevelMapping:
  derived_from: tosca.policies.Root
  description: The VirtualLinkToLevelMapping type is a policy type representing the
number of NS instances of a nested NS to be deployed at each NS level of the composite
NS, as defined in ETSI GS NFV-IFA 014
  properties:
    aspect:
      type: string
      description: Represents the scaling aspect to which this policy applies
      required: true
    bit_rate_requirements:
      type: map # key: Ns level
      description: Bitrate requirements of a VL for each NS level.
      required: true
    key_schema:
      type: integer # First level is level 0.
      constraints:
        - greater_or_equal: 0
    entry_schema:
      type: tosca.datatypes.nfv.LinkBitrateRequirements
      constraints:
        - min_length: 1
  targets: [ tosca.nodes.nfv.NsVirtualLink ]
```

7.10.10.4 Examples

See clause A.17.

7.10.11 tosca.policies.nfv.NsInstantiationLevels

7.10.11.1 Description

The NsInstantiationLevels type is a policy type representing all the instantiation levels of resources to be instantiated within a deployment flavour and including default instantiation level in term of the number of VNF instances and nested NS instances to be created as defined in ETSI GS NFV-IFA 014 [2]. This policy concerns the whole NS(deployment flavour) represented by the topology_template and thus has no explicit target list. Table 7.10.11.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.11.1-1: Type name, shorthand, and URI

Shorthand Name	NsInstantiationLevels
Type Qualified Name	tosca.nfv.NsInstantiationLevels
Type URI	tosca.policies.nfv.NsInstantiationLevels

7.10.11.2 Properties

The properties of the NsInstantiationLevels policy type shall comply with the provisions set out in table 7.10.11.2-1.

Table 7.10.11.2-1: Properties

Name	Required	Type	Constraints	Description
ns_levels	yes	map of tosca.datatypes.nfv.NsLevels		Describes the various levels of resources that can be used to instantiate the NS using this flavour.
default_level	no	string		The default instantiation level for this flavour.

7.10.11.3 Definition

The syntax of the NsInstantiationLevels policy type shall comply with the following definition:

```

tosca.policies.nfv.NsInstantiationLevels:
  derived_from: toska.policies.Root
  description: The NsInstantiationLevels type is a policy type representing all the
instantiation levels of resources to be instantiated within a deployment flavour and
including default instantiation level in term of the number of VNF and nested NS
instances to be created as defined in ETSI GS NFV-IFA 014.
  properties:
    ns_levels:
      type: map # key: levelId
      description: Describes the various levels of resources that can be used to
instantiate the VNF using this flavour.
      required: true
      entry_schema:
        type: toska.datatypes.nfv.NsLevels
      constraints:
        - min_length: 1
    default_level:
      type: string # levelId
      description: The default instantiation level for this flavour.
      required: false # required if multiple entries in ns_levels

```

7.10.12 toska.policies.nfv.VnfToInstantiationLevelMapping

7.10.12.1 Description

The VnfToInstantiationLevelMapping type is a policy type representing the number of VNF instances that correspond to every NS instantiation level, as defined in ETSI GS NFV-IFA 014 [2]. Table 7.10.12.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.12.1-1: Type name, shorthand, and URI

Shorthand Name	VnfToInstantiationLevelMapping
Type Qualified Name	toscanfv: VnfToInstantiationLevelMapping
Type URI	tosca.policies.nfv.VnfToInstantiationLevelMapping

7.10.12.2 Properties

The properties of the VnfToInstantiationLevelMapping policy type shall comply with the provisions set out in table 7.10.12.2-1.

Table 7.10.12.2-1: Properties

Name	Required	Type	Constraints	Description
number_of_instances	yes	map of integer		Describes the number of VNF instances to be deployed for each NS instantiation level.

7.10.12.3 Definition

The syntax of the VnfToInstantiationLevelMapping policy type shall comply with the following definition:

```

tosca.policies.nfv.VnfToInstantiationLevelMapping:
  derived_from: toscanfv.Root
  description: The VnfToInstantiationLevelMapping type is a policy type representing
the number of VNF instances to be deployed at each NS instantiation level, as defined
in ETSI GS NFV-IFA 014
  properties:
    number_of_instances:
      type: map # key: Ns instantiation level
      description: Number of VNF instances to be deployed for each NS instantiation
level.
      required: true
      entry_schema:
        type: integer
        constraints:
          - greater_or_equal: 0
      constraints:
        - min_length: 1
    targets: [ toscanfv.VNF ]

```

7.10.12.4 Examples

See clause A.17.

7.10.12.5 Additional requirements

There shall be one VnfToInstantiationLevelMapping policy defined for each VNF of the NS. If no instances of a given VNF have to be deployed at NS instantiation time for a certain instantiation level, the number_of_instances in the corresponding map entry shall be set to 0.

7.10.13 toscanfv.NsToInstantiationLevelMapping

7.10.13.1 Description

The NsToInstantiationLevelMapping type is a policy type representing the number of instances of a nested NS that correspond to every NS instantiation level of the composite NS, as defined in ETSI GS NFV-IFA 014 [2].

Table 7.10.13.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.13.1-1: Type name, shorthand, and URI

Shorthand Name	NsToInstantiationLevelMapping
Type Qualified Name	tosca.nfv: NsToInstantiationLevelMapping
Type URI	tosca.policies.nfv.NsToInstantiationLevelMapping

7.10.13.2 Properties

The properties of the NsToInstantiationLevelMapping policy type shall comply with the provisions set out in table 7.10.13.2-1.

Table 7.10.13.2-1: Properties

Name	Required	Type	Constraints	Description
number_of_instances	yes	map of integer		Describes the number of NS instances of a nested NS to be deployed for each NS instantiation level of the composite NS.

7.10.13.3 Definition

The syntax of the NsToInstantiationLevelMapping policy type shall comply with the following definition:

```
tosca.policies.nfv.NsToInstantiationLevelMapping:
  derived_from: toscapolicies.Root
  description: The NsToInstantiationLevelMapping type is a policy type representing
the number of NS instances of a nested NS to be deployed at each NS instantiation
level of the composite NS, as defined in ETSI GS NFV-IFA 014
  properties:
    number_of_instances:
      type: map # key: Ns instantiation level
      description: Number of NS instances of a nested NS to be deployed for each NS
instantiation level of the composite NS.
      required: true
      entry_schema:
        type: integer
        constraints:
          - greater_or_equal: 0
      constraints:
        - min_length: 1
  targets: [ toscanodes.nfv.NS ]
```

7.10.13.4 Examples

See clause A.17.

7.10.13.5 Additional requirements

There shall be one NsToInstantiationLevelMapping policy defined for each nested NS of the composite NS. If no instances of a given nested NS have to be deployed at NS instantiation time for a certain instantiation level, the number_of_instances in the corresponding map entry shall be set to 0.

7.10.14 tosca.policies.nfv.VirtualLinkToInstantiationLevelMapping

7.10.14.1 Description

The VirtualLinkToInstantiationLevelMapping type is a policy type representing the bitrate requirements of a VL for every NS instantiation level of a particular aspect, as defined in ETSI GS NFV-IFA 014 [2]. Table 7.10.14.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.14.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualLinkToInstantiationLevelMapping
Type Qualified Name	toscanfv: VirtualLinkToInstantiationLevelMapping
Type URI	tosca.policies.nfv.VirtualLinkToInstantiationLevelMapping

7.10.14.2 Properties

The properties of the VirtualLinkToInstantiationLevelMapping policy type shall comply with the provisions set out in table 7.10.14.2-1.

Table 7.10.14.2-1: Properties

Name	Required	Type	Constraints	Description
bit_rate_requirements	yes	map of tosca.datatypes .nfv. LinkBitrateRequirements		Describes the bitrate requirements of a VL for each NS instantiation level.

7.10.14.3 Definition

The syntax of the VirtualLinkToInstantiationLevelMapping policy type shall comply with the following definition:

```
tosca.policies.nfv.VirtualLinkToInstantiationLevelMapping:
  derived_from: tosca.policies.Root
  description: The VirtualLinkToInstantiationLevelMapping type is a policy type
  describing the bitrate requirements of a VL at each NS instantiation level of the
  composite NS, as defined in ETSI GS NFV-IFA 014
  properties:
    bit_rate_requirements:
      type: map # key: Ns instantiation level
      description: Bitrate requirements of a VL for each NS instantiation level.
      required: true
      entry_schema:
        type: tosca.datatypes.nfv.LinkBitrateRequirements
      constraints:
        - min_length: 1
  targets: [ tosca.nodes.nfv.NsVirtualLink ]
```

7.10.14.4 Examples

See clause A.17.

7.10.15 tosa.policies.nfv.NsAutoScale

7.10.15.1 Description

The NsAutoScale is a base policy type for defining NS auto-scale specific policies as defined in ETSI GS NFV-IFA 014 [2].

Table 7.10.15.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.15.1-1: Type name, shorthand, and URI

Shorthand Name	NsAutoScale
Type Qualified Name	toscanfv:NsAutoScale
Type URI	tosca.policies.nfv.NsAutoScale

7.10.15.2 Properties

None.

7.10.15.3 Definition

The syntax of the NsAutoScale policy type shall comply with the following definition:

```
tosca.policies.nfv.NsAutoScale:
  derived_from: tosa.policies.Root
  description: The NsAutoScale policy type is a base policy type for defining NS
  auto-scale specific policies as defined in ETSI GS NFV-IFA 014.
  targets: [ tosa.nodes.nfv.NS ]
```

7.10.15.4 Additional requirements

The NSD service template may include specific policies of NsAutoScale type with the following requirements:

- a) it shall include one or more trigger definitions which:
 - shall include an event with a value equal to the full name of a notification in the NsVnfIndicator interface definition of the NS node where the policy applies;
 - may include a condition definition which can assert the value of vnf indicator attributes and other node attributes using arbitrary AND and OR combinations of the individual assertions;
 - may include an action invoking one or multiple operations of the Nslcm interface;
- b) the target shall be set to the node template to which the policy applies, i.e. to the node template of the NS specific type present in the topology template that represents a particular deployment flavour.

7.10.16 tosa.policies.nfv.DataFlowInfo

7.10.16.1 Description

The DataFlowInfo type is a policy type representing the information to identify an external connection point of the VNF/PNF or a Sap from which the data flows are expected to be mirrored as defined in ETSI GS NFV-IFA 014 [2].

Table 7.10.16.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.16.1-1: Type name, shorthand, and URI

Shorthand Name	DataFlowInfo
Type Qualified Name	toscanfv>DataFlowInfo
Type URI	tosca.policies.nfv.DataFlowInfo

7.10.16.2 Properties

The properties of the DataFlowInfo policy type shall comply with the provisions set out in table 7.10.16.2-1.

Table 7.10.16.2-1: Properties

Name	Required	Type	Constraints	Description
data_flow_info_id	yes	string		Identifier of the Data flow information.
associated_mirroring_job_name	yes	string		Associated data mirroring job name as defined in NsVirtualLink node.
direction	yes	string	Valid values: See YAML definition constraints	The direction of the data flow that are requested to be mirrored. The direction is specified against the associated CP. I.e. 'in' means the data flow that enters the target connection point needs to be mirrored. 'out' means the data flow that sends out the target connection point needs to be mirrored. 'both' means the data flows that both enters and sends out the target connection point all need to be mirrored.

7.10.16.3 Definition

The syntax of the DataFlowInfo policy type shall comply with the following definition:

```

tosca.policies.nfv.DataFlowInfo:
  derived_from: toscanfv.Root
  description: The NsDataFlowMirroring type is a policy type representing the
information to identify an external connection point of the VNF/PNF or a Sap from
which the data flows are expected to be mirrored as defined in ETSI GS NFV-IFA 014.
  properties:
    data_flow_info_id:
      type: string
      description: Identifier of the Data flow information
      required: true
    associated_mirroring_job_name:
      type: string
      description: Associated data mirroring job name as defined in NsVirtualLink
node
      required: true
    direction:
      type: string
      description: The direction of the data flow that are requested to be
mirrored. The direction is specified against the associated CP. I.e. 'in' means the
data flow that enters the target connection point needs to be mirrored. 'out' means
the data flow that sends out the target connection point needs to be mirrored. 'both'
means the data flows that both enters and sends out the target connection point all
need to be mirrored.
      required: true
      constraints:
        - valid_values: [ in, out, both ]
    targets: [ toscanfv.Forwarding, toscanfv.Sap ]

```

7.10.16.4 Additional requirements

An NSD service template may contain a policy of this type to indicate the information of data flows that are expected to be mirrored, the targets shall include a Forwarding node template name representing external connection point of the constituent VNF/PNF or a Sap node template name in this NSD service template.

7.10.17 toska.policies.nfv.SapToDeployableModuleMapping

7.10.17.1 Description

The SapToDeployableModuleMapping is a policy type for defining how SAPs in an NSD relate to deployable modules of VNFs that are constituent of the NS defined by the NSD, or that are constituent of a nested NS of that NS, as defined in ETSI GS NFV-IFA 014 [2].

Table 7.10.17.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 7.10.17.1-1: Type name, shorthand, and URI

Shorthand Name	SapToDeployableModuleMapping
Type Qualified Name	toscanfv:SapToDeployableModuleMapping
Type URI	tosca.policies.nfv.SapToDeployableModuleMapping

7.10.17.2 Properties

The properties of the SapToDeployableModuleMapping policy type shall comply with the provisions set out in table 7.10.17.2-1.

Table 7.10.17.2-1: Properties

Name	Required	Type	Constraints	Description
sap_to_depl_mod_mapping	yes	map of list of String		Indicates the deployable modules to which the VduCp exposed by a SAP belongs. Each deployable module is indicated in an element of the list. If the policy targets a VNF node, it is the name of a deployable module of the VNF node. If the policy targets a nested NS node, the name of the deployable module is preceded by the name of the VNF node template in the nested NS that has the deployable module. If there is more than one level of NS nesting, the name of the VNF node template is preceded by the name(s) of the nested NS node template(s). A colon character ':' shall be used as separator between the VNF node template name and the deployable module name, and between node template names.

7.10.17.3 Definition

The syntax of the SapToDeployableModuleMapping policy type shall comply with the following definition:

```

tosca.policies.nfv.SapToDeployableModuleMapping:
  derived_from: toska.policies.Root
  description: The SapToDeployableModuleMapping policy type is a policy type for
    defining how SAPs in an NSD relate to deployable modules of VNFs that are constituent
    of the NS defined by the NSD, or that are constituent of a nested NS of that NS as
    defined in ETSI GS NFV-IFA 014.
  properties:
    sap_to_depl_mod_mapping:
      type: map # name of the virtual_link requirement in the NS node. Only SAPs
        that map to connection points in the target node that are part of a deployable module
        have to be listed.
      description: Indicates the deployable modules to which the VduCpd exposed by
        a SAP belongs, each deployable module in an element of the list.
      required: true
      constraints:
        - min_length: 1
      entry_schema:
        type: list
        entry_schema: string
        constraints:
          - min_length: 1
      targets: [ toska.nodes.nfv.NS, toska.nodes.nfv.VNF ]

```

7.10.17.4 Additional requirements

If a SAP defined in the NS topology template, represented by a virtual_link requirement in the NS node, exposes a VnfExtCp of a constituent VNF that, in turn, either:

- a) is exposing a VduCp whose VDU belongs to one or more deployable modules;
- b) is exposing a VipCp that is bound to a VduCp whose VDU belongs to one or more deployable modules; or
- c) is exposing a VirtualCp bound to a Vdu.OsContainerDeployableUnit that belongs to one or more deployable modules;

then, this (these) deployable module(s) shall be indicated in an instance of this policy targeting the VNF node that represents that constituent VNF. If a VDU belongs to more than one deployable module, each deployable module is indicated in an element of the list.

The sap_to_depl_mod_mapping property shall have one map entry for each SAP that fulfils the condition in the paragraph above. The map only contains entries related to the VNF node which the policy is targeting.

If the NS has multiple SAPs exposing VnfExtCps of different VNFs fulfilling this condition, instances of this policy shall be defined in each of these VNF nodes, each with the map entries relevant to the VNF node.

If a SAP defined in topology template of a composite NS, represented by a virtual_link requirement in the NS node, exposes a SAP of a constituent nested NS, and the SAP of the nested NS is indicated in the service template of the nested NS in an instance of this policy, an instance of the policy shall also be defined in the composite NS. The policy shall target the nested NS node template. In this case the value indicated in the string shall be the one indicated in the policy in the nested NS service template but preceded by the node template name targeted by the policy in the nested NS and with a colon ':' as separator between both.

7.10.17.5 Example

See clauses A.22.1 and A.22.2.

7.11 NSD TOSCA service template design

7.11.1 General

The TOSCA service template design for an NSD in the general case uses two levels of service templates as described in clause 7.11.2. In this design, the top level contains an abstract NS node template, i.e. without an implementation of the creation operation and is therefore substituted by one of the lower level service templates. This design is applicable regardless of whether the NS has one or multiple deployment flavours.

In the particular case of an NS with only one deployment flavour there is an alternative design which is described in clause 7.11.3 and which uses only one service template.

7.11.2 Single or multiple deployment flavour design with two levels of service templates

NSD shall be implemented as one top-level service template and one or multiple lower level service templates, where each lower level service template represents a deployment flavour. A separate YAML file with an NS specific node type definition which shall be derived from `tosca.nodes.nfv.NS` node type as defined in clause 7.8.1 shall be provided and is also considered as a part of an NSD. The top level service template shall be the main entry point of the NSD file structure as specified in ETSI GS NFV-SOL 007 [i.11], i.e. the Entry-definitions file. The file names of all the lower service templates shall be declared as the value of the Other-Definitions key in the TOSCA.meta file of the NSD file structure as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

See clause A.11 for an example of NSD design with multiple deployment flavours.

The top level service template shall comply with TOSCA-Simple-Profile-YAML-v1.3 [20] and shall include:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.3;
- b) an import statement referencing a yaml file which contains an NS specific node type definition;
- c) a topology template with a node template of the NS-specific node type, which:
 - a. shall include the `flavour_id` and other properties that are marked as required but do not have a default value in the VNF specific node type definition;
 - b. shall include the requirements as defined in clause 7.8.1;
 - c. may include other properties specified in the NS specific node type definition, excluding the `'service_availability_level'`, `'ns_profile'` and the `'priority'` properties; and
 - d. may include a substitute directive;
- d) optionally, additional NS-specific type definitions and import statements referencing additional NS-specific files containing only type definitions used by this TOSCA service template;
- e) optionally, metadata and dsl definitions as specified in section 3.10 of TOSCA-Simple-Profile-YAML-v1.3 [20].

Irrespective of the presence or absence of the substitute directive, the deployment and lifecycle management of instances of this NS node type is done by means of substitution by any of the lower level service templates as declared in the Other-Definitions of the TOSCA.meta file in the NSD file structure. The NSD consumer shall silently ignore the substitute directive if explicit directives are not supported.

The lower level service template is an implementable TOSCA service template for the deployment of a specific deployment flavour.

The lower level service template shall comply with TOSCA-Simple-Profile-YAML-v1.3 [20] and shall include:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.3;
- b) an import statement referencing a yaml file which contains an NS specific node type definition which shall be derived from `tosca.nodes.nfv.NS` node type as defined in clause 7.8.1;

- c) one of more import statements respectively referencing the yaml file which contains the included VNF specific node type definition if any or the included PNF specific node type definition if any or the included NS specific node type definition if any.

- If the imported files correspond to different versions of the same VNF, each shall be imported in its own namespace in the NSD to avoid conflicts between the type definitions defined in each of them, since versions of the same VNF are likely to have some common type definitions. This is done by defining a namespace prefix directly after the import statement:

imports:

- file: company.Vnf_A.yaml
- namespace_prefix: VNF-A

Note that in this case all references in the NSD to types defined in the company.Vnf_A.yaml file shall be then preceded by the namespace prefix and colon ':':

If the imported files contain a topology template, this topology template shall be ignored during the parsing of the NSD. To facilitate parsing the NSD, the imported files may be included in the NSD file structure as specified in ETSI GS NFV-SOL 007 [i.11], in which case the node type definitions they contain shall be identical to those contained in the corresponding VNF package, PNF archive and NSD file structure.

NOTE 1: These node type definitions are only used to parse the NSD. For the LCM operations of the corresponding VNF or nested NS, such node type definitions do not apply, only the files included in the related VNF package or NSD file structure will be used.

- d) optionally, additional NS-specific type definitions and import statements referencing additional NS-specific files containing type definitions used by this TOSCA service template; and

- e) a topology template describing the internal topology of the NS with:

- substitution_mappings indicating:
 - the same node type as defined in the NS specific node type definition service template;
 - a flavour_id property and its value as defined in substitution_filter which identifies the DF corresponding to this low level template within the NSD;

NOTE 2: Starting with version 3.3.1 of the present document, the property_mapping grammar was changed to support substitution_filter. For backward compatibility, TOSCA-Simple-Profile-YAML-v1.3 [20], clause 3.8.8.3. specifies the provisions for handling the previous grammar. Support of the Release 2 property_mapping grammar can be removed in subsequent versions of the present document.

- the mapping of the virtual_link requirements on SAPs;

NOTE 3: When a Sap re-exposes a VNF or PNF external connection point or a Sap of a nested NS, the virtual_link requirement of the NS is mapped to the virtual_link requirement of the VNF or PNF or nested NS node or a corresponding Forwarding node (i.e. the forwarding node that has a capability matching the virtual link requirement of the node whose external connection point or Sap is been exposed). Otherwise the virtual_link requirement of the NS is mapped to the external_virtual_link requirement of the SAP node.

- a node template referencing the NS specific node type containing the 'service_availability_level' property with a value, if applicable; the 'priority' property with a value, if applicable; implementations of the operations of the LCM interface to be executed by the NFVO, if applicable;
- additional node templates of type VNF, PNF, NS, NsVirtualLink, Sap, etc. that define the topology and composition of the NS flavour, the dependency requirements as defined in TOSCA-Simple-Profile-YAML-v1.3 [20] may be used between different VNF node templates, or between a VNF node template and a nested NS node template, or between different nested NS node templates to specify the order in which instances of the VNFs and/or nested NSs have to be created; and
- additional group definitions, policy definitions and parameter definitions if applicable.

- f) optionally, metadata and dsl definitions as specified in section 3.10 of TOSCA-Simple-Profile-YAML-v1.3 [20].

Either the `service_availability_level` property in the NS node template or the `service_availability_level` attribute in any of the VNF node templates may be provided, but not both. Either the `service_availability_level` property in the NS node template or the `service_availability_level` attribute in any of the `NsVirtualLink` node templates may be provided, but not both.

NOTE 4: The format and structure of an NSD file structure is defined in ETSI GS NFV-SOL 007 [i.11].

NOTE 5: All the imported type definition files as indicated either in the top level service template or in any of the lower level service template are considered as parts of an NSD.

When the `flavour_id` of an NS has been chosen (e.g. through an input parameter of an NS instantiation request received by a NFVO) among the values included in the NS node type imported into the top level service template, it is then used as the filter for selecting a particular lower level TOSCA service template inside the NSD file structure as described in TOSCA-Simple-Profile-YAML-v1.3 [20].

7.11.3 Single deployment flavour design with one service template

In case of the single deployment flavour scenario with one service template design, the NSD shall use TOSCA-Simple-Profile-YAML-v1.3 [20] and shall include:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.3;
- b) one of more import statements respectively referencing the yaml file which contains the included VNF specific node type definition if any or the included PNF specific node type definition if any or the included NS specific node type definition if any;

If the imported files correspond to different versions of the same VNF, each shall be imported in its own namespace in the NSD to avoid conflicts between the type definitions defined in each of them, since versions of the same VNF are likely to have some common type definitions. This is done by defining a namespace prefix directly after the import statement:

- imports:
 - file: `company.Vnf_A.yaml`
 - namespace_prefix: `VNF-A`

Note that in this case all references in the NSD to types defined in the `company.Vnf_A.yaml` file shall be then preceded by the namespace prefix and colon `':'`.

If the imported files contain a topology template, this topology template shall be ignored during the parsing of the NSD. To facilitate parsing the NSD, the imported files may be included in the NSD file structure as specified in ETSI GS NFV-SOL 007 [i.11], in which case the node type definitions they contain shall be identical to those contained in the corresponding VNF package, PNF archive and NSD file structure.

NOTE 1: These node type definitions are only used to parse the NSD. For the LCM operations of the corresponding VNF or nested NS, such node type definitions do not apply, only the files included in the related VNF package or NSD file structure will be used.

- c) optionally, additional VNF-specific type definitions and import statements referencing additional NS-specific files containing only type definitions used by this TOSCA service template;
- d) an NS specific node type definition derived from the `tosca.nodes.nfv.NS` node type, as defined in clause 7.8.1; and
- e) a topology template describing the internal topology of the NS with:
 - `substitution_mappings` indicating the same NS specific node type and the mapping of the `virtual_link` requirements on SAPs;

NOTE 2: When a Sap re-exposes a VNF or PNF external connection point or a Sap of a nested NS, the `virtual_link` requirement of the NS is mapped to the `virtual_link` requirement of the VNF or PNF or nested NS node or a corresponding Forwarding node (i.e. the forwarding node that has a capability matching the virtual link requirement of the node whose external connection point or Sap is been exposed). Otherwise the `virtual_link` requirement of the NS is mapped to the `external_virtual_link` requirement of the SAP node.

- a node template of this NS specific node type with the `flavour_id` and other properties and, if applicable, implementations of the operations of the LCM interface to be executed by the NFVO; and
 - additional node templates of type VNF, PNF, NS, `NsVirtualLink`, Sap, etc. that define the topology and composition of the NS flavour, the dependency requirements as defined in TOSCA-Simple-Profile-YAML-v1.3 [20] may be used between different VNF node templates, or between a VNF node template and a nested NS node template, or between different nested NS node templates to specify the order in which instances of the VNFs and/or nested NSs have to be created;
 - additional group definitions, policy definitions and parameter definitions if applicable;
- f) optionally, metadata and dsl definitions as specified in section 3.10 of TOSCA-Simple-Profile-YAML-v1.3 [20].

See clause A.8 for an example of NSD design with single deployment flavour.

NOTE 3: The service template is deployed stand-alone, i.e. without performing a substitution. However including the `substitution_mappings` rule indicates its ability to substitute a node template of the NS specific node type, which may appear in an NSD.

NOTE 4: All the imported type definition files as indicated in the service template are considered as parts of an NSD.

8 PNFD TOSCA model

8.1 Introduction

The PNFD information model specified by ETSI GS NFV-IFA 014 [2] is mapped to the TOSCA concepts. It represents as TOSCA topology template to be used by NFVO for preparing network connection.

Table 8.1-1 shows the TOSCA Type "derived from" values used when applying the TOSCA-Simple-Profile-YAML-v1.3 [20] to the PNFD.

Table 8.1-1: Mapping of ETSI GS NFV-IFA 014 [2] information elements with TOSCA types

ETSI NFV Information Element ETSI GS NFV-IFA 014 [2]	TOSCA type	Derived from
Pnf	<code>tosca.nodes.nfv.PNF</code>	<code>tosca.nodes.Root</code>
PnfExtCpd (PNF External Connection Point)	<code>tosca.nodes.nfv.PnfExtCp</code>	<code>tosca.nodes.Root</code>

Figure 8.1-1 provides an overview of the TOSCA node types used to build a service template for a PNFD.

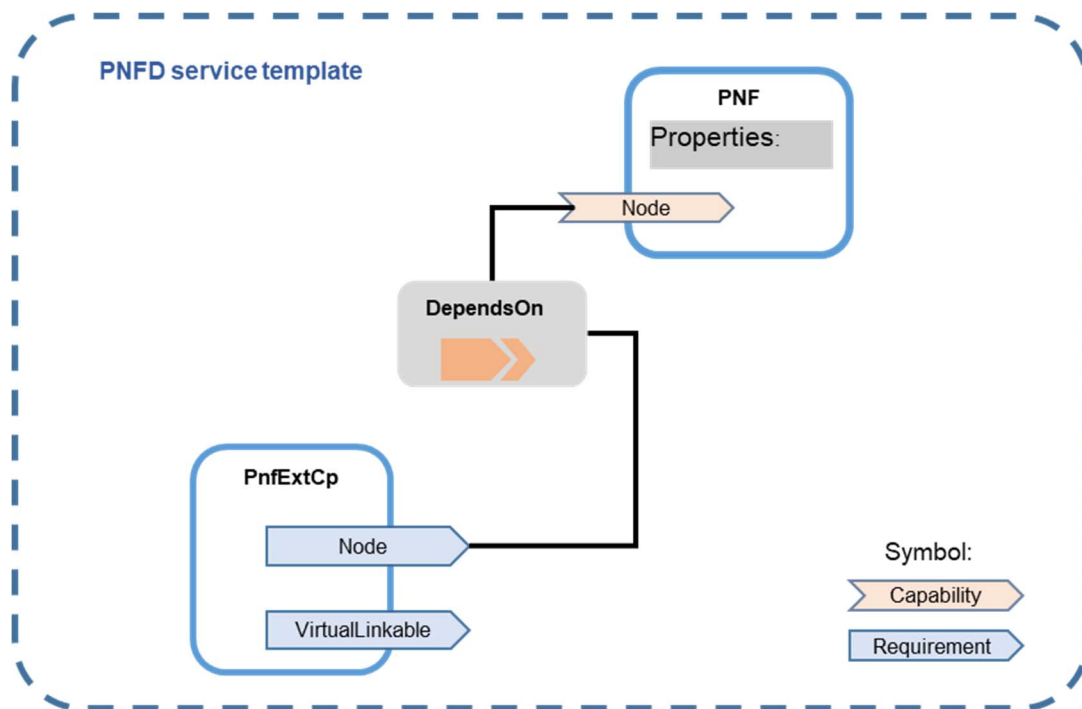


Figure 8.1-1: Service template PNFD overview

8.2 Data Types

8.2.1 `tosca.datatypes.nfv.CpProtocolData`

8.2.1.1 Description

The `CpProtocolData` data type is defined in clause 9.2.6 of the present document.

8.2.2 `tosca.datatypes.nfv.AddressData`

8.2.2.1 Description

The `AddressData` data type is defined in clause 9.2.3 of the present document.

8.2.3 `tosca.datatypes.nfv.L2AddressData`

8.2.3.1 Description

The `L2AddressData` data type is defined in clause 9.2.1 of the present document.

8.2.4 `tosca.datatypes.nfv.L3AddressData`

8.2.4.1 Description

The `L3AddressData` data type is defined in clause 9.2.2 of the present document.

8.2.5 toska.datatypes.nfv.LocationInfo

8.2.5.1 Description

The LocationInfo data type represents geographical information on the location where a PNF is deployed as specified in ETSI GS NFV-IFA 011 [1]. Table 8.2.5.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 8.2.5.1-1: Type name, shorthand, and URI

Shorthand Name	LocationInfo
Type Qualified Name	toscanfv:LocationInfo
Type URI	tosca.datatypes.nfv.LocationInfo

8.2.5.2 Properties

The properties of the LocationInfo data type shall comply with the provisions set out in table 8.2.5.2-1.

Table 8.2.5.2-1: Properties

Name	Required	Type	Constraints	Description
country_code	yes	string		Shall be a two-letter ISO 3166 [10] country code in capital letters.
civic_address_element	no	list of toska.datatypes.nfv.CivicAddressElement		Elements composing the civic address where the PNF is deployed.
geographic_coordinates	no	tosca.datatypes.nfv.GeographicCoordinates		Geographic coordinates (e.g. Altitude, Longitude, Latitude) where the PNF is deployed.

8.2.5.3 Definition

The syntax of the LocationInfo data type shall comply with the following definition:

```
tosca.datatypes.nfv.LocationInfo:
  derived_from: toska.datatypes.Root
  description: Represents geographical information on the location where a PNF is
  deployed.
  properties:
    country_code:
      type: string #two-letter ISO 3166 country code
      description: Country code
      required: true
    civic_address_element:
      type: list
      entry_schema:
        type: toska.datatypes.nfv.CivicAddressElement
        description: Elements composing the civic address where the PNF is deployed.
        required: false
    geographic_coordinates:
      type: toska.datatypes.nfv.GeographicCoordinates
      description: Geographic coordinates (e.g. Altitude, Longitude, Latitude) where
  the PNF is deployed.
      required: false
```

8.2.5.4 Examples

```
<some_tosca_entity>:
  properties:
    geographical_location_info:
      country_code: FR
      civic_address_element:
        - element_1
          ca_type: 3
          ca_value: Paris
```

8.2.5.5 Additional Requirements

None.

8.2.6 `tosca.datatypes.nfv.CivicAddressElement`

8.2.6.1 Description

The `CivicAddressElement` data type represents an element of a civic location as specified in IETF RFC 4776 [11]. Table 8.2.6.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 8.2.6.1-1: Type name, shorthand, and URI

Shorthand Name	<code>CivicAddressElement</code>
Type Qualified Name	<code>toscanfv: CivicAddressElement</code>
Type URI	<code>tosca.datatypes.nfv.CivicAddressElement</code>

8.2.6.2 Properties

The properties of the `CivicAddressElement` data type shall comply with the provisions set out in table 8.2.6.2-1.

Table 8.2.6.2-1

Name	Required	Type	Constraints	Description
<code>ca_type</code>	yes	string		Describe the content type of <code>caValue</code> . The value of <code>caType</code> shall comply with section 3.4 of IETF RFC 4776 [11].
<code>ca_value</code>	yes	string		Content of civic address element corresponding to the <code>caType</code> . The format <code>caValue</code> shall comply with section 3.4 of IETF RFC 4776 [11].

8.2.6.3 Definition

The syntax of the `CivicAddressElement` data type shall comply with the following definition:

```
tosca.datatypes.nfv.CivicAddressElement:
  derived_from: toska.datatypes.Root
  description: Represents an element of a civic location as specified in IETF
RFC 4776 [11].
  properties:
    ca_type:
      type: string # IETF RFC 4776
      description: caType as per IETF RFC 4776
      required: true
```

```

ca_value:
  type: string # IETF RFC 4776
  description: caValue as per IETF RFC 4776.
  required: true

```

8.2.6.4 Examples

See clause 8.2.5.4.

8.2.6.5 Additional Requirements

None.

8.2.7 `tosca.datatypes.nfv.GeographicCoordinates`

8.2.7.1 Description

The `GeographicCoordinates` data type represents a geographic coordinate location as specified in IETF RFC 6225 [21]. Table 8.2.7.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 8.2.7.1-1: Type name, shorthand, and URI

Shorthand Name	<code>GeographicCoordinates</code>
Type Qualified Name	<code>toscanfv: GeographicCoordinates</code>
Type URI	<code>tosca.datatypes.nfv.GeographicCoordinates</code>

8.2.7.2 Properties

The properties of the `GeographicCoordinates` data type shall comply with the provisions set out in table 8.2.7.2-1.

Table 8.2.7.2-1: Properties

Name	Required	Type	Constraints	Description
<code>latitude_uncertainty</code>	no	string		Describe the content of <code>latitude_uncertainty</code> . The value of <code>latitude_uncertainty</code> shall comply with <code>LatUnc</code> in section 2.3 of IETF RFC 6225 [21].
<code>latitude</code>	yes	string		Describe the content of <code>latitude</code> . The value of <code>latitude</code> shall comply with <code>Latitude</code> in section 2.3 of IETF RFC 6225 [21].
<code>longitude_uncertainty</code>	no	string		Describe the content of <code>longitude_uncertainty</code> . The value of <code>longitude_uncertainty</code> shall comply with <code>LongUnc</code> in section 2.3 of IETF RFC 6225 [21].
<code>longitude</code>	yes	string		Describe the content type of <code>longitude</code> . The value of <code>longitude</code> shall comply with <code>Longitude</code> in section 2.3 of IETF RFC 6225 [21].
<code>altitude_type</code>	yes	string		Describe the content type of <code>altitude_type</code> . The value of <code>altitude_type</code> shall comply with <code>AType</code> in section 2.4 of IETF RFC 6225 [21].
<code>altitude_uncertainty</code>	no	string		Describe the content of <code>altitude_uncertainty</code> . The value of <code>altitude_uncertainty</code> shall comply with <code>AltUnc</code> in section 2.4 of IETF RFC 6225 [21].
<code>altitude</code>	yes	string		Describe the content of <code>altitude</code> . The value of <code>altitude</code> shall comply with <code>Altitude</code> in section 2.4 of IETF RFC 6225 [21].

8.2.7.3 Definition

The syntax of the GeographicCoordinates data type shall comply with the following definition:

```
tosca.datatypes.nfv.GeographicCoordinates:
  derived_from: toska.datatypes.Root
  description: Represents an element of a geographic coordinate location as
  specified in IETF RFC 6225 [21].
  properties:
    latitude_uncertainty:
      type: string # IETF RFC 6225
      description: LatUnc as per IETF RFC 6225
      required: false
    latitude:
      type: string # IETF RFC 6225
      description: Latitude value as per IETF RFC 6225
      required: true
    longitude_uncertainty:
      type: string # IETF RFC 6225
      description: LongUnc as per IETF RFC 6225
      required: false
    longitude:
      type: string # IETF RFC 6225
      description: Longitude value as per IETF RFC 6225
      required: true
    altitude_type:
      type: string # IETF RFC 6225
      description: AType value as per IETF RFC 6225
      required: true
    altitude_uncertainty:
      type: string # IETF RFC 6225
      description: AltUnc as per IETF RFC 6225
      required: false
    altitude:
      type: string # IETF RFC 6225
      description: Altitude value as per IETF RFC 6225
      required: true
```

8.2.7.4 Examples

None.

8.2.7.5 Additional Requirements

None.

8.3 Artifact Types

None.

8.4 Capability Types

8.4.1 toska.capabilities.nfv.VirtualLinkable

8.4.1.1 Description

The VirtualLinkable capability type is defined in clause 9.4.1 of the present document.

8.5 Requirement Types

None.

8.6 Relationship Types

8.6.1 `tosca.relationships.nfv.VirtualLinksTo`

8.6.1.1 Description

The `VirtualLinksTo` relationship type is defined in clause 9.6.1 of the present document representing an association relationship between a PNF external connection point and an `NsVirtualLink` node type.

8.7 Interface Types

None.

8.8 Node Types

8.8.1 `tosca.nodes.nfv.PNF`

8.8.1.1 Description

The Physical Network Function (PNF) node type describes a PNF in terms of deployment behaviour requirements, which it contains PNFD identifier, version and functional description and so on as defined by ETSI GS NFV-IFA 014 [2]. Table 8.8.1.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 8.8.1.1-1: Type name, shorthand, and URI

Shorthand Name	PNF
Type Qualified Name	<code>tosca:PNF</code>
Type URI	<code>tosca.nodes.nfv.PNF</code>

8.8.1.2 Properties

The properties of the PNF node type shall comply with the provisions set out in table 8.8.1.2-1.

Table 8.8.1.2-1: Properties

Name	Required	Type	Constraints	Description
<code>descriptor_id</code>	yes	string		Identifier of this PNFD information element. It uniquely identifies the PNFD. See note 1.
<code>function_description</code>	yes	string		Describes the PNF function.
<code>provider</code>	yes	string		Identifies the provider of the PNFD. See note 2.
<code>version</code>	yes	string		Identifies the version of the PNFD.
<code>descriptor_invariant_id</code>	yes	string		Identifier of this PNFD in a version independent manner. This attribute is invariant across versions of PNFD with no constraint on the changes across versions. See note 1.

Name	Required	Type	Constraints	Description
ext_invariant_id	no	string		Identifies a PNFD in a version independent manner. This attribute is invariant across versions of the PNFD that expose the same external connectivity, i.e. same number and name of the requirements for VirtualLinkable capability that represent PnfExtCps. See note 3. When used in a PNF node template in an NSD it allows for PNF instances during NS LCM the use of a PNFD different from the one referenced by the descriptor_id attribute, provided they have the same ext_invariant_id. See note 4.
name	yes	string		Name to identify the PNFD.
geographical_location_info	no	tosca.datatype.nfv.LocationInfo		Provides information about the geographical location (e.g. geographic coordinates or address of the building, etc.) of the PNF.
version_dependency	no	list of tosc.datatypes.nfv.VersionDependency		Identifies versions of descriptors of other constituents in the NSD upon which the PNF depends. The dependencies may be described for the PNFD referenced with descriptor_id and for PNFs with the same ext_invariant_id. There shall not be more than one version_dependency present with the same dependent_constituent_id. See notes 5 and 6.
NOTE 1: The value of the descriptor_id string and the value of the descriptor_invariant_id string shall comply with an UUID format as specified in section 3 of [9]. NOTE 2: The provider of the PNFD might be different from the provider of the PNF. NOTE 3: This condition implies that PNFs with the same ext_invariant_id preserve external invariancy. Therefore, fulfilling this condition allows to use a different version of a PNFD in an NS instance without modification of the NSD on which the NS instance is based. The use of a different version of a PNFD is ultimately under the control of the service provider of the NS and it should consider if the NSD fulfils the requirements of the PnfExtCps (e.g. bitrate, IP version, etc.). NOTE 4: When the PNF node is used in an NSD, this property may only be included if the PNF node type definition in the PNFD includes the property with a value. If the property is supported in the PNF node in the PNFD, it should be included in the PNF node in the NSD to avoid changes in the NSD caused by version changes in the PNFD. NOTE 5: This property may only be used in a PNF node template inside an NSD service template. NOTE 6: This property does not indicate any ordering requirement, i.e. the dependent PNF and the other constituents upon which this PNF depends may be created or incorporated to the NS in any order, unless specified otherwise by the 'dependency' requirement in the node templates in the NSD.				

8.8.1.3 Attributes

None.

8.8.1.4 Requirements

The requirements of the VNF node type shall comply with the provisions set out in table 8.8.1.4-1.

Table 8.8.1.4-1: Requirements

Name	Required	Capability type	Constraints	Description
virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Describes the requirements for linking to virtual link

8.8.1.5 Capabilities

None.

8.8.1.6 Definition

The syntax of the PNF node type shall comply with the following definition:

```
tosca.nodes.nfv.PNF:
  derived_from: tosca.nodes.Root
  properties:
    descriptor_id: # instead of pnfd_id
      type: string # UUID
      required: true
      description: Identifier of this PNFD information element. It uniquely
identifies the PNFD.
    function_description:
      type: string
      required: true
      description: Describes the PNF function.
    provider:
      type: string
      required: true
      description: Identifies the provider of the PNFD.
    version:
      type: string
      required: true
      description: Identifies the version of the PNFD.
    descriptor_invariant_id: # instead of pnfd-invariant-id
      type: string # UUID
      required: true
      description: Identifier of this PNFD in a version independent manner. This
attribute is invariant across versions of PNFD with no constraint on the changes
across versions.
    ext_invariant_id:
      type: string # UUID
      required: false
      description: Identifies a PNFD in a version independent manner. This
attribute is invariant across versions of the PNFD that expose the same external
connectivity, i.e. same number and name of the requirements for VirtualLinkable
capability that represent PnfExtCps. When used in a PNF node template in an NSD it
allows for PNF instances during NS LCM the use of a PNFD different from the one
referenced by the descriptor_id attribute, provided they have the same
ext_invariant_id.
    name:
      type: string
      required: true
      description: Name to identify the PNFD.
    geographical_location_info:
      type: tosca.datatypes.nfv.LocationInfo
      required: false
      description: Provides information about the geographical location (e.g.
geographic coordinates or address of the building, etc.) of the PNF.
    version_dependency:
      type: list
      description: Identifies versions of descriptors of other constituents in
the NSD upon which the PNF depends. The dependencies may be described for the PNFD
referenced with descriptor_id and for PNFDs with the same ext_invariant_id.
      required: false
    entry_schema:
      type: tosca.datatypes.nfv.VersionDependency
```

```

requirements:
  - virtual_link:
      capability: tosca.capabilities.nfv.VirtualLinkable
      relationship: tosca.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 1 ]
  # Additional requirements shall be defined in the PNF specific node type
  (deriving from tosca.nodes.nfv.PNF) corresponding to NS virtual links that need to
  connect to PnfExtCps

```

8.8.1.7 Artifact

None.

8.8.1.8 Additional Requirements

For a given PNFD, a new PNF node type shall be defined following the below requirements:

- a) The node type shall be derived from: `tosca.nodes.nfv.PNF`.
- b) All properties listed in `tosca.nodes.nfv.PNF` where the "required:" field is set to "true" shall be included with their values indicates as constraints or assigned as final fixed values if only one value is permitted.
- c) The requirements of `tosca.nodes.nfv.PNF` shall be preserved.
- d) Depending on the number of external connection points of the PNF that need to connect to NS virtual links, additional requirements for VirtualLinkable capability shall be defined. In this case, it is the PNFD author's choice to use the requirement for VirtualLinkable capability defined in the `tosca.nodes.nfv.PNF` node type or use only the additional requirements defined in the derived PNF specific node type. In the latter case, the `virtual_link` requirement should be included in the node type definition with occurrences [0, 0].
- e) If the PNFD supports external invariancy the PNF node type definition shall include the `ext_invariant_id` property with its value indicated as constraint.

8.8.1.9 Example

See clause A.10.

8.8.2 `tosca.nodes.nfv.PnfExtCp`

8.8.2.1 Description

The `PnfExtCp` node type describes the characteristics of an external interface, also known as an external CP, where to connect the PNF to a VL, as defined by ETSI GS NFV-IFA 014 [2]. Table 8.8.2.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 8.8.2.1-1: Type name, shorthand, and URI

Shorthand Name	<code>PnfExtCp</code>
Type Qualified Name	<code>toscanfv:PnfExtCp</code>
Type URI	<code>tosca.nodes.nfv.PnfExtCp</code>

8.8.2.2 Properties

The properties applied to `PnfExtCp` node are derived from `Cp` node type.

8.8.2.3 Attributes

None.

8.8.2.4 Requirements

The requirements of the PnfExtCp node type shall comply with the provisions set out in table 8.8.2.4-1.

Table 8.8.2.4-1: Requirements

Name	Required	Capability type	Constraints	Description
external_virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Specifies that CP instances require to be connected to a node that has a VirtualLinkable capability.

8.8.2.5 Capabilities

None.

8.8.2.6 Definition

The syntax of the PnfExtCp node type shall comply with the following definition:

```
tosca.nodes.nfv.PnfExtCp:
  derived_from: toska.nodes.nfv.Cp
  description: node definition of PnfExtCp.
  requirements:
    - external_virtual_link:
        capability: toska.capabilities.nfv.VirtualLinkable
        relationship: toska.relationships.nfv.VirtualLinksTo
        occurrences: [0, 1]
```

8.8.3 toska.nodes.nfv.Cp

8.8.3.1 Description

The Cp node type is defined in clause 9.8.1 of the present document.

8.9 Group Types

None.

8.10 Policy Types

8.10.1 toska.policies.nfv.PnfSecurityGroupRule

8.10.1.1 Description

The PnfSecurityGroupRule policy type when used in a PNFD specifies the matching criteria for the ingress and/or egress traffic to and from visited PNF external connection points. Table 8.10.1.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 8.10.1.1-1: Type name, shorthand, and URI

Shorthand Name	PnfSecurityGroupRule
Type Qualified Name	toscanfv:PnfSecurityGroupRule
Type URI	tosca.policies.nfv.PnfSecurityGroupRule

8.10.1.2 Properties

None.

8.10.1.3 targets

The targets of the SecurityGroupRule policy types shall comply with the provisions set out in table 8.10.1.3-1.

Table 8.10.1.3-1: Targets

Name	Required	Type	Constraints	Description
targets	yes	string[]	Valid types: tosca.nodes.nfv.P nfExtCp	Target connection points of PnfExtCp

8.10.1.4 Definition

The syntax of the NsSecurityGroupRule policy type shall comply with the following definition:

```
tosca.policies.nfv.PnfSecurityGroupRule:
  derived_from: toska.policies.nfv.Abstract.SecurityGroupRule
  description: The PnfSecurityGroupRule type is a policy type specified the
  matching criteria for the ingress and/or egress traffic to/from visited PNF
  external connection points.
  targets: [ toska.nodes.nfv.PnfExtCp ]
```

8.10.1.5 Additional Requirements

None.

8.10.2 toska.policies.nfv.Abstract.SecurityGroupRule

8.10.2.1 Description

The Abstract.SecurityGroupRule policy type is defined in clause 9.10.1 of the present document.

8.11 PNFD TOSCA service template design

8.11.1 General

One single TOSCA service template is used to design a PNFD which shall comply with TOSCA-Simple-Profile-YAML-v1.3 [20] and includes:

- an import statement referencing the TOSCA types definition file as defined in clause B.4;
- a PNF specific node type definition derived from the toska.nodes.nfv.PNF node type, as defined in clause 8.8.1; and
- a topology template describing the internal topology of the PNF with:
 - substitution_mappings indicating the same PNF specific node type and the mapping of the virtual_link requirements on PNF external connection point;
 - a node template of this PNF specific node type with the properties as defined in toska.nodes.nfv.PNF; and
 - additional node templates of type PnfExtCp that define the connection information of the PNF;

- d) optionally, metadata and dsl definitions as specified in section 3.10 of TOSCA-Simple-Profile-YAML-v1.3 [20].

See clause A.10 for an example of PNFD design.

NOTE: The service template is deployed stand-alone, i.e. without performing a substitution. However including the substitution_mappings rule indicates its ability to substitute a node template of the PNF specific node type, which may appear in an NSD.

9 Common Definitions

9.1 Introduction

This clause defines the TOSCA type definitions which are used by at least two types of deployment templates among those identified in clause 5.1.

9.2 Data Types

9.2.1 `tosca.datatypes.nfv.L2AddressData`

9.2.1.1 Description

The L2AddressData data type describes the information on the MAC addresses to be assigned to a connection point as defined in ETSI GS NFV-IFA 011 [1]. Table 9.2.1.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.1.1-1: Type name, shorthand, and URI

Shorthand Name	L2AddressData
Type Qualified Name	toscanfv:L2AddressData
Type URI	tosca.datatypes.nfv.L2AddressData

9.2.1.2 Properties

The properties of the L2AddressData data type shall comply with the provisions set out in table 9.2.1.2-1.

Table 9.2.1.2-1: Properties

Name	Required	Type	Constraints	Description
mac_address_assignment	yes	boolean		Specifies which mode is used for the MAC address assignment. If it is set to True, a MAC address is expected to be provided by a management entity via the NFV MANO interfaces towards the VNFM using attributes standardized for this purpose in the NFV-MANO information model and is further transferred from the VNFM to the VIM/CISM. A MAC address will be automatically assigned by the VIM/NFVI/CISM as fallback if not provided. If it is set to False, a MAC address is expected to be assigned by means specific to the VNF itself (e.g. by an LCM script, by the EM) and is further transferred from the VNFM to the VIM/CISM. A MAC address will be automatically assigned by the VIM/NFVI/CISM as fallback if not provided to the VIM/CISM.

9.2.1.3 Definition

The syntax of the L2AddressData data type shall comply with the following definition:

```

tosca.datatypes.nfv.L2AddressData:
  derived_from: toska.datatypes.Root
  description: Describes the information on the MAC addresses to be assigned to a
connection point.
  properties:
    mac_address_assignment:
      type: boolean
      description: Specifies which mode is used for the MAC address assignment. If
it is set to True, a MAC address is expected to be provided by a management entity via
the NFV MANO interfaces towards the VNFM using attributes standardized for this
purpose in the NFV-MANO information model and is further transferred from the VNFM to
the VIM/CISM. A MAC address will be automatically assigned by the VIM/NFVI/CISM as
fallback if not provided. If it is set to False, a MAC address is expected to be
assigned by means specific to the VNF itself (e.g. by an LCM script, by the EM) and is
further transferred from the VNFM to the VIM/CISM. A MAC address will be automatically
assigned by the VIM/NFVI/CISM as fallback if not provided to the VIM/CISM.
      required: true

```

9.2.1.4 Examples

```

<some_tosca_entity>:
  properties:
    l2_address_data:
      mac_address_assignment: true

```

9.2.1.5 Additional Requirements

None.

9.2.2 toska.datatypes.nfv.L3AddressData

9.2.2.1 Description

The L3AddressData data type supports providing information about Layer 3 level addressing scheme and parameters applicable to a CP, as defined in ETSI GS NFV-IFA 011 [1]. Table 9.2.2.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.2.1-1: Type name, shorthand, and URI

Shorthand Name	L3AddressData
Type Qualified Name	toscanfv:L3AddressData
Type URI	tosca.datatypes.nfv.L3AddressData

9.2.2.2 Properties

The properties of the L3AddressData data type shall comply with the provisions set out in table 9.2.2.2-1.

Table 9.2.2.2-1: Properties

Name	Required	Type	Constraints	Description
ip_address_assignment	no	boolean		Specify which mode is used for the IP address assignment. If it is set to True and this property is not used in the context of the VirtualCp node type, IP configuration information shall be provided for the VNF by a management entity using the NFV MANO interfaces towards the VNFM. If it is set to True and this property is used in the context of the VirtualCp node type, IP configuration information should be provided for the VNF by a management entity using the NFV MANO interfaces towards the VNFM. If it is not provided, the CISM assigns an IP address. See note 2. If it is set to False, the value of the ipAddressAssignmentSubtype property defines the method of IP address assignment. Shall be present if the fixed_ip_address property is not present and should be absent otherwise. See note 1.
ip_address_assignment_subtype	no	string	Valid values: See YAML definition constraints	Method of IP address assignment in case the IP configuration is not provided using the NFV MANO interfaces towards the VNFM. Shall be present in case the ip_address_assignment property is set to False and shall be absent otherwise. Description of the valid values: - dynamic: the VNF gets an IP address that is dynamically assigned by the NFVI/VIM/CISM without receiving IP configuration information from the MANO interfaces. - vnf_pkg: an IP address defined by the VNF provider is assigned by means included as part of the VNF package (e.g. LCM script). - external: an IP address is provided by an external management entity (such as EM) directly towards the VNF.
floating_ip_activated	yes	boolean		Specify if the floating IP scheme is activated on the Connection Point or not. See notes 1 and 3.
ip_address_type	no	string	Valid values: See YAML definition constraints	Define address type. The address type should be aligned with the address type supported by the layer_protocols properties of the connection point. See note 1.
number_of_ip_address	no	Integer	greater_than: 0	Minimum number of IP addresses to be assigned. See note 1.

Name	Required	Type	Constraints	Description
fixed_ip_address	no	list of string		Fixed IP addresses to be assigned to the internal CP instance. This property enables the VNF provider to define fixed IP addresses for internal CP instances to be assigned by the VNFM or the NFVO. This property is only permitted for Cpd without external connectivity, i.e. connectivity outside the VNF. If present, it shall be compatible with the values of the L3ProtocolData of the VnfVirtualLink referred to by the Cp, if L3ProtocolData is included in the VnfVirtualLink. See note 1.
NOTE 1: If the fixed_ip_address property is present: - the ip_address_assignment property should not be present. If it is present in this context, its value has no meaning and shall be ignored when processing the VNFD. Using the ip_address_assignment property in this context is deprecated; implementations need to be aware that support can be removed in subsequent versions of the present document; - the value of the floating_ip_activated property shall be set to false; - the value of the ip_address_type property, if present, shall be set consistently with the fixed_ip_address; - the value of the number_of_ip_address property, if present, shall be set consistently with the cardinality of the fixed_ip_address. NOTE 2: For VirtualCps exposed by a VNF component realized by one or more OS containers, if the used container technology does not support the capability to set a defined IP address and the "ip_address_assignment" property is set to True, IP configuration information should not be provided for the VNF by a management entity using the NFV MANO interfaces towards the VNFM. If it is provided nevertheless, the CISM might not be able to assign that IP address to the VirtualCp instance. NOTE 3: For CPs of a VNFC realized by one or a set of OS containers the ability to configure a floating IP address might not be supported by all container technologies.				

9.2.2.3 Definition

The syntax of the L3AddressData data type shall comply with the following definition:

```

tosca.datatypes.nfv.L3AddressData:
  derived_from: toska.datatypes.Root
  description: Provides information about Layer 3 level addressing scheme and
parameters applicable to a CP
  properties:
    ip_address_assignment:
      type: boolean
      description: Specify which mode is used for the IP address assignment. If it
is set to True and this property is not used in the context of the VirtualCp node
type, IP configuration information shall be provided for the VNF by a management
entity using the NFV MANO interfaces towards the VNFM. If it is set to True and this
property is used in the context of the VirtualCp node type, IP configuration
information should be provided for the VNF by a management entity using the NFV MANO
interfaces towards the VNFM. If it is not provided, the CISM assigns an IP address. If
it is set to False, the value of the ip_address_assignment_subtype property defines
the method of IP address assignment. Shall be present if the fixed_ip_address property
is not present and should be absent otherwise.
      required: false
    ip_address_assignment_subtype:
      type: string
      description: "Method of IP address assignment in case the IP configuration is
not provided using the NFV MANO interfaces towards the VNFM. Description of the valid
values: (1) dynamic: the VNF gets an IP address that is dynamically assigned by the
NFVI/VIM/CISM without receiving IP configuration information from the MANO interfaces,
(2) vnf_pkg: an IP address defined by the VNF provider is assigned by means included

```

as part of the VNF package (e.g. LCM script); (3) external: an IP address is provided by an external management entity (such as EM) directly towards the VNF. Shall be present in case the `ip_address_assignment` property is set to `False` and shall be absent otherwise."

```

    required: false
    constraints:
      - valid_values: [ dynamic, vnf_pkg, external ]
floating_ip_activated:
  type: boolean
  description: Specifies if the floating IP scheme is activated on the
Connection Point or not
  required: true
ip_address_type:
  type: string
  description: Defines address type. The address type should be aligned with the
address type supported by the layer_protocols properties of the connection point
  required: false
  constraints:
    - valid_values: [ ipv4, ipv6 ]
number_of_ip_address:
  type: integer
  description: Minimum number of IP addresses to be assigned
  required: false
  constraints:
    - greater_than: 0
fixed_ip_address:
  type: list
  description: Fixed IP addresses to be assigned to the internal CP instance.
This property enables the VNF provider to define fixed IP addresses for internal CP
instances to be assigned by the VNFM or the NFVO. This property is only permitted for
Cpds without external connectivity, i.e. connectivity outside the VNF. If present, it shall be compatible
with the values of the L3ProtocolData of the VnfVirtualLink referred to by the Cp, if L3ProtocolData is included in
the VnfVirtualLink
  required: false
  entry_schema:
    type: string

```

9.2.2.4 Examples

The following is an example of the case using dynamic IP address.

```

<some_tosca_entity>:
properties:
  l3_address_data:
    ip_address_assignment: true
    floating_ip_activated: true
    ip_address_type: ipv4
    number_of_ip_address: 4

```

The following is an example of the case using fixed IP address.

```

<some_tosca_entity>:
properties:
  l3_address_data:
    floating_ip_activated: false
    ip_address_type: ipv4
    number_of_ip_address: 1
    fixed_ip_address:
      - 192.168.0.1

```

9.2.2.5 Additional Requirements

None.

9.2.3 `tosca.datatypes.nfv.AddressData`

9.2.3.1 Description

The AddressData data type describes information about the addressing scheme and parameters applicable to a CP, as defined in ETSI GS NFV-IFA 011 [1]. Table 9.2.3.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.3.1-1: Type name, shorthand, and URI

Shorthand Name	AddressData
Type Qualified Name	tosca.nfv.AddressData
Type URI	tosca.datatypes.nfv.AddressData

9.2.3.2 Properties

The properties of the AddressData data type shall comply with the provisions set out in table 9.2.3.2-1.

Table 9.2.3.2-1: Properties

Name	Required	Type	Constraints	Description
address_type	yes	string	Valid values: See YAML definition constraints	Describes the type of the address to be assigned to a connection point The content type shall be aligned with the address type supported by the layerProtocol property of the connection point.
l2_address_data	no	tosca.datatypes.nfv.L2AddressData	Shall be present when the address_type is mac_address.	Provides the information on the MAC addresses to be assigned to a connection point.
l3_address_data	no	tosca.datatypes.nfv.L3AddressData	Shall be present when the address_type is ip_address.	Provides the information on the IP addresses to be assigned to a connection point.

9.2.3.3 Definition

The syntax of the AddressData data type shall comply with the following definition:

```

tosca.datatypes.nfv.AddressData:
  derived_from: tosca.datatypes.Root
  description: Describes information about the addressing scheme and parameters
  applicable to a CP
  properties:
    address_type:
      type: string
      description: Describes the type of the address to be assigned to a connection
      point. The content type shall be aligned with the address type supported by the
      layerProtocol property of the connection point
      required: true
      constraints:
        - valid_values: [ mac_address, ip_address ]
    l2_address_data:
      type: tosca.datatypes.nfv.L2AddressData
      description: Provides the information on the MAC addresses to be assigned to a
      connection point.
      required: false
    l3_address_data:
      type: tosca.datatypes.nfv.L3AddressData
      description: Provides the information on the IP addresses to be assigned to a
      connection point
      required: false

```

9.2.3.4 Examples

The following is an example of the case using dynamic IP address.

```

<some_tosca_entity>:
  properties:
    address_data:
      address_type: ip_address
    l3_address_data:
      ip_address_assignment: true
      floating_ip_activated: true
      ip_address_type: ipv4
      number_of_ip_address: 4

```

The following is an example of the case using fixed IP address.

```

<some_tosca_entity>:
  properties:
    l3_address_data:
      ip_address_assignment: true
      floating_ip_activated: false
      ip_address_type: ipv4
      number_of_ip_address: 1
      fixed_ip_address:
        - 192.168.0.1

```

9.2.3.5 Additional Requirements

None.

9.2.4 toska.datatypes.nfv.ConnectivityType

9.2.4.1 Description

The ConnectivityType data type describes the protocol exposed by a virtual link and the flow pattern supported by the virtual link, as defined in ETSI GS NFV-IFA 011 [1]. Table 9.2.4.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.4.1-1: Type name, shorthand, and URI

Shorthand Name	ConnectivityType
Type Qualified Name	toscanfv:ConnectivityType
Type URI	tosca.datatypes.nfv.ConnectivityType

9.2.4.2 Properties

The properties of the ConnectivityType shall comply with the provisions set out in table 9.2.4.2-1.

Table 9.2.4.2-1: Properties

Name	Required	Type	Constraints	Description
layer_protocols	yes	list of string	Valid values: See YAML definition constraints	Identifies the protocol a virtualLink gives access to (ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire). The top layer protocol of the virtualLink protocol stack shall always be provided. The lower layer protocols may be included when there are specific requirements on these layers. See note.
flow_pattern	no	string	Valid values: See YAML definition constraints	Identifies the flow pattern of the connectivity.
NOTE: If more than 1 values are present, the first value represents the highest layer protocol data, and the last value represents the lowest layer protocol data.				

9.2.4.3 Definition

The syntax of the ConnectivityType data type shall comply with the following definition:

```
tosca.datatypes.nfv.ConnectivityType:
  derived_from: toska.datatypes.Root
  description: describes additional connectivity information of a virtualLink
  properties:
    layer_protocols:
      type: list
      description: Identifies the protocol a virtualLink gives access to (ethernet,
mpls, odu2, ipv4, ipv6, pseudo-wire). The top layer protocol of the virtualLink
protocol stack shall always be provided. The lower layer protocols may be included
when there are specific requirements on these layers.
      required: true
      entry_schema:
        type: string
        constraints:
          - valid_values: [ ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire ]
    flow_pattern:
      type: string
      description: Identifies the flow pattern of the connectivity
      required: false
      constraints:
        - valid_values: [ line, tree, mesh ]
```

9.2.4.4 Examples

```
<some_tosca_entity>:
  properties:
    connectivity_type:
      layer_protocol:
        - ipv4
      flow_pattern: mesh
```

9.2.4.5 Additional Requirements

None.

9.2.5 tosca.datatypes.nfv.LinkBitrateRequirements

9.2.5.1 Description

The LinkBitrateRequirements data type describes the requirements in terms of bitrate for a virtual link. Table 9.2.5.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.5.1-1: Type name, shorthand, and URI

Shorthand Name	LinkBitrateRequirements
Type Qualified Name	toscanfv:LinkBitrateRequirements
Type URI	tosca.datatypes.nfv.LinkBitrateRequirements

9.2.5.2 Properties

The properties of the LinkBitrateRequirements data type shall comply with the provisions set out in table 9.2.5.2-1.

Table 9.2.5.2-1: Properties

Name	Required	Type	Constraints	Description
root	yes	integer	greater_or_equal: 0	Specifies the throughput requirement in bits per second of the link (e.g. bitrate of E-Line, root bitrate of E-Tree, aggregate capacity of E-LAN).
leaf	no	integer	greater_or_equal: 0	Specifies the throughput requirement in bits per second of leaf connections to the link when applicable to the connectivity type (e.g. for E-Tree and E-LAN branches).

9.2.5.3 Definition

The syntax of the LinkBitrateRequirements data type shall comply with the following definition:

```

tosca.datatypes.nfv.LinkBitrateRequirements:
  derived_from: tosca.datatypes.Root
  description: describes the requirements in terms of bitrate for a virtual link
  properties:
    root:
      type: integer # in bits per second
      description: Specifies the throughput requirement in bits per second of the
link (e.g. bitrate of E-Line, root bitrate of E-Tree, aggregate capacity of E-LAN).
      required: true
      constraints:
        - greater_or_equal: 0
    leaf:
      type: integer # in bits per second
      description: Specifies the throughput requirement in bits per second of leaf
connections to the link when applicable to the connectivity type (e.g. for E-Tree and
E LAN branches).
      required: false
      constraints:
        - greater_or_equal: 0

```

9.2.5.4 Examples

None.

9.2.5.5 Additional Requirements

None.

9.2.6 tosca.datatypes.nfv.CpProtocolData

9.2.6.1 Description

The CpProtocolData data type describes and associates the protocol layer that a CP uses together with other protocol and connection point information. Table 9.2.6.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.6.1-1: Type name, shorthand, and URI

Shorthand Name	CpProtocolData
Type Qualified Name	toscanfv:CpProtocolData
Type URI	tosca.datatypes.nfv.CpProtocolData

9.2.6.2 Properties

The properties of the CpProtocolData data type shall comply with the provisions set out in table 9.2.6.2-1.

Table 9.2.6.2-1: Properties

Name	Required	Type	Constraints	Description
associated_layer_protocol	yes	string	Valid values: See YAML definition constraints	One of the values of the property layer_protocols of the CP.
address_data	no	list of toska.datatypes.nfv.AddressData		Provides information on the addresses to be assigned to the CP.

9.2.6.3 Definition

The syntax of the CpProtocolData data type shall comply with the following definition:

```

tosca.datatypes.nfv.CpProtocolData:
  derived_from: toska.datatypes.Root
  description: Describes and associates the protocol layer that a CP uses together
with other protocol and connection point information
  properties:
    associated_layer_protocol:
      type: string
      description: One of the values of the property layer_protocols of the CP
      required: true
      constraints:
        - valid_values: [ ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire ]
    address_data:
      type: list
      description: Provides information on the addresses to be assigned to the CP
      entry_schema:
        type: toska.datatypes.nfv.AddressData
        required: false

```

9.2.6.4 Examples

None.

9.2.6.5 Additional Requirements

None.

9.2.7 toska.datatypes.nfv.Qos

9.2.7.1 Description

The QoS describes QoS data type a given VL used in a VNF deployment flavour. Table 9.2.7.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.7.1-1: Type name, shorthand, and URI

Shorthand Name	Qos
Type Qualified Name	toscanfv:Qos
Type URI	tosca.datatypes.nfv.Qos

9.2.7.2 Properties

The properties of the Qos data type shall comply with the provisions set out in table 9.2.7.2-1.

Table 9.2.7.2-1: Properties

Name	Required	Type	Constraints	Description
latency	yes	scalar-unit.time	greater_than: 0 s	Specifies the maximum latency.
packet_delay_variation	yes	scalar-unit.time		Specifies the maximum jitter.
packet_loss_ratio	no	float	in_range: [0,1]	Specifies the maximum packet loss ratio.

9.2.7.3 Definition

The syntax of the Qos data type shall comply with the following definition:

```

tosca.datatypes.nfv.Qos:
  derived_from: tosca.datatypes.Root
  description: describes QoS data for a given VL used in a VNF deployment flavour
  properties:
    latency:
      type: scalar-unit.time #Number
      description: Specifies the maximum latency
      required: true
      constraints:
        - greater_than: 0 s
    packet_delay_variation:
      type: scalar-unit.time #Number
      description: Specifies the maximum jitter
      required: true
      constraints:
        - greater_or_equal: 0 s
    packet_loss_ratio:
      type: float
      description: Specifies the maximum packet loss ratio
      required: false
      constraints:
        - in_range: [ 0.0, 1.0 ]

```

9.2.7.4 Examples

None.

9.2.7.5 Additional Requirements

None.

9.2.8 tosca.datatypes.nfv.VnfProfile

9.2.8.1 Description

The VnfProfile data type describes a profile for instantiating VNFs of a particular NS DF according to a specific VNFD and VNF DF as defined in ETSI GS NFV-IFA 014 [2]. Table 9.2.8.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.8.1-1: Type name, shorthand, and URI

Shorthand Name	VnfProfile
Type Qualified Name	toscanfv:VnfProfile
Type URI	tosca.datatypes.nfv.VnfProfile

9.2.8.2 Properties

The properties of the VnfProfile data type shall comply with the provisions set out in table 9.2.8.2-1.

Table 9.2.8.2-1: Properties

Name	Required	Type	Constraints	Description
instantiation_level	no	string		Identifier of the instantiation level of the VNF DF to be used for instantiation. See note 3.
target_vnf_scale_level_info	no	map of toscadataypes.nfv.ScaleInfo		For each scaling aspect of the current VNF deployment flavour, it specifies the scale level of VNF constituents (e.g. VDU level) to be instantiated. If the property is present it shall contain all scaling aspects. This property is applicable if VNF supports target scale level instantiation. See note 3.
min_number_of_instances	yes	integer	greater_or_equal: 0	Minimum number of instances of the VNF based on this VNFD that is permitted to exist for this VnfProfile.
max_number_of_instances	yes	integer	greater_or_equal: 0	Maximum number of instances of the VNF based on this VNFD that is permitted to exist for this VnfProfile.
service_availability_level	no	integer	greater_or_equal: 1	If present, specifies the service availability level for the VNF instance created from this profile. See note 1.
version_dependency	no	list of toscadataypes.nfv.VersionDependency		Identifies versions of descriptors of other constituents in the NSD upon which the VNF depends. The dependencies may be described for the VNFD referenced with descriptor_id in the VNF node where this profile is defined and for VNFDs with the same ext_invariant_id. There shall not be more than one version_dependency present with the same dependent_constituent_id. See note 2.
selected_deployable_modules	no	tosca.dataatypes.nfv.SelectedDeployableModules		Indicates the selected deployable module(s) for the VNF instances created from this profile.
vdu_capacity-overriding-allowed	no	map of boolean		Indicates whether the capacity related attributes specified in the VNFD as being configurable can be overridden by new values at run time. See note 4. The key is the name of a Vdu.OsContainerDeployableUnit or a Vdu.Compute node template in the VNFD or the string "ALL" if the value applies to all VDUs in the VNFD. If the value is FALSE for a particular VDU, the capacity related attributes specified in the VNFD as being configurable cannot be overridden by new values at run time. If the value is TRUE or no value is indicated for the VDU nor for the "ALL" key overriding is allowed. If there is an entry for the "ALL" key and also an entry for a specific VDU, the latter takes precedence for that VDU.

NOTE 1:	The value '1' expresses the highest service availability level.
NOTE 2:	This property does not indicate any ordering requirement, i.e. the dependent VNF instance and the other constituents upon which this VNF instance depends may be created or incorporated to the NS in any order, unless specified otherwise by the 'dependency' requirement in the node templates in the NSD.
NOTE 3:	The target size for VNF instantiation is either expressed as an instantiation level of that deployment flavour as defined in the VNFD, or alternatively as target_vnf_scale_level_info, but not both. If none of the two attributes (instantiation_level or target_vnf_scale_level_info) are present, the default instantiation level as declared in the VNFD shall be used.
NOTE 4:	Neither the present document version, nor current versions of referenced documents specify the means for overriding the values at runtime.

9.2.8.3 Definition

The syntax of the VnfProfile data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfProfile:
  derived_from: tosca.datatypes.Root
  description: describes a profile for instantiating VNFs of a particular NS DF
  according to a specific VNFD and VNF DF.
  properties:
    instantiation_level:
      type: string
      description: Identifier of the instantiation level of the VNF DF to be used
      for instantiation. If not present, the default instantiation level as declared in the
      VNFD shall be used.
      required: false
    target_vnf_scale_level_info:
      type: map # key: aspectId
      description: For each scaling aspect of the current VNF deployment flavour, it
      specifies the scale level of VNF constituents (e.g. VDU level) to be instantiated. If
      the property is present it shall contain all scaling aspects
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.ScaleInfo
    min_number_of_instances:
      type: integer
      description: Minimum number of instances of the VNF based on this VNFD that is
      permitted to exist for this VnfProfile.
      required: true
      constraints:
        - greater_or_equal: 0
    max_number_of_instances:
      type: integer
      description: Maximum number of instances of the VNF based on this VNFD that is
      permitted to exist for this VnfProfile.
      required: true
      constraints:
        - greater_or_equal: 0
    service_availability_level:
      type: integer
      description: Specifies the service availability level for the VNF instance
      created from this profile.
      required: false
      constraints:
        - greater_or_equal: 1
    version_dependency:
      type: list
      description: Identifies versions of descriptors of other constituents in the
      NSD upon which the VNF depends. The dependencies may be described for the VNFD
      referenced with descriptor_id in the VNF node where this profile is defined and for
      VNFDs with the same ext_invariant_id.

```

```

    required: false
    entry_schema:
      type: toska.datatypes.nfv.VersionDependency
  selected_deployable_modules:
    type: toska.datatypes.nfv.SelectedDeployableModules
    description: Indicates the selected deployable module(s) for the VNF instances
created from this profile.
    required: false
    vdu_capacity-overriding-allowed:
      type: map # key: name of a Vdu.OsContainerDeployableUnit or a Vdu.Compute node
template in the VNFD or the string "ALL" if the value applies to all VDUs in the VNFD.
      description: Indicates whether the capacity related attributes specified in
the VNFD as being configurable can be overridden by new values at run time.
      required: false
      entry-schema:
        type: boolean

```

9.2.8.4 Example

None.

9.2.8.5 Additional Requirements

None.

9.2.9 toska.datatypes.nfv.VnfMonitoringParameter

9.2.9.1 Description

This data type provides information on virtualised resource related performance metrics applicable to VNF. Table 9.2.9.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.9.1-1: Type name, shorthand, and URI

Shorthand Name	VnfMonitoringParameter
Type Qualified Name	toscanfv:VnfMonitoringParameter
Type URI	tosca.datatypes.nfv.VnfMonitoringParameter

9.2.9.2 Properties

The properties of the VnfMonitoringParameter data type shall comply with the provisions set out in table 9.2.9.2-1.

Table 9.2.9.2-1: Properties

Name	Required	Type	Constraints	Description
name	no	string		Human readable name of the monitoring parameter.
performance_metric	yes	string	valid values: See YAML definition constraints	Identifies a performance metric to be monitored. Performance metric values shall be set to a measurement name defined in clause 7.2 of ETSI GS NFV-IFA 027 [7], without appending a sub-counter. In this case the VNFM computes these measurements from lower-level metrics collected from the VIM or CISM.
collection_period	no	scalar-unit.time		Describes the periodicity at which to collect the performance information.

9.2.9.3 Definition

The syntax of the VnfMonitoringParameter data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfMonitoringParameter:
  derived_from: toska.datatypes.Root
  description: Represents information on virtualised resource related performance
metrics applicable to the VNF.
  properties:
    name:
      type: string
      description: Human readable name of the monitoring parameter
      required: true
    performance_metric:
      type: string
      description: Identifies a performance metric to be monitored, according to
ETSI GS NFV-IFA 027.
      required: true
      constraints:
        - valid_values: [ v_cpu_usage_mean_vnf, v_cpu_usage_peak_vnf,
v_memory_usage_mean_vnf, v_memory_usage_peak_vnf, v_disk_usage_mean_vnf,
v_disk_usage_peak_vnf, byte_incoming_vnf_ext_cp, byte_outgoing_vnf_ext_cp,
packet_incoming_vnf_ext_cp, packet_outgoing_vnf_ext_cp, cpu_usage_mean_os_container,
cpu_usage_peak_os_container, memory_usage_mean_os_container,
memory_usage_peak_os_container, byte_incoming_compute_mcio,
byte_outgoing_compute_mcio, packet_incoming_compute_mcio,
packet_outgoing_compute_mcio, storage_usage_mean_os_container,
storage_usage_peak_os_container, energy_consumption_vnf, energy_consumption_mean_vnf ]
    collection_period:
      type: scalar-unit.time
      description: Describes the periodicity at which to collect the performance
information.
      required: false
      constraints:
        - greater_than: 0 s

```

9.2.9.4 Examples

See clause A.8.

9.2.9.5 Additional Requirements

None.

9.2.10 toska.datatypes.nfv.VersionDependency

9.2.10.1 Description

The VersionDependency data type describes all dependencies that an NSD constituent has on the versions of other NSD constituents. Table 9.2.10.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.10.1-1: Type name, shorthand and URI

Shorthand Name	VersionDependency
Type Qualified Name	toscanfv:VersionDependency
Type URI	tosca.datatypes.nfv.VersionDependency

9.2.10.2 Properties

The properties of the VersionDependency data type shall comply with the provisions set out in table 9.2.10.2-1.

Table 9.2.10.2-1: Properties

Name	Required	Type	Constraints	Description
dependent_constituent_id	yes	string		Identifier of the NSD constituent which has version dependencies on other NSD constituents.
version_dependency_statement	yes	list of toska.datatypes.nfv.VersionDependencyStatement		Identifies one or multiple versions of an NSD constituent upon which the dependent constituent identified by dependent_constituent_id has a dependency.

9.2.10.3 Definition

The syntax of the VersionDependency data type shall comply with the following definition:

```
tosca.datatypes.nfv.VersionDependency:
  derived_from: toska.datatypes.Root
  description: Describes and associates the protocol layer that a CP uses together
with other protocol and connection point information.
  properties:
    dependent_constituent_id:
      type: string
      description: Identifier of the NSD constituent which has version dependencies
on other NSD constituents.
      required: true
    version_dependency_statement:
      type: list
      description: Identifies one or multiple versions of an NSD constituent upon
which the dependent constituent identified by dependent_constituent_id has a
dependency.
      entry_schema:
        type: toska.datatypes.nfv.VersionDependencyStatement
        required: true
```


9.2.10.4 Examples

Every `version_dependency_statement` shall be fulfilled if the NS constituent based on the descriptor referenced by `dependent_constituent_id` is present in an NS instance. If more than one `descriptor_id` is indicated in a particular `version_dependency_statement`, the `version_dependency_statement` is fulfilled if at least one NS constituent based on one of the `descriptors_ids` referenced in the `version_dependency_statement` is present in the NS instance.

The following example shows an excerpt of the FrontEnd VNF node template definition, which is part of an NSD service template. The FrontEnd VNF has a version dependency with two elements of the `version_dependency_statement` list.

The first element in the `version_dependency_statement` list indicates a dependency to a version 1.0 of the BackEnd VNF.

The second element in the `version_dependency_statement` list indicates a dependency to either version 1.0 or 1.1 of the DB VNF.

In order to fulfil the dependency of the FrontEnd VNF in an instance of the NS:

- An instance of the BackEnd VNF v1.0 shall be part of the NS instance
- An instance of the DB VNF, either v1.0 or v1.1, shall be part of the NS instance

```
FrontEnd:
  type: MyCompany.FrontEnd_1_0
  properties:
    descriptor_id: d8ec18e0-2002-42c5-8758-ebf03b52103d
    # other properties omitted
  vnf_profile:
    # other properties omitted
    version_dependency:
      - dependent_constituent_id: d8ec18e0-2002-42c5-8758-ebf03b52103d
        version_dependency_statement:
          - descriptor_id: 028df007-7d7d-4f8a-aaec-db01d39f37b7 # VNFD of BackEnd
v1.0.
          - descriptor_id: [ df1e20a4-88bf-4e49-bd2d-9b1800e6b48b, e1bc5e71-ab87-
48e8-b0e0-05e9062534d8 ] # VNFDs of DB v1.0 and v1.1.
```

9.2.10.5 Additional Requirements

None.

9.2.11 `tosca.datatypes.nfv.VersionDependencyStatement`

9.2.11.1 Description

The `VersionDependencyStatement` data type lists one or more VNF, NS or PNF descriptor identifiers which describe one single dependency. Table 9.2.11.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.11.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VersionDependencyStatement</code>
Type Qualified Name	<code>toscanfv:VersionDependencyStatement</code>
Type URI	<code>tosca.datatypes.nfv.VersionDependencyStatement</code>

9.2.11.2 Properties

The properties of the VersionDependencyStatement data type shall comply with the provisions set out in table 9.2.11.2-1.

Table 9.2.11.2-1: Properties

Name	Required	Type	Constraints	Description
descriptor_id	yes	list of string		List of identifiers of VNFDs, NSDs or PNFDs upon which the entity using this information element depends. When more than one descriptor is indicated, they shall correspond to versions of the same VNF, NS or PNF and they represent alternatives, i.e. the presence of one of them fulfils the dependency.

9.2.11.3 Definition

The syntax of the VersionDependencyStatement data type shall comply with the following definition:

```

tosca.datatypes.nfv.VersionDependencyStatement:
  derived_from: toska.datatypes.Root
  description: Lists one or more VNF, NS or PNF descriptor identifiers which
describe one single dependency.
  properties:
    descriptor_id:
      description: List of identifiers of VNFDs, NSDs or PNFDs upon which the entity
using this information element depends. When more than one descriptor is indicated,
they shall correspond to versions of the same VNF, NS or PNF and they represent
alternatives, i.e. the presence of one of them fulfils the dependency.
      type: list
      entry_schema:
        type: string
      required: true

```

9.2.11.4 Examples

See clause 9.2.10.4.

9.2.11.5 Additional Requirements

None.

9.2.12 toska.datatypes.nfv.ScaleInfo

9.2.12.1 Description

The scaleInfo data type indicates for a given scaleAspect the corresponding scaleLevel. Table 9.2.12.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.12.1-1: Type name, shorthand and URI

Shorthand Name	ScaleInfo
Type Qualified Name	toscanfv:ScaleInfo
Type URI	tosca.datatypes.nfv.ScaleInfo

9.2.12.2 Properties

The properties of the ScaleInfo data type shall comply with the provisions set out in table 9.2.12.2-1.

Table 9.2.12.2-1: Properties

Name	Required	Type	Constraints	Description
scale_level	yes	integer	greater_or_equal: 0	The scale level for a particular aspect.

9.2.12.3 Definition

The syntax of the ScaleInfo data type shall comply with the following definition:

```

tosca.datatypes.nfv.ScaleInfo:
  derived_from: toska.datatypes.Root
  description: Indicates for a given scaleAspect the corresponding scaleLevel
  properties:
    scale_level:
      type: integer
      description: The scale level for a particular aspect
      required: true
      constraints:
        - greater_or_equal: 0

```

9.2.12.4 Examples

None.

9.2.12.5 Additional Requirements

None.

9.2.13 toska.datatypes.nfv.SelectedDeployableModules

9.2.13.1 Description

The SelectedDeployableModules data type indicates the selected deployable modules of a VNF, which is a constituent of an NS, as defined in ETSI GS NFV-IFA 011 [1]. Table 9.2.13.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.13.1-1: Type name, shorthand, and URI

Shorthand Name	SelectedDeployableModules
Type Qualified Name	toscanfv:SelectedDeployableModules
Type URI	tosca.datatypes.nfv.SelectedDeployableModules

9.2.13.2 Properties

The properties of the SelectedDeployableModules data type shall comply with the provisions set out in table 9.2.13.2-1.

Table 9.2.13.2-1: Properties

Name	Required	Type	Constraints	Description
deployable_module	no	list of string	min_length: 0	Name(s) of the selected deployable module(s) for the VNF instances created from the applicable VNF or NS profile. See note 1.
overriding_selection_allowed	yes	boolean	default: true	Indicates whether it is allowed or not to override the selection of deployable modules indicated in a property of this data type by means of attributes in an NS LCM operation, or by attributes in the NsProfile if the VNF is part of a nested NS, during the lifecycle of a VNF instance. If not present, overriding the selection is allowed. See note 2.
NOTE 1: Length 0 is applicable in the case where no deployable module is selected. NOTE 2: When this data type is used in the NsProfile data type, overriding the selection indicated with this information element is only allowed by means of attributes in the NS LCM operation. In other words, the selection of deployable modules for constituent VNFs of a nested NS indicated in the NsProfile cannot be overridden in a NsProfile of the parent NS, if this parent NS is in turn a nested NS of another parent NS.				

9.2.13.3 Definition

The syntax of the SelectedDeployableModules data type shall comply with the following definition:

```

tosca.datatypes.nfv.SelectedDeployableModules:
  derived_from: toska.datatypes.Root
  description: Indicates the selected deployable modules of a VNF, which is a
constituent of an NS.
  properties:
    deployable_module:
      type: list
      description: Name(s) of the selected deployable module(s) for the VNF
instances created from the applicable VNF or NS profile.
      required: false
      entry_schema:
        type: string
      constraints:
        - min_length: 0
    overriding_selection_allowed:
      type: boolean
      description: Indicates whether it is allowed or not to override the selection
of deployable modules indicated in a property of this data type by means of attributes
in an NS LCM operation, or by attributes in the NsProfile if the VNF is part of a
nested NS, during the lifecycle of a VNF instance. If not present, overriding the
selection is allowed.
      required: true
      default: true

```

9.2.13.4 Examples

None.

9.2.13.5 Additional Requirements

None.

9.2.14 toska.datatypes.nfv.DomainNameData

9.2.14.1 Description

The DomainNameData data type describes the information on the domain names to be applied to a connection point as defined in ETSI GS NFV-IFA 011 [1]. Table 9.2.14.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.2.14.1-1: Type name, shorthand, and URI

Shorthand Name	DomainNameData
Type Qualified Name	toscanfv:DomainNameData
Type URI	tosca.datatypes.nfv.DomainNameData

9.2.14.2 Properties

The properties of the DomainNameData data type shall comply with the provisions set out in table 9.2.14.2-1.

Table 9.2.14.2-1: Properties

Name	Required	Type	Constraints	Description
domain_name_assignment	yes	string	Valid values: See YAML definition constraints	Specifies which mode is used for the domain name assignment. Description of the valid values: - mano: if domain name information is not included in the CP node, domain name information shall be provided for the VNF by a management entity using the NFV-MANO interfaces towards the VNFM. If domain name information is included in the CP node, domain name information may be provided for the VNF by a management entity using the NFV-MANO interfaces towards the VNFM and it will override the values provided in the CP node. - dynamic: the VNF gets a domain name that is dynamically assigned by the NFVI/VIM/CISM without receiving domain name configuration information from the NFV-MANO interfaces. - vnf_pkg: a domain name defined by the VNF provider is assigned by means included as part of the VNF package (e.g. LCM script). See note 1. - external: a domain name is provided by an external management entity (such as EM) directly towards the VNF.
fully_qualified_domain_name	no	string		Specifies the Fully Qualified Domain Name (FQDN) to apply to the CP. See notes 1 and 2.
relative_domain_name	no	string		Specifies a value of relative domain name to be considered when setting the fully qualified domain name. See note 2.
NOTE 1: In order to avoid conflicts with other domain names, this value is only permitted for Cp nodes without external connectivity, i.e. connectivity outside the VNF, and only in the case of the VNF having a dedicated DNS assigned to it.				
NOTE 2: If a value is provided at design-time, this value may be overridden at run-time based on other deployment requirements or constraints.				

9.2.14.3 Definition

The syntax of the DomainNameData data type shall comply with the following definition:

```

tosca.datatypes.nfv.DomainNameData:
  derived_from: toska.datatypes.Root
  description: Describes the information on the domain names to be applied to a connection point.
  properties:
    domain_name_assignment:
      type: string
      description: Specifies which mode is used for the domain name assignment.
      required: true
      constraints:
        - valid_values: [ mano, dynamic, vnf_pkg, external ]
    fully_qualified_domain_name:
      type: string
      description: Specifies the fully qualified domain name (FQDN) to apply to the
CP.
      required: false
    relative_domain_name:
      type: string
      description: Specifies a value of relative domain name to be considered when
setting the fully qualified domain name.
      required: false

```

9.2.14.4 Examples

```

<some_tosca_entity>:
  properties:
    domain_name:
      domain_name_assignment: external

```

9.2.14.5 Additional Requirements

None.

9.3 Artifact Types

None.

9.4 Capability Types

9.4.1 toska.capabilities.nfv.VirtualLinkable

9.4.1.1 Description

A node type that includes the VirtualLinkable capability indicates that it can be pointed by toska.relationships.nfv.VirtualLinksTo relationship type, which is used to model the association between a VduCpd and an intVirtualLinkDesc and the association between a VnfExtCpd and an intVirtualLinkDesc as specified in ETSI GS NFV-IFA 011 [1] as well as the association represented by the NsVirtualLinkConnectivity information element in ETSI GS NFV-IFA 014 [2]. Table 9.4.1.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.4.1.1-1: Type name, shorthand, and URI

Shorthand Name	VirtualLinkable
Type Qualified Name	toscanfv:VirtualLinkable
Type URI	tosca.capabilities.nfv.VirtualLinkable

9.4.1.2 Properties

None.

9.4.1.3 Definition

The syntax of the VirtualLinkable capability type shall comply with the following definition:

```
tosca.capabilities.nfv.VirtualLinkable:
  derived_from: toska.capabilities.Node
  description: A node type that includes the VirtualLinkable capability indicates
that it can be pointed by toska.relationships.nfv.VirtualLinksTo relationship type
```

9.4.2 Void

9.4.3 toska.capabilities.nfv.AssociablePaasService

9.4.3.1 Description

The AssociablePaasService capability indicates that the node that includes it can be pointed by a toska.relationships.nfv.PaaSServiceAssociates relationship type. Table 9.4.3.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.4.3.1-1: Type name, shorthand, and URI

Shorthand Name	AssociablePaasService
Type Qualified Name	toscanfv:AssociablePaasService
Type URI	tosca.capabilities.nfv.AssociablePaasService

9.4.3.2 Properties

None.

9.4.3.3 Definition

The syntax of the AssociablePaasService capability type shall comply with the following definition:

```
tosca.capabilities.nfv.AssociablePaasService:
  derived_from: toska.capabilities.Node
  description: Indicates that the node that includes it can be pointed by a
toska.relationships.nfv.PaaSServiceAssociates relationship type.
```

9.5 Requirement Types

None.

9.6 Relationship Types

9.6.1 `tosca.relationships.nfv.VirtualLinksTo`

9.6.1.1 Description

This relationship type represents an association between the `VduCp` or the `VnfExtCp` and the `VnfVirtualLink` node types or the association between either the `VnfExtCp`, the `VduCp`, the `VirtualCp`, the `PnfExtCp` or the `Sap` and an `NsVirtualLink` node types. Table 9.6.1.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.6.1.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VirtualLinksTo</code>
Type Qualified Name	<code>toscanfv:VirtualLinksTo</code>
Type URI	<code>tosca.relationships.nfv.VirtualLinksTo</code>

9.6.1.2 Properties

None.

9.6.1.3 Definition

The syntax of the `VirtualLinksTo` relationship type shall comply with the following definition:

```
tosca.relationships.nfv.VirtualLinksTo:
  derived_from: toska.relationships.DependsOn
  description: Represents an association relationship between the VduCp or the
VnfExtCp and the VnfVirtualLink node types or the association between either a
VnfExtCp, a VduCp, a VirtualCp, a PnfExtCp or a Sap and an NsVirtualLink node types.
  valid_target_types: [ toska.capabilities.nfv.VirtualLinkable ]
```

9.6.2 Void

9.6.3 `tosca.relationships.nfv.VipVirtualLinksTo`

9.6.3.1 Description

This relationship type represents an association between the `VipCp` and a `VnfVirtualLink` node types or between the former and an `NsVirtualLink` node types. Table 9.6.3.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.6.3.1-1: Type name, shorthand, and URI

Shorthand Name	<code>VipVirtualLinksTo</code>
Type Qualified Name	<code>toscanfv:VipVirtualLinksTo</code>
Type URI	<code>tosca.relationships.nfv.VipVirtualLinksTo</code>

9.6.3.2 Properties

None.

9.6.3.3 Definition

The syntax of the `VipVirtualLinksTo` relationship type shall comply with the following definition:


```

tosca.relationships.nfv.VipVirtualLinksTo:
  derived_from: toska.relationships.DependsOn
  description: Represents an association relationship between the VipCp and a
VnfVirtualLink node types or between the former and a NsVirtualLink node types.
  valid_target_types: [ toska.capabilities.nfv.VirtualLinkable ]

```

9.6.4 toska.relationships.nfv.PaaSServiceAssociates

9.6.4.1 Description

This relationship type represents an association between the VNF or NS and the NsPaasServiceProfile node types. Table 9.6.4.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.6.4.1-1: Type name, shorthand, and URI

Shorthand Name	PaasServiceAssociates
Type Qualified Name	toscanfv:PaasServiceAssociates
Type URI	tosca.relationships.nfv.PaaSServiceAssociates

9.6.4.2 Properties

None.

9.6.4.3 Definition

The syntax of the PaasServiceAssociates relationship type shall comply with the following definition:

```

tosca.relationships.nfv.PaaSServiceAssociates:
  derived_from: toska.relationships.DependsOn
  description: Represents an association relationship between the VNF or NS and
NsPaasServiceProfile node types
  valid_target_types: [ toska.capabilities.nfv.AssociablePaasService ]

```

9.7 Interface Types

None.

9.8 Node Types

9.8.1 toska.nodes.nfv.Cp

9.8.1.1 Description

A Cp node type represents the Cp information element as defined in ETSI GS NFV-IFA 011 [1], which describes network connectivity to a compute resource or a VL. This is an abstract type used as parent for the various Cp node types. Table 9.8.1.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.8.1.1-1: Type name, shorthand, and URI

Shorthand Name	Cp
Type Qualified Name	toscanfv:Cp
Type URI	tosca.nodes.nfv.Cp

9.8.1.2 Properties

The properties of the Cp node type shall comply with the provisions set out in table 9.8.1.2-1.

Table 9.8.1.2-1: Properties

Name	Required	Type	Constraints	Description
layer_protocols	yes	list of string	Valid values: See YAML definition constraints	Identifies which protocol the connection point uses for connectivity purposes.
ip_stack_mode	no	string	Valid values: See YAML definition constraints	Specifies the capability of the CP to support IP dual stack or tunnelling. Values: • ipv4_xor_ipv6: the CP supports both IPV4 and IPV6 but it can only be configured with IPV4 or with IPV6. • ipv4_or_ipv6: the CP supports both IPV4 and IPV6 and it can be configured with either of them or with both. • tunnel_ipv6_over_ipv4: the CP supports IPv6 tunnelling over IPv4. • tunnel_ipv4_over_ipv6: the CP supports IPv4 tunnelling over IPv6. See notes 1 and 2.
role	no	string	Valid values: See YAML definition constraints	Identifies the role of the port in the context of the traffic flow patterns in the VNF or parent NS. For example a VNF with a tree flow pattern within the VNF will have legal cpRoles of ROOT and LEAF.
description	no	string		Provides human-readable information on the purpose of the connection point (e.g. connection point for control plane traffic).
protocol	no	list of tosa.datatypes.nfv.Cp ProtocolData		Provides information on the addresses to be assigned to the connection point(s) instantiated from this Connection Point Descriptor.
trunk_mode	no	boolean		Information about whether the CP instantiated from this Cp is in Trunk mode (802.1Q or other). When operating in "trunk mode", the Cp is capable of carrying traffic for several VLANs. Absence of this property implies that trunkMode is not configured for the Cp i.e. It is equivalent to boolean value "false".

Name	Required	Type	Constraints	Description
purpose	no	list of string	Valid values: See YAML definition constraints	Identifies in a machine-processable form the purpose of the CP. Values: • cp: for control plane traffic. • dp: for data plane traffic. • mp: for management plane traffic. • mp_external: for management plane traffic to external management entities. See note 3.
domain_name	no	list of <code>tosca.datatypes.nfv.DomainNameData</code>		Provides information on the domain names to be applied to the CP.
NOTE 1: If multiple values are indicated in the <code>layer_protocols</code> property they represent layers of a protocol stack with the top layer first, with the exception of <code>ipv4</code> and <code>ipv6</code> values, when used as consecutive values, in which case the interpretation of their presence shall be specified through the <code>ip_stack_mode</code> property. NOTE 2: The <code>ip_stack_mode</code> property shall be present If <code>layer_protocols</code> indicates both <code>ipv4</code> and <code>ipv6</code> and these are listed as consecutive values and shall be absent otherwise. NOTE 3: This property shall be present when the purpose of the CP is to connect to external management entities (e.g. OSS/EM), i.e. in this case one valid value of this property is "mp_external". Otherwise this property may be absent.				

9.8.1.3 Attributes

None.

9.8.1.4 Requirements

None.

9.8.1.5 Capabilities

None.

9.8.1.6 Definition

The syntax of the Cp node type shall comply with the following definition:

```
tosca.nodes.nfv.Cp:
  derived_from: toska.nodes.Root
  description: Provides information regarding the purpose of the connection point
  properties:
    layer_protocols:
      type: list
      description: Identifies which protocol the connection point uses for
connectivity purposes
      required: true
    entry_schema:
      type: string
      constraints:
        - valid_values: [ ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire ]
    ip_stack_mode:
      type: string
      description: Specifies the capability of the CP to support IP dual stack or
tunnelling
      required: false
```

```

    constraints:
      - valid_values: [ ipv4_xor_ipv6, ipv4_or_ipv6, tunnel_ipv6_over_ipv4,
tunnel_ipv4_over_ipv6 ]
    role: #Name in ETSI GS NFV-IFA 011: cpRole
    type: string
    description: Identifies the role of the port in the context of the traffic
flow patterns in the VNF or parent NS
    required: false
    constraints:
      - valid_values: [ root, leaf ]
    description:
    type: string
    description: Provides human-readable information on the purpose of the
connection point
    required: false
    protocol:
    type: list
    description: Provides information on the addresses to be assigned to the
connection point(s) instantiated from this Connection Point Descriptor
    required: false
    entry_schema:
      type: tosca.datatypes.nfv.CpProtocolData
    trunk_mode:
    type: boolean
    description: Provides information about whether the CP instantiated from this
Cp is in Trunk mode (802.1Q or other). When operating in "trunk mode", the Cp is
capable of carrying traffic for several VLANs. Absence of this property implies that
trunkMode is not configured for the Cp i.e. It is equivalent to boolean value "false".
    required: false
    purpose:
    type: list
    entry_schema:
      type: string
      constraints:
        - valid_values: [ cp, dp, mp, mp_external ]
    description: Identifies in a machine-processable form the purpose of the CP.
    required: false
    domain_name:
    type: list
    description: Provides information on the domain names to be applied to the CP.
    required: false
    entry_schema:
      type: tosca.datatypes.nfv.DomainNameData

```

9.8.1.7 Additional requirements

The 'protocol' property shall not be included in a derived PnfExtCp node and shall be included in all other cases.

9.9 Group Types

None.

9.10 Policy Types

9.10.1 tosa.policies.nfv.Abstract.SecurityGroupRule

9.10.1.1 Description

The Abstract.SecurityGroupRule type represents an abstract policy type without any target requirements. Table 9.10.1.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].

Table 9.10.1.1-1: Type name, shorthand, and URI

Shorthand Name	Abstract.SecurityGroupRule
Type Qualified Name	toscanfv:Abstract.SecurityGroupRule
Type URI	tosca.policies.nfv.Abstract.SecurityGroupRule

9.10.1.2 Properties

The properties of the Abstract.SecurityGroupRule policy type shall comply with the provisions set out in table 9.10.1.2-1.

Table 9.10.1.2-1: Properties

Name	Required	Type	Constraints	Description
description	no	string		Human readable description of the security group rule.
direction	yes	string	ingress, egress	The direction in which the security group rule is applied. The direction of 'ingress' or 'egress' is specified against the associated CP. I.e. 'ingress' means the packets entering a CP, while 'egress' means the packets sent out of a CP.
ether_type	yes	string	ipv4, ipv6	Indicates the protocol carried over the Ethernet layer.
protocol	yes	string	see note	Indicates the protocol carried over the IP layer. Permitted values: any protocol defined in the IANA protocol registry [19], e.g. TCP, UDP, ICMP, etc.
port_range_min	yes	integer	0 - 65 535	Indicates minimum port number in the range that is matched by the security group rule. If a value is provided at design-time, this value may be overridden at run-time based on other deployment requirements or constraints.
port_range_max	yes	integer	0 - 65 535	Indicates maximum port number in the range that is matched by the security group rule. If a value is provided at design-time, this value may be overridden at run-time based on other deployment requirements or constraints.
NOTE: "protocol" constraints values: hopopt, icmp, igmp, ggp, ipv4, st, tcp, cbt, egp, igp, bbn_rcc_mon, nvp_ii, pup, argus, emcon, xnet, chaos, udp, mux, dcn_meas, hmp, prm, xns_idp, trunk_1, trunk_2, leaf_1, leaf_2, rdp, irtp, iso_tp4, netblt, mfe_nsp, merit_inp, dccp, 3pc, idpr, xtp, ddp, idpr_cmt, tp++, il, ipv6, sdrp, ipv6_route, ipv6_frag, idrp, rsvp, gre, dsr, bna, esp, ah, i_nls, swipe, narp, mobile, tisp, skip, ipv6_icmp, ipv6_no_nxt, ipv6_opts, cftp, sat_expak, kryptolan, rvd, ippc, sat_mon, visa, ipcv, cpnx, cphb, wsn, pvp, br_sat_mon, sun_nd, wb_mon, wb_expak, iso_ip, vmtp, secure_vmtp, vines, ttp, iptm, nsfnet_igp, dgp, tcf, eigrp, ospfigp, sprite_rpc, larp, mtp, ax.25, ipip, micp, scc_sp, etherip, encap, gmt, ifmp, pnni, pim, aris, scps, qnx, a/n, ip_comp, snp, compaq_peer, ipx_in_ip, vrrp, pgm, l2tp, ddx, iatp, stp, srp, uti, smp, sm, ptp, isis, fire, crtp, crudp, sscopmce, iplt, sps, pipe, sctp, fc, rsvp_e2e_ignore, mobility, udp_lite, mpls_in_ip, manet, hip, shim6, wesp, rohc.				

9.10.1.3 Definition

The syntax of the Abstract.SecurityGroupRule policy type shall comply with the following definition:

```

tosca.policies.nfv.Abstract.SecurityGroupRule:
  derived_from: tosca.policies.Root
  description: The Abstract.SecurityGroupRule type represents an abstract policy
type without any target requirements
  properties:
    description:
      type: string
      description: Human readable description of the security group rule.
      required: false
    direction:
      type: string
      description: The direction in which the security group rule is applied.
The direction of 'ingress' or 'egress' is specified against the associated CP.
I.e. 'ingress' means the packets entering a CP, while 'egress' means the packets
sent out of a CP.
      required: true
      constraints:
        - valid_values: [ ingress, egress ]
      default: ingress
    ether_type:
      type: string
      description: Indicates the protocol carried over the Ethernet layer.
      required: true
      constraints:
        - valid_values: [ ipv4, ipv6 ]
      default: ipv4
    protocol:
      type: string
      description: Indicates the protocol carried over the IP layer. Permitted
values include any protocol defined in the IANA protocol registry, e.g. TCP, UDP,
ICMP, etc.
      required: true
      constraints:
        - valid_values: [ hopopt, icmp, igmp, ggp, ipv4, st, tcp, cbt, egg, igp,
bbn_rcc_mon, nvp_ii, pup, argus, emcon, xnet, chaos, udp, mux, dcn_meas, hmp, prm,
xns_idp, trunk_1, trunk_2, leaf_1, leaf_2, rdp, irtp, iso_tp4, netblt, mfe_nsp,
merit_inp, dccp, 3pc, idpr, xtp, ddp, idpr_cmtp, tp++, il, ipv6, sdrp, ipv6_route,
ipv6_frag, idrp, rsvp, gre, dsr, bna, esp, ah, i_nlsp, swipe, narp, mobile, tlsp,
skip, ipv6_icmp, ipv6_no_nxt, ipv6_opts, cftp, sat_expak, kryptolan, rvd, ippc,
sat_mon, visa, ipcv, cpnx, cphb, wsn, pvp, br_sat_mon, sun_nd, wb_mon, wb_expak,
iso_ip, vmtp, secure_vmtp, vines, ttp, iptm, nsfnet_igp, dgp, tcf, eigrp, ospfigp,
sprite_rpc, larp, mtp, ax.25, ipip, micp, scc_sp, etherip, encap, gmtp, ifmp,
pnni, pim, aris, scps, qnx, a/n, ip_comp, snp, compaq_peer, ipx_in_ip, vrrp, pgm,
l2tp, ddx, iatp, stp, srp, uti, smp, ptp, isis, fire, crtp, crudp, sscompce,
iplt, sps, pipe, sctp, fc, rsvp_e2e_ignore, mobility, udp_lite, mpls_in_ip, manet,
hip, shim6, wesp, rohc ]
      default: tcp
    port_range_min:
      type: integer
      description: Indicates minimum port number in the range that is matched by
the security group rule. If a value is provided at design-time, this value may be
overridden at run-time based on other deployment requirements or constraints.
      required: true
      constraints:
        - greater_or_equal: 0
        - less_or_equal: 65535
      default: 0

```

```
port_range_max:
  type: integer
  description: Indicates maximum port number in the range that is matched by
the security group rule. If a value is provided at design-time, this value may be
overridden at run-time based on other deployment requirements or constraints.
  required: true
  constraints:
    - greater_or_equal: 0
    - less_or_equal: 65535
  default: 65535
```

9.10.1.4 Additional Requirements

The design of security group rule follows a permissive model where all security group rules applied to a connection point are dealt with in an "OR" logic fashion, i.e. the traffic is allowed if it matches any security group rule applied to this connection point.

Annex A (informative): Examples

A.1 Deployment flavour design mapping

A.1.1 Introduction

This clause describes the main design principle for VNF/NS deployment flavour and the mapping between VNF/NS deployment flavour elements to TOSCA concept.

A.1.2 Design principle for VNF deployment flavour

Each Deployment flavour as specified in ETSI GS NFV-IFA 011 [1] describes a given deployment configuration of a VNF in terms of its internal topology and resource needs. Different deployment flavours can define different topologies of the same VNF, with different scalingAspect, different VDUs and different internal connectivity requirements. The idea of VNF deployment flavour as specified in [1] is that each deployment flavour describes the required vduProfiles and virtualLinkProfiles which are the additional instantiation data for the given VDUs and virtualLinks, once a specific deployment flavour has been chosen at the instantiation time, in order to successfully deploy the given VDUs and virtualLinks, the bindings between the VDU with the corresponding VduProfile and the virtualLinkDesc with the corresponding virtualLinkProfile are required.

To achieve the concept of deployment flavour by using TOSCA, the main design principle is to describe each deployment flavour as a standalone implementable TOSCA service template, and binding VDU and virtualLink with the corresponding VduProfile and virtualLinkProfile, respectively, together at the design time. Once a specific deployment flavour has been chosen at the instantiation time, the corresponding TOSCA service template will be used for deploying the VNF with the given deployment flavour.

Figure A.1.2-1 shows the general principle for a VNF deployment flavour design.

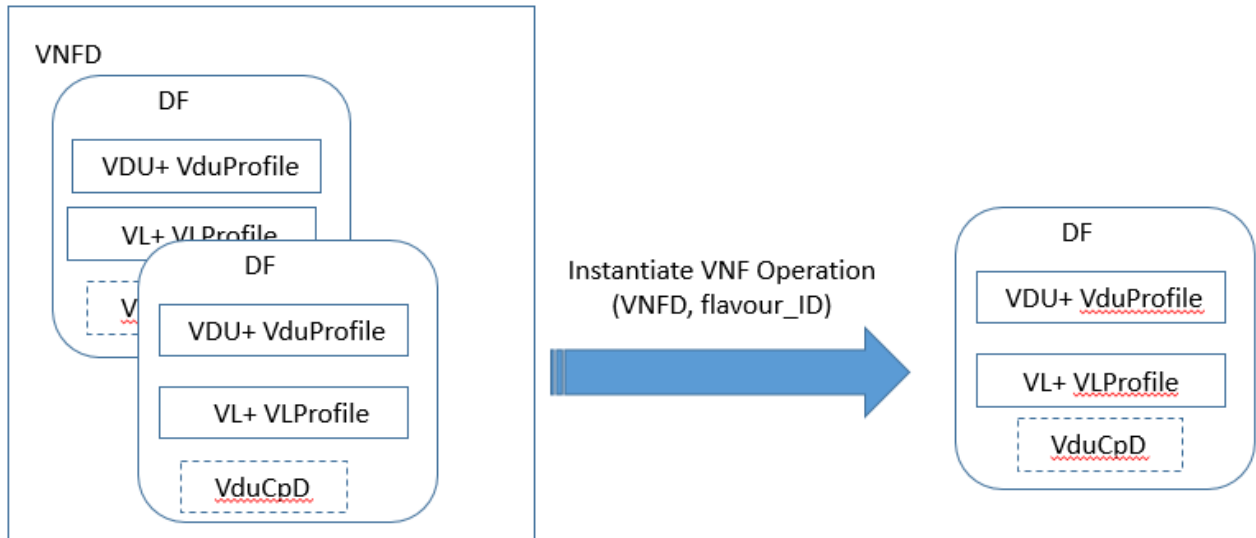


Figure A.1.2-1: General principle for a VNF deployment flavour design

A.1.3 Design principle for NS deployment flavour

The design principle for NSD deployment flavour is the same with the VNF deployment flavour design. Each NS deployment flavour is described as a standalone implementable TOSCA service template, the constituent VNF, virtualLink, nested NS and PNF is bound with the corresponding VnfProfile, VirtualLinkProfile, NsProfile and virtual_link requirements respectively together at the design time. Once a specific deployment flavour has been chosen at the instantiation time, the corresponding TOSCA service template will be used for deploying the NS with the given deployment flavour.

Figure A.1.3-1 shows the general principle for a NS deployment flavour design.

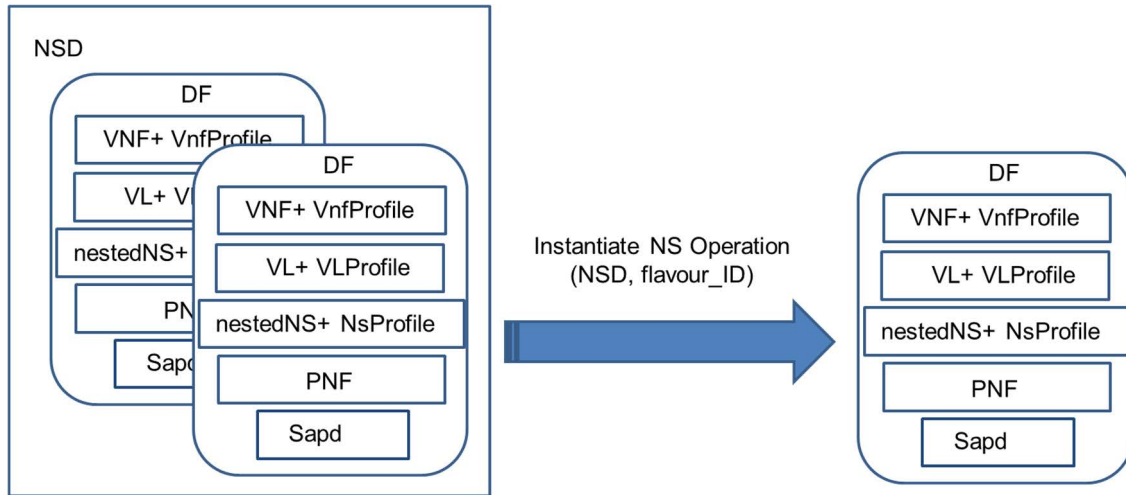


Figure A.1.3-1: General principle for a NS deployment flavour design

A.2 VNFD with deployment flavour modelling design example

Deployment flavours are represented as deployable TOSCA topology templates. This way one VNF service template represents one deployment flavour, and different deployment flavours are described by different VNF service templates. This is in line with the idea that different deployment flavours can define different topologies of the same VNF, with different scaling aspects, different VDUs and different internal connectivity.

In order to represent a VNF a top-level service template is used. This top-level service template contains a topology template with only an abstract VNF node which defines the common parts of the different deployment flavours (such as product information, modifiable attributes and parts of the lifecycle management interface definition). It also sets a constraint on the deployment flavour property (the required value of the flavour_id property); this constraint comes from the VNF instantiation request which contains a flavour_id selected among those available in the VNFD.

As a result, the VNFM will look into the available further service templates representing the different VNF deployment flavours of the VNF and use the one that has the matching flavour_id property value to substitute for the abstract VNF. These are the low-level service templates.

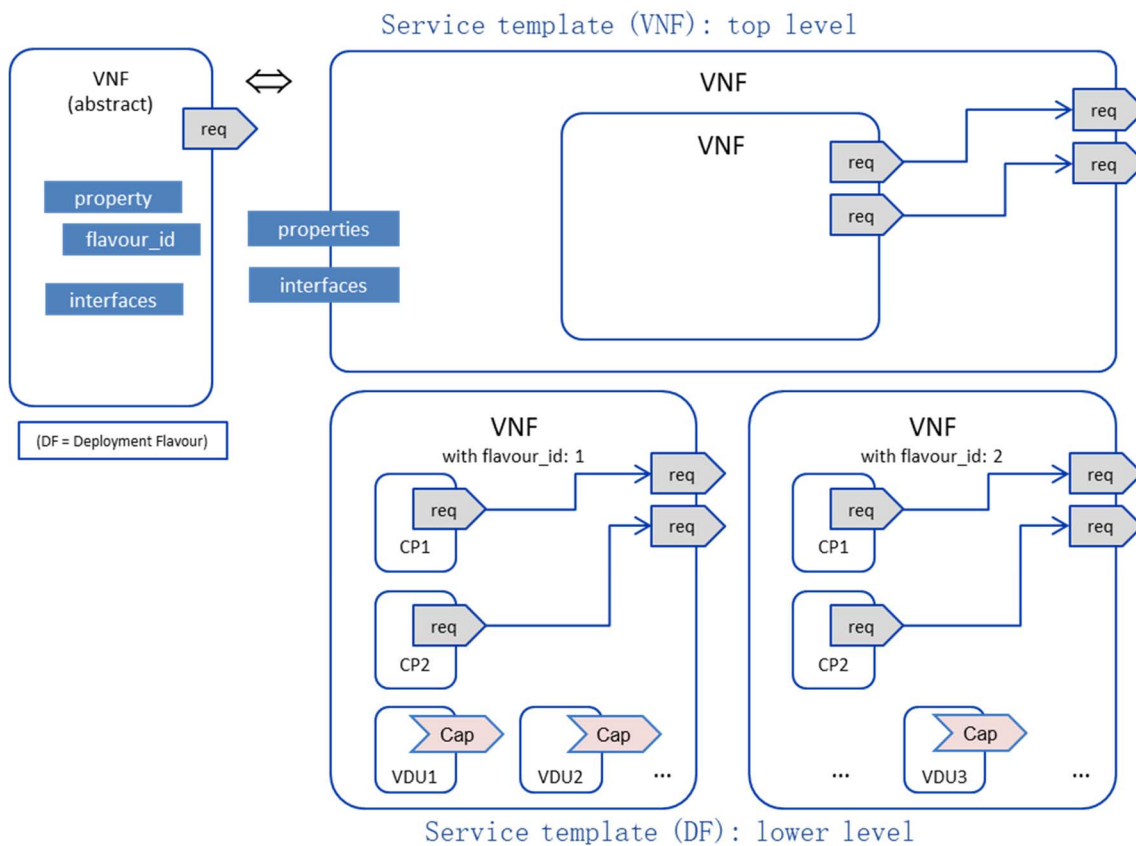


Figure A.2-1: VNFD overview with deployment flavour

A VNFD contains a TOSCA top-level Service Template as entry point in the VNF package and one or more TOSCA low-level Service templates representing the different deployment flavours (see figure A.2-1). The VNFD is interpreted by an NFVO or VNF manager. In this example, the templates describe two variants of the VNF each corresponding to a deployment flavour: a simple one and a complex one. The simple VNF consists of one server: a DB backend whereas, the complex VNF variant consists of minimum three DB backend servers and one serviceNode, which may be scaled out in one-size increments.

SunshineDB: VNFD-top level

sunshine.vnfd.tosca.yaml

```
tosca_definitions_version: tosca_simple_yaml_1_3

description: Relational database, non-scalable
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001
  - sunshineVNF.yaml # contains the VNF node type definition

topology_template:
  inputs:
    flavour_id:
      type: string
      description: VNF deployment flavour selected by the consumer. It is provided in
the API

  node_templates:
    SunshineDB:
      type: MyCompany.SunshineDB.1_0.1_0
      properties:
```

```

    flavour_id: { get_input: flavour_id }
  # requirements:
    #- virtual_link_backend # mapped in lower-level templates
    #- virtual_link_service # mapped in lower-level templates

```

The sunshineVNF.yaml file has the following content:

```

tosca_definitions_version: tosca_simple_yaml_1_3

description: Relational database, non-scalable

imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001

data_types:
  MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters:
    derived_from: tosca.datatypes.nfv.VnfOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
        required: true
        default: value_1
      parameter_2:
        type: string
        required: true
        default: value_2

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: SunshineDB ]
        default: SunshineDB
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      flavour_id:
        type: string
        constraints: [ valid_values: [ simple, complex ] ]
        default: simple
      flavour_description:
        type: string
        default: "" #empty string

```

```

vnfm_info:
  type: list
  entry_schema:
    type: string
  constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
  default: [ '0:MyCompany-1.0.0' ]
requirements:
- virtual_link:
  capability: tosa.capabilities.nfv.VirtualLinkable
  relationship: tosa.relationships.nfv.VirtualLinksTo
  occurrences: [ 0, 0 ]
- virtual_link_backend:
  capability: tosa.capabilities.nfv.VirtualLinkable
  relationship: tosa.relationships.nfv.VirtualLinksTo
  occurrences: [ 0, 1 ]
- virtual_link_service:
  capability: tosa.capabilities.nfv.VirtualLinkable
  relationship: tosa.relationships.nfv.VirtualLinksTo
  occurrences: [ 0, 1 ]
interfaces:
  Vnflcm:
    type: tosa.interfaces.nfv.Vnflcm
    operations:
      instantiate:
        inputs:
          additional_parameters:
            type: MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters
            required: false

```

The vnf node template in the `sunshine.vnfd.tosca.yaml` file is abstract and is subject to substitution; the lower-level templates in the subsequent sections provide these substitutions. The actual lower-level template is selected based on the node type and a value constraint on the `flavour_id` property.

Each low level service template contains a node template of type `MyCompany.SunshineDB.1_0.1_0` with implementation of the LCM interfaces.

SunshineDB (simple): Lower level

This example illustrates one `Vdu.Compute` nodes (`dbBackend`) with two connection points and two virtual links (see figure A.2-2). The `flavour_id` is "Simple".

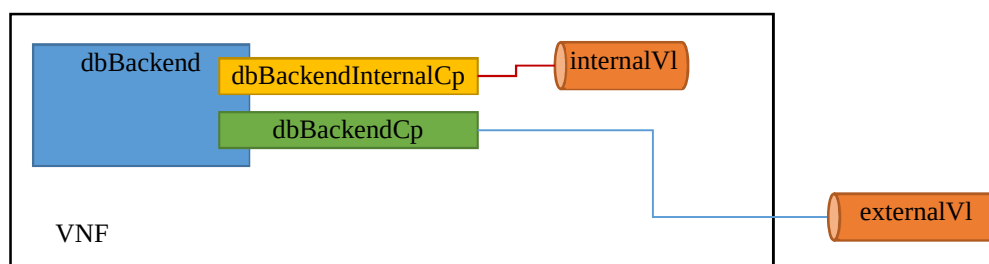


Figure A.2-2: SunshineDB (simple): Lower level

`sunshinedbsimple.vnfd.tosca.yaml`

```

tosca_definitions_version: tosa_simple_yaml_1_3

description: Relational database, simple
imports:

```

```
- etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
GS NFV-SOL 001
- sunshineVNF.yaml # contains the VNF node type definition
```

```
node_types:
```

```
  MyCompany.nodes.nfv.Vdu.Aux:
    derived_from: toska.nodes.nfv.Vdu.Compute
    properties:
      configurable_properties:
        type: MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties
        required: false
```

```
data_types:
```

```
  MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties:
    derived_from: toska.datatypes.nfv.VnfcConfigurableProperties
    properties:
      additional_vnfc_configurable_properties:
        type: MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties
        required: true
```

```
  MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties:
    derived_from: toska.datatypes.nfv.VnfcAdditionalConfigurableProperties
    properties:
      name_prefix_in_vim:
        type: string
        required: true
        default: "MyCustomer"
      dns_server:
        type: string
        required: true
        default: "90.200.250.57"
```

```
topology_template:
```

```
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    substitution_filter:
      properties:
        - flavour_id: { equal: simple }
    requirements:
      virtual_link_backend: [ dbBackendCp, virtual_link ] # IPv4 for SQL
```

```
node_templates:
```

```
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    properties:
      flavour_description: A simple flavour
    interfaces:
      Vnflcm:
        operations:
          instantiate:
            implementation: instantiate.workbook.mistral.yaml
          terminate:
            implementation: terminate.workbook.mistral.yaml
```

```
dbBackend:
```

```
  type: MyCompany.nodes.nfv.Vdu.Aux
  properties:
    name: dbBackend
    description: dbBackend compute node
    nfvi_constraints:
      key_1: value_1
```

```

    key_2: value_2
  vdu_profile:
    min_number_of_instances: 1
    max_number_of_instances: 1
  capabilities:
    virtual_compute:
      properties:
        virtual_memory:
          virtual_mem_size: 8192 MiB
        virtual_cpu:
          cpu_architecture: x86
          num_virtual_cpu: 2
          virtual_cpu_clock: 1800 MHz
  requirements:
    - virtual_storage: mariaDbStorage

mariaDbStorage:
  type: toska.nodes.nfv.Vdu.VirtualBlockStorage
  properties:
    virtual_block_storage_data:
      size_of_storage: 100 GB
      rdma_enabled: true
# ..
artifacts:
  sw_image:
    type: toska.artifacts.nfv.SwImage
    file: maria.db.image.v1.0.qcow2
    properties:
      name: Software of Maria Db
      version: '1.0'
      checksum:
        algorithm: sha-256
        hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d
      container_format: bare
      disk_format: qcow2
      min_disk: 2 GB
      min_ram: 8192 MiB
      size: 2 GB
      operating_system: Linux
      supported_virtualisation_environments:
        - KVM
dbBackendCp:
  type: toska.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: External connection point to access the DB on IPv4
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
  requirements:
    - virtual_binding: dbBackend
    # - virtual_link: # the target node is determined in the NSD

dbBackendInternalCp:
  type: toska.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: Internal connection point on an VL
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false

```

```

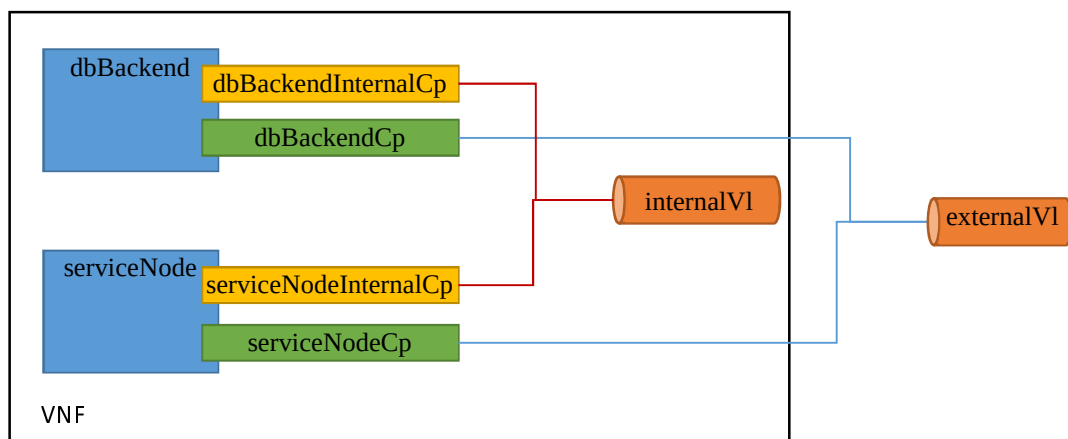
requirements:
  - virtual_binding: dbBackend
  - virtual_link: internalVl

internalVl:
  type: toska.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
      flow_pattern: mesh
    description: Internal Virtual link in the VNF
    vl_profile:
      max_bitrate_requirements:
        root: 100000
        leaf: 20000
      min_bitrate_requirements:
        root: 10000
        leaf: 10000

```

SunshineDB (complex): Lower level

This example illustrates two Vdu.Compute nodes (dbBackend and serviceNode) with two connection points and two virtual links (see figure A.2-3). The flavour_id is "complex".



NOTE: The single external VL above illustrates that both serviceNodeCp and dbBackendCp are connected to the same external VL. Alternatively, external connection points can be connected to separate external VLs.

Figure A.2-3: SunshineDB (complex): Lower level

sunshinedbcomplex.vnfd.tosca.yaml

```

tosca_definitions_version: toska_simple_yaml_1_3

description: Relational database, complex
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001
  - sunshineVNF.yaml # contains the VNF node type definition

node_types:
  MyCompany.nodes.nfv.Vdu.Aux:
    derived_from: toska.nodes.nfv.Vdu.Compute
    properties:

```

```

    configurable_properties:
      type: MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties
      required: false

data_types:
  MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcConfigurableProperties
    properties:
      additional_vnfc_configurable_properties:
        type: MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties
        required: true

  MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties
    properties:
      name_prefix_in_vim:
        type: string
        required: true
        default: "MyCustomer"
      dns_server:
        type: string
        required: true
        default: "90.200.250.57"

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    substitution_filter:
      properties:
        - flavour_id: { equal: complex }
    requirements:
      virtual_link_backend: [ dbBackendCp, virtual_link ] # IPv4 for SQL
      virtual_link_service: [ serviceNodeCp, virtual_link ] # IPv4 for SSH

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    properties:
      flavour_id : complex
      flavour_description: A complex flavour
    interfaces:
      Vnflcm:
        operations:
          instantiate:
            implementation: instantiate.workbook.mistral.yaml
          terminate:
            implementation: terminate.workbook.mistral.yaml
          heal:
            implementation: heal.workbook.mistral.yaml

  dbBackend:
    type: MyCompany.nodes.nfv.Vdu.Aux
    properties:
      name: dbBackend
      description: dbBackend compute node
      nfvi_constraints:
        key_1: value_1
        key_2: value_2
      vdu_profile:
        min_number_of_instances: 3
        max_number_of_instances: 4

```



```

capabilities:
  virtual_compute:
    properties:
      virtual_memory:
        virtual_mem_size: 8192 MiB
      virtual_cpu:
        cpu_architecture: x86
        num_virtual_cpu: 2
        virtual_cpu_clock: 1800 MHz
  requirements:
    - virtual_storage: mariaDbStorage

serviceName:
  type: toska.nodes.nfv.Vdu.Compute
  properties:
    name: serviceName
    description: brief description about serviceName
    nfvi_constraints:
      key_3: value_3
      key_4: value_4
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 1
  capabilities:
    virtual_compute:
      properties:
        virtual_memory:
          virtual_mem_size: 8192 MiB
        virtual_cpu:
          cpu_architecture: x86
          num_virtual_cpu: 2
          virtual_cpu_clock: 1800 MHz
  requirements:
    - virtual_storage: mariaDbStorage
  artifacts:
    sw_image:
      type: toska.artifacts.nfv.SwImage
      file: maria.db.image.v1.0.qcow2
      properties:
        name: Software of Maria Db
        version: '1.0'
        checksum:
          algorithm: sha-256
          hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d
        container_format: bare
        disk_format: qcow2
        min_ram: 8192 MiB
        size: 2 GB
        operating_system: Linux
        supported_virtualisation_environments:
          - KVM

mariaDbStorage:
  type: toska.nodes.nfv.Vdu.VirtualBlockStorage
  properties:
    virtual_block_storage_data:
      size_of_storage: 100 GB
      rdma_enabled: true

  artifacts:
    sw_image:

```

```
type: toska.artifacts.nfv.SwImage
file: maria.db.image.v1.0.qcow2
properties:
  name: Software of Maria Db
  version: '1.0'
  checksum:
    algorithm: sha-256
    hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d
  container_format: bare
  disk_format: qcow2
  min_disk: 2 GB
  min_ram: 8192 MiB
  size: 2 GB
  operating_system: Linux
  supported_virtualisation_environments:
    - KVM

dbBackendCp:
  type: toska.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: External connection point to access the DB on IPv4
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
  requirements:
    - virtual_binding: dbBackend
    #- virtual_link: # the target node is determined in the NSD

dbBackendInternalCp:
  type: toska.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: Internal connection point on an VL
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
  requirements:
    - virtual_binding: dbBackend
    - virtual_link: internalVL

serviceNameCp:
  type: toska.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: External connection point to access the DB on IPv4
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
  requirements:
    - virtual_binding: serviceName
    #- virtual_link: # the target node is determined in the NSD

serviceNameInternalCp:
  type: toska.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: Internal connection point on VL
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
```

```

requirements:
  - virtual_binding: serviceNode
  - virtual_link: internalVl

internalVl:
  type: toasca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
      flow_pattern: mesh
    description: Internal VL
    vl_profile:
      max_bitrate_requirements:
        root: 100000
        leaf: 20000
      min_bitrate_requirements:
        root: 10000
        leaf: 10000

```

A.3 VNF external connection point

A.3.1 General

A VNF external connection point may either be an internal connection point (a VDU connection point) which is re-exposed externally, i.e. it may be connected to an external virtual link, or a virtual link port. In the latter case the external connection point node has an association relationship of type VirtualLinksTo to the internal virtual link node.

The following clauses illustrate the use of both models of VNF external connection point, re-exposure of an internal connection point and connected to an internal virtual link.

A.3.2 External connection point re-exposing an internal connection point

In this case there is no need for a VnfExtCp node template. When substituting the VNFD low level service template for the abstract VNF node, the virtual_link requirement of the abstract VNF node is mapped to the VDU connection point's virtual_link requirement. This is shown in figure A.3.2-1. In this case the VNF Service Template does not include a node template of type toasca.nodes.nfv.VnfExtCp, but the functionality of it is provided by a VDU connection point.

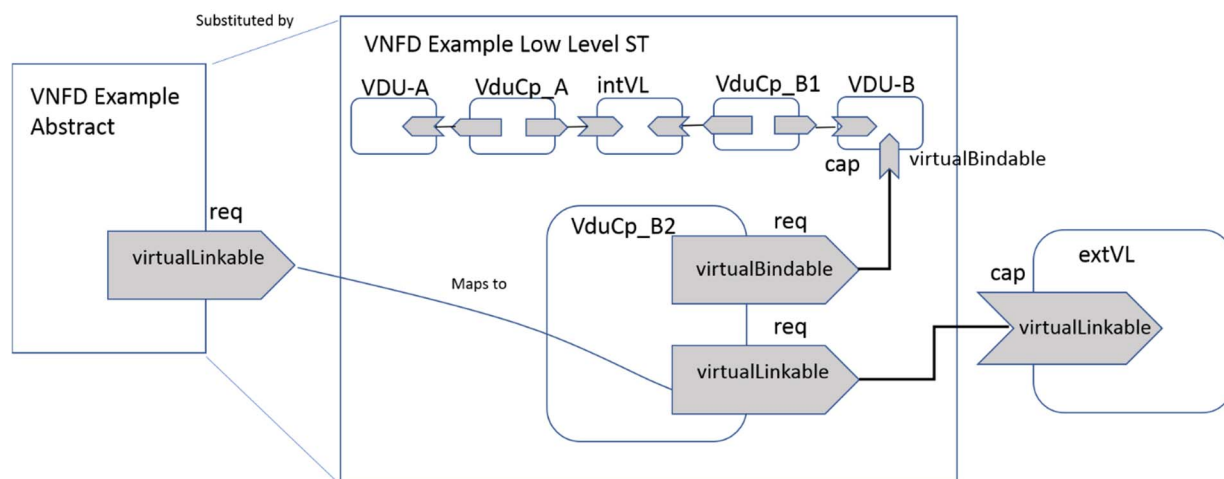


Figure A.3.2-1: VNFD with an VDU connection point acting as VnfExtCp

The following snippet shows the relevant part of the service template:

```
tosca_definitions_version: tosca_simple_yaml_1_3
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001
  - exampleVNF.yaml # contains the VNF node type definition

topology_template:

#...

  substitution_mappings:
    node_type: tosca.nodes.nfv.exampleVNF
    substitution_filter:
      properties:
        - flavour_id: { equal: simple }
    requirements:
      virtual_link: [vduCp_B2, virtual_link]

  node_templates:
#..
    ExampleVNF:
      type: tosca.nodes.nfv.exampleVNF
      properties:
        flavour_description: A simple flavour
      interfaces:
        Vnflcm:
          operations:
            instantiate:
              implementation: instantiate.workbook.mistral.yaml
            terminate:
              implementation: terminate.workbook.mistral.yaml

    vduCp_B2:
      type: tosca.nodes.nfv.VduCp
      properties:
        layer_protocols: [ ipv4 ]
      # other properties omitted for brevity
      requirements:
        - virtual_binding: VDU-B
      # - virtual_link: # mapped to virtual_link requirement of VNF node

# other node template definitions (VDU-A, VDU-B, intVL, etc.):
VDU-A:
  type: tosca.nodes.nfv.Vdu.Compute
  properties:
    name: VDU-A
    description: VDU-A description
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 1
  capabilities:
    virtual_compute:
      properties:
        virtual_memory:
          virtual_mem_size: 1 GB
```

```
    virtual_cpu:
      num_virtual_cpu: 1

vduCp_A:
  type: toska.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: true
  requirements:
    - virtual_binding: VDU-A
    - virtual_link: intVL

intVL:
  type: toska.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
    vl_profile:
      max_bitrate_requirements:
        root: 1000000
        leaf: 100000
      min_bitrate_requirements:
        root: 100000
        leaf: 10000

vduCp_B:
  type: toska.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: true
  requirements:
    - virtual_binding: VDU-B
    - virtual_link: intVL

VDU-B:
  type: toska.nodes.nfv.Vdu.Compute
  properties:
    name: VDU-B
    description: VDU-B description
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 1
  capabilities:
    virtual_compute:
      properties:
        virtual_memory:
          virtual_mem_size: 1 GB
        virtual_cpu:
          num_virtual_cpu: 1
```

The following snippet shows the relevant part of the file containing the imported node type definition:

exampleVNF.yaml

```
tosca_definitions_version: tosca_simple_yaml_1_3
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001
node_types:
  tosca.nodes.nfv.exampleVNF:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: ExampleVNF ]
        default: ExampleVNF
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      flavour_id:
        type: string
        constraints: [ valid_values: [ simple, complex ] ]
        default: simple
      flavour_description:
        type: string
        default: "" #empty string
      vnfm_info:
        type: list
        entry_schema:
          type: string
          constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
          default: [ '0:MyCompany-1.0.0' ]
    # other properties omitted for brevity
  requirements:
    - virtual_link:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
```

A.3.3 External connection point connected to an internal virtual link

In this case a VnfExtCp node template is needed. When substituting the VNFD low level service template for the abstract VNF node type, the virtual_link requirement of the abstract VNF node maps to the external_virtual_link requirement of the VnfExtCp node. The internal_virtual_link requirement of the VnfExtCp node is fulfilled with the corresponding capability of the internal VirtualLink node. This is shown in figure A.3.3-1.


```

requirements:
  # - external_virtual_link: # mapped to virtual_link requirement of VNF node
  - internal_virtual_link: intVL-A

```

```

intVL-A:
  type: tosca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
      flow_pattern: mesh
    description: Internal VL
  vl_profile:
    max_bitrate_requirements:
      root: 1000000
      leaf: 100000
    min_bitrate_requirements:
      root: 100000
      leaf: 10000
    qos:
      latency: 100 ms
      packet_delay_variation: 80 ms
      packet_loss_ratio: 0.00001

```

```

# Other node templates e.g. VDU-A, VDU-B, VduCP_A, etc.:

```

```

# ...

```

A.4 VNFD modelling design example by using TOSCA composition

The following example in figure A.4-1 shows a VNF descriptor contains three VDUs, which are interconnected by two virtualLinks.

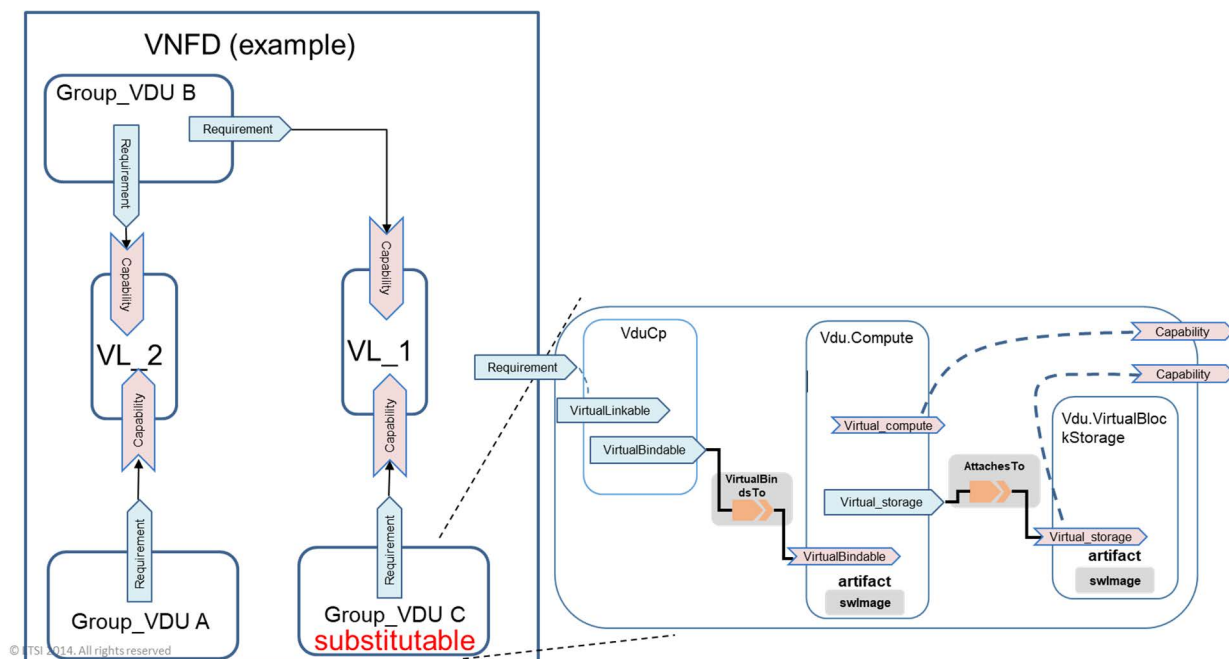


Figure A.4-1: Example of using substitution mapping for the VNFD design

In this example, a separate service template is used to describe the composition of Vdu.Compute, Vdu.VirtualBlockStorage and VduCp, and then substituted as a node template in a VNFD service template. `tosca.nodes.nfv.groupVDU_A`, `tosca.nodes.nfv.groupVDU_B`, `tosca.nodes.nfv.groupVDU_C` types are used for `substitution_mapping`.

```
tosca_definitions_version: toska_simple_yaml_1_3
description: Template of a VNFD example
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001
  - MyGroups.yaml #contains the node types definitions

topology_template:

  node_templates:
    Group_VDU_A:
      type: toska.nodes.nfv.groupVDU_A
      capabilities:
        virtual_compute:
          properties:
            virtual_memory:
              virtual_mem_size: 1 GB
            virtual_cpu:
              num_virtual_cpu: 1
      requirements:
        - virtual_link: VL_2

    Group_VDU_B:
      type: toska.nodes.nfv.groupVDU_B
      capabilities:
        virtual_compute:
          properties:
            virtual_memory:
              virtual_mem_size: 1 GB
            virtual_cpu:
              num_virtual_cpu: 1
      requirements:
        - virtual_link: VL_2
        - virtual_link1: VL_1

    Group_VDU_C:
      type: toska.nodes.nfv.groupVDU_C # the description of this type is described
                                      # in another service template.
      capabilities:
        virtual_compute:
          properties:
            virtual_memory:
              virtual_mem_size: 1 GB
            virtual_cpu:
              num_virtual_cpu: 1
      requirements:
        - virtual_link: VL_1

    VL_1:
      type: toska.nodes.nfv.VnfVirtualLink
      properties:
        connectivity_type:
          layer_protocols: [ ipv4 ]
        vl_profile:
          max_bitrate_requirements:
```



```

    type: toska.nodes.nfv.Vdu.Compute
    properties:
      name: vduC_compute
      description: vduC_compute ..
      vdu_profile:
        min_number_of_instances: 1
        max_number_of_instances: 1
        # other properties omitted here for brevity
    capabilities:
      virtual_compute:
        properties:
          virtual_memory:
            virtual_mem_size: 1 GB
          virtual_cpu:
            num_virtual_cpu: 1
#   artifacts:
#   sw_image:          # omitted here for brevity
    requirements:
      - virtual_storage: vduC_storage

    vduC_storage:
      type: toska.nodes.nfv.Vdu.VirtualBlockStorage
      properties:
        virtual_block_storage_data:
          size_of_storage: 2 GB
        # other properties omitted here for brevity

    internalCpd:
      type: toska.nodes.nfv.VduCp
      properties:
        layer_protocols: [ ipv4 ]
#       protocol: [ associated_layer_protocol: ipv4 ]
#       trunk_mode: false
#   properties omitted here for brevity
      requirements:
        # - virtual_link:
        - virtual_binding: vduC_compute

```

The following template fragment provides the group definitions used in the above examples.

MyGroups.yaml

```

tosca_definitions_version: toska_simple_yaml_1_3
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of VNFD types as defined in NFV SOL 001 GS

node_types:

  toska.nodes.nfv.groupVDU:
    description: Abstract group VDU.
    derived_from: toska.nodes.Root
    capabilities:
      virtual_compute:
        description: Describes virtual compute resources capabilities.
        type: toska.capabilities.nfv.VirtualCompute
      virtual_storage:
        description: Defines the capabilities of virtual_storage.
        type: toska.capabilities.nfv.VirtualStorage
    requirements:
      - virtual_link:

```

```

    capability: tosca.capabilities.nfv.VirtualLinkable
    relationship: tosca.relationships.nfv.VirtualLinksTo
    node: tosca.nodes.nfv.VnfVirtualLink
    occurrences: [ 1, 1 ]

tosca.nodes.nfv.groupVDU_A:
  description: Abstract group VDU A.
  derived_from: tosca.nodes.nfv.groupVDU

tosca.nodes.nfv.groupVDU_B:
  description: Abstract group VDU B.
  derived_from: tosca.nodes.nfv.groupVDU
  requirements:
    - virtual_link1:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
        node: tosca.nodes.nfv.VnfVirtualLink
        occurrences: [ 1, 1 ]

tosca.nodes.nfv.groupVDU_C:
  description: Abstract group VDU C.
  derived_from: tosca.nodes.nfv.groupVDU

```

A.5 VNFD with Single deployment flavour modelling design example

In this example, there is one deployment flavour applied to this VNFD, and TOSCA-Simple-Profile-YAML-v1.3 [20] is used for designing and processing this VNFD TOSCA model.

The one service template design illustrated by this example is only applicable when the VNF has only one deployment flavour.

The service template is the main entry point in the VNF Package, i.e. the Entry-definitions file, and is deployed as a stand-alone service template, i.e. without substituting for a node template. However, the service template still contains substitution_mappings to indicate its ability to substitute for a node template of the specific node type.

The service template contains a node template of type MyCompany.SunshineDB.1_0.1_0. The node template contains the properties defined in the node type definition and implementations for the LCM interfaces.

sunshinesimple.vnfd.tosca.yaml

This example illustrates a VNF with one VDU.Compute nodes (dbBackend) with two VDU connection points and one VNF virtual link (see figure A.5-1). The VNF is also connected to an external virtual link. The flavour_id is "simple".

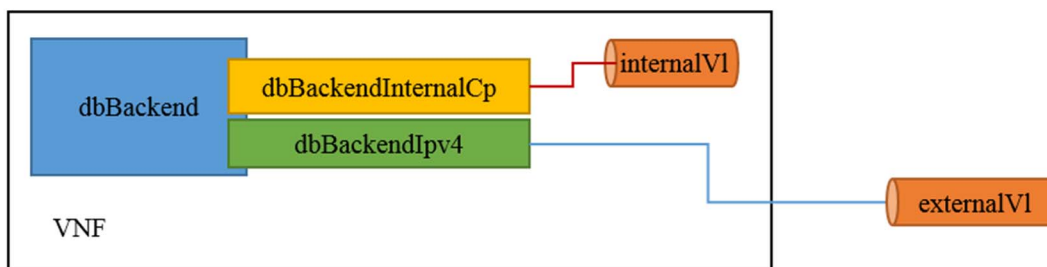


Figure A.5-1: SunshineDB (simple)

```

tosca_definitions_version: tosca_simple_yaml_1_3

description: Relational database, simple

imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001 for a VNFD

data_types:
  MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters:
    derived_from: tosca.datatypes.nfv.VnfOperationAdditionalParameters
    properties:
      segmentation_id_of_internalVl:
        type: string
        required: true
        default: 1-4095
      parameter_2:
        type: string
        required: true
        default: value_2

  MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcConfigurableProperties
    properties:
      additional_vnfc_configurable_properties:
        type: MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties
        required: true

  MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties
    properties:
      name_prefix_in_vim:
        type: string
        required: true
        default: "MyCustomer"
      dns_server:
        type: string
        required: true
        default: "90.200.250.57"

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: SunshineDB ]
        default: SunshineDB
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:

```

```

    type: string
    constraints: [ equal: '1.0' ]
    default: '1.0'
  flavour_id:
    type: string
    constraints: [ equal: simple ]
    default: simple
  flavour_description:
    type: string
    default: ""
  vnfm_info:
    type: list
    entry_schema:
      type: string
      constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
      default: [ '0:MyCompany-1.0.0' ]
  interfaces:
    Vnflcm:
      type: toska.interfaces.nfv.Vnflcm
      operations:
        instantiate:
          inputs:
            additional_parameters:
              type: MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters
              required: false
          #terminate:

MyCompany.nodes.nfv.Vdu.Aux:
  derived_from: toska.nodes.nfv.Vdu.Compute
  properties:
    configurable_properties:
      type: MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties
      required: false

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
  requirements:
    virtual_link: [ dbBackendIpv4, virtual_link ] # IPv4 for SQL

inputs:
  segmentation_id_of_internalVl:
    type: string
    required: true

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    properties:
      flavour_description: 'vnf simple flavour description'
    interfaces:
      Vnflcm:
        operations:
          instantiate:
            implementation: instantiate.workbook.mistral.yaml
          terminate:
            implementation: terminate.workbook.mistral.yaml

```

```
    heal:
      implementation: heal.workbook.mistral.yaml

dbBackend:
  type: MyCompany.nodes.nfv.Vdu.Aux
  properties:
    name: dbbackend
    description: dbBackend
    nfvi_constraints:
      key_1: value_1
      key_2: value_2
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 4
  capabilities:
    virtual_compute:
      properties:
        virtual_memory:
          virtual_mem_size: 8192 MiB
        virtual_cpu:
          cpu_architecture: x86
          num_virtual_cpu: 2
          virtual_cpu_clock: 1800 MHz
  requirements:
    - virtual_storage: mariaDbStorage

mariaDbStorage:
  type: tosca.nodes.nfv.Vdu.VirtualBlockStorage
  properties:
    virtual_block_storage_data:
      size_of_storage: '200 GB'
      rdma_enabled: true
  artifacts:
    sw_image:
      type: tosca.artifacts.nfv.SwImage
      file: maria.db.image.v1.0.qcow2
      properties:
        name: Software of Maria Db
        version: '1.0'
        checksum:
          algorithm: sha-256
          hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d
        container_format: bare
        disk_format: qcow2
        min_disk: 2 GB
        min_ram: 8192 MiB
        size: 2 GB
        operating_system: Linux
        supported_virtualisation_environments:
          - KVM

dbBackendInternalCp:
  type: tosca.nodes.nfv.VduCp
  properties:
    protocol: [associated_layer_protocol: ipv4 ]
    trunk_mode: false
    layer_protocols: [ ipv4 ]
    role: leaf
    description: Internal connection point on an VL
  requirements:
    - virtual_binding: dbBackend
```

```

- virtual_link: internalVl

internalVl:
  type: toska.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4, ethernet ]
      flow_pattern: mesh
    test_access: []
    description: Internal Virtual link in the VNF
    vl_profile:
      max_bitrate_requirements:
        root: 100000
        leaf: 20000
      min_bitrate_requirements:
        root: 10000
        leaf: 10000
      virtual_link_protocol_data:
        - associated_layer_protocol: ethernet
          l2_protocol_data:
            network_type: vlan
            segmentation_id: { get_input: segmentation_id_of_internalVl }

dbBackendIpv4:
  type: toska.nodes.nfv.VduCp
  properties:
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
    layer_protocols: [ ipv4 ]
    role: leaf
    description: External connection point to access the DB on IPv4
  requirements:
    #- virtual_link: # the target node is determined in the NSD
    - virtual_binding: dbBackend

```

A.6 Scaling and Instantiation Level examples

A.6.1 ScalingAspect and InstantiationLevels policies with uniform delta

This example shows instantiationLevels, and ScalingAspect policy types in the complex scaling scenario where scaling aspect delta is based on "uniform delta" values, where it has only one entry. For a uniform aspect, the example 1 shows that step_deltas is omitted, whereas the example 2 shows step_deltas is included as a list of entries.

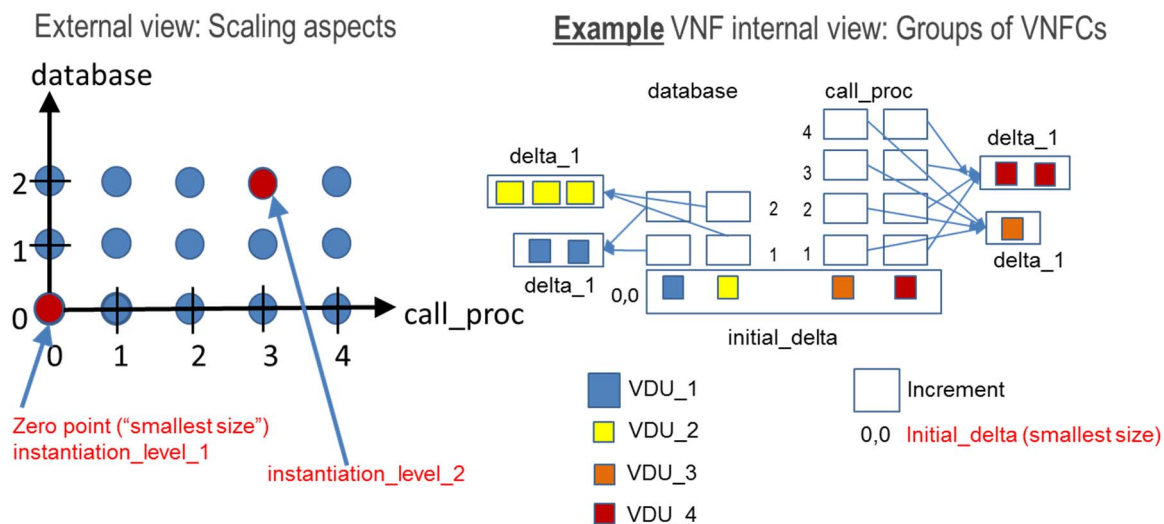


Figure A.6.1-1: Complex scaling example with uniform delta

EXAMPLE 1:

```
tosca_definitions_version: tosca_simple_yaml_1_3
description: Complex scaling example (uniform delta value) described with policies
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
GS NFV-SOL 001
```

```
topology_template:
  node_templates:
    vdu_1:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: ..
        description: ..
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 5
        nfvi_constraints:
          key_1: value_1
          key_2: value_2
      capabilities:
        virtual_compute:
          properties:
            virtual_memory:
              virtual_mem_size: 1 GB
            virtual_cpu:
              num_virtual_cpu: 1

    vdu_2:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: ..
        description: ..
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 7
        nfvi_constraints:
          key_1: value_1
          key_2: value_2
```

```
capabilities:
  virtual_compute:
    properties:
      virtual_memory:
        virtual_mem_size: 1 GB
      virtual_cpu:
        num_virtual_cpu: 1

vdu_3:
  type: tosa.nodes.nfv.Vdu.Compute
  properties:
    name: ..
    description: ..
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 5
    nfvi_constraints:
      key_1: value_1
      key_2: value_2
  capabilities:
    virtual_compute:
      properties:
        virtual_memory:
          virtual_mem_size: 1 GB
        virtual_cpu:
          num_virtual_cpu: 1

vdu_4:
  type: tosa.nodes.nfv.Vdu.Compute
  properties:
    name: ..
    description: ..
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 9
    nfvi_constraints:
      key_1: value_1
      key_2: value_2
  capabilities:
    virtual_compute:
      properties:
        virtual_memory:
          virtual_mem_size: 1 GB
        virtual_cpu:
          num_virtual_cpu: 1

vl_1:
  type: tosa.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
    vl_profile:
      min_bitrate_requirements:
        root: 1000000
      max_bitrate_requirements:
        root: 2000000

vl_2:
  type: tosa.nodes.nfv.VnfVirtualLink
```

```
properties:
  connectivity_type:
    layer_protocols: [ ipv4 ]
  vl_profile:
    min_bitrate_requirements:
      root: 1000000
    max_bitrate_requirements:
      root: 4000000

policies:
- scaling_aspects:
  type: tosca.policies.nfv.ScalingAspects
  properties:
    aspects:
      database:
        name: ..
        description: ..
        max_scale_level: 2

      call_proc:
        name: ..
        description: ..
        max_scale_level: 4

- vdu_1_initial_delta:
  type: tosca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
  targets: [ vdu_1 ]

- vdu_1_scaling_aspect_deltas:
  type: tosca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: database
    deltas:
      delta_1:
        number_of_instances: 2
  targets: [ vdu_1 ]

- vdu_2_initial_delta:
  type: tosca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
  targets: [ vdu_2 ]

- vdu_2_scaling_aspect_deltas:
  type: tosca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: database
    deltas:
      delta_1:
        number_of_instances: 3
  targets: [ vdu_2 ]

- vl_1_bitrate_initial_delta:
  type: tosca.policies.nfv.VirtualLinkBitrateInitialDelta
```

```
properties:
  initial_delta:
    bitrate_requirements:
      root: 1000000
  targets: [ vl_1 ]

- vl_1_bitrate_scaling_aspect_deltas:
  type: tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas
  properties:
    aspect: database
    deltas:
      delta_1:
        bitrate_requirements:
          root: 1000000
  targets: [ vl_1 ]

- vdu_3_initial_delta:
  type: tosca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
  targets: [ vdu_3 ]

- vdu_3_scaling_aspect_deltas:
  type: tosca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: call_proc
    deltas:
      delta_1:
        number_of_instances: 1
  targets: [ vdu_3 ]

- vdu_4_initial_delta:
  type: tosca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
  targets: [ vdu_4 ]

- vdu_4_scaling_aspect_deltas:
  type: tosca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: call_proc
    deltas:
      delta_1:
        number_of_instances: 2
  targets: [ vdu_4 ]

- vl_2_bitrate_initial_delta:
  type: tosca.policies.nfv.VirtualLinkBitrateInitialDelta
  properties:
    initial_delta:
      bitrate_requirements:
        root: 1000000
  targets: [ vl_2 ]

- vl_2_bitrate_scaling_aspect_deltas:
  type: tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas
  properties:
    aspect: call_proc
```

```
    deltas:
      delta_1:
        bitrate_requirements:
          root: 1000000
    targets: [ vl_2 ]

- instantiation_levels:
  type: tosca.policies.nfv.InstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        description: ..
        scale_info:
          database:
            scale_level: 0
          call_proc:
            scale_level: 0
      instantiation_level_2:
        description: ..
        scale_info:
          database:
            scale_level: 2
          call_proc:
            scale_level: 3
      default_level: instantiation_level_1

- vdu_1_instantiation_levels:
  type: tosca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
      instantiation_level_2:
        number_of_instances: 5
    targets: [ vdu_1 ]

- vdu_2_instantiation_levels:
  type: tosca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
      instantiation_level_2:
        number_of_instances: 7
    targets: [ vdu_2 ]

- vl_1_instantiation_levels:
  type: tosca.policies.nfv.VirtualLinkInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        bitrate_requirements:
          root: 30000
      instantiation_level_2:
        bitrate_requirements:
          root: 30000
    targets: [ vl_1 ]

- vdu_3_instantiation_levels:
  type: tosca.policies.nfv.VduInstantiationLevels
  properties:
```

```

    levels:
      instantiation_level_1:
        number_of_instances: 1
      instantiation_level_2:
        number_of_instances: 4
    targets: [ vdu_3 ]

- vdu_4_instantiation_levels:
  type: toasca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
      instantiation_level_2:
        number_of_instances: 7
    targets: [ vdu_4 ]

- vl_2_instantiation_levels:
  type: toasca.policies.nfv.VirtualLinkInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        bitrate_requirements:
          root: 30000
      instantiation_level_2:
        bitrate_requirements:
          root: 60000
    targets: [ vl_2 ]

```

EXAMPLE 2: This example is same as the example 1, except for ScalingAspects, which shows step_deltas list of entries.

```

topology_template:
. . .
  policies:
    - scaling_aspects:
      type: toasca.policies.nfv.ScalingAspects
      properties:
        aspects:
          database:
            name: ..
            description: ..
            max_scale_level: 2
            step_deltas:
              - delta_1
              - delta_1
          call_proc:
            name: ..
            description: ..
            max_scale_level: 4
            step_deltas:
              - delta_1
              - delta_1
              - delta_1
              - delta_1
. . .

```

A.6.2 ScalingAspect and InstantiationLevels policies with non-uniform deltas

This example shows instantiationLevel, ScalingAspect policies and group types in the complex scaling scenario where scaling aspect delta is based on "delta" (non-uniform) values for Processing Auxiliary VNFC.

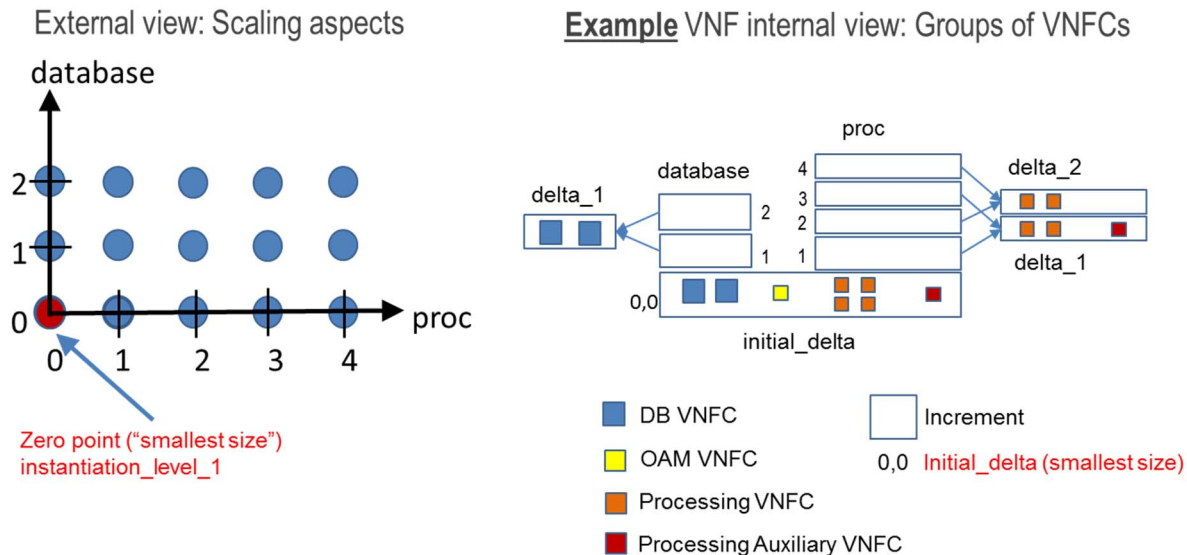


Figure A.6.2-1: Complex scaling example with non-uniform delta

For simplicity, virtual link is not illustrated and instantiation level is one level (default) in this example, as these properties are already illustrated in clause A.6.1.

The "Proc" aspect contains a uniform delta part ("processing" and a non-uniform part ("processing_auxiliary").

The "DB" aspect contains a non-uniform part ("db"), and no delta information for "oam".

```
tosca_definitions_version: tosca_simple_yaml_1_3
description: Complex example (uniform and non-uniform delta value) described with policies
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001

topology_template:
  node_templates:
    db:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: ..
        description: ..

      vdu_profile:
        min_number_of_instances: 2
        max_number_of_instances: 6
      nfvi_constraints:
        key_1: value_1
        key_2: value_2
      capabilities:
        virtual_compute:
          properties:
            virtual_memory:
              virtual_mem_size: 1 GB
```

```
        virtual_cpu:
          num_virtual_cpu: 1

oam:
  type: toska.nodes.nfv.Vdu.Compute
  properties:
    name: ..
    description: ..
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 1
    nfvi_constraints:
      key_1: value_1
      key_2: value_2
  capabilities:
    virtual_compute:
      properties:
        virtual_memory:
          virtual_mem_size: 1 GB
        virtual_cpu:
          num_virtual_cpu: 1

processing:
  type: toska.nodes.nfv.Vdu.Compute
  properties:
    name: ..
    description: ..

    vdu_profile:
      min_number_of_instances: 4
      max_number_of_instances: 12
    nfvi_constraints:
      key_1: value_1
      key_2: value_2
  capabilities:
    virtual_compute:
      properties:
        virtual_memory:
          virtual_mem_size: 1 GB
        virtual_cpu:
          num_virtual_cpu: 1

processing_auxiliary:
  type: toska.nodes.nfv.Vdu.Compute
  properties:
    name: ..
    description: ..

    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 3
    nfvi_constraints:
      key_1: value_1
      key_2: value_2
  capabilities:
    virtual_compute:
      properties:
        virtual_memory:
          virtual_mem_size: 1 GB
```



```
    virtual_cpu:
      num_virtual_cpu: 1

policies:
- scaling_aspects:
  type: toska.policies.nfv.ScalingAspects
  properties:
    aspects:
      database:
        name: ..
        description: ..
        max_scale_level: 2
        step_deltas:
          - delta_1
          - delta_1
      proc:
        name: ..
        description: ..
        max_scale_level: 4
        step_deltas:
          - delta_1
          - delta_2
          - delta_1
          - delta_2

- db_initial_delta:
  type: toska.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 2
  targets: [ db ]

- db_scaling_aspect_deltas:
  type: toska.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: database
    deltas:
      delta_1:
        number_of_instances: 2
  targets: [ db ]

- oam_initial_delta:
  type: toska.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
  targets: [ oam ]

- processing_initial_delta:
  type: toska.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 4
  targets: [ processing ]

- processing_scaling_aspect_deltas:
  type: toska.policies.nfv.VduScalingAspectDeltas
```

```
properties:
  aspect: proc
  deltas:
    delta_1:
      number_of_instances: 2
    delta_2:
      number_of_instances: 2
  targets: [ processing ]

- processing_auxiliary_initial_delta:
  type: toasca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
  targets: [ processing_auxiliary ]

- processing_auxiliary_scaling_aspect_deltas:
  type: toasca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: proc
    deltas:
      delta_1:
        number_of_instances: 1
      delta_2:
        number_of_instances: 0
  targets: [ processing_auxiliary ]

- instantiation_levels:
  type: toasca.policies.nfv.InstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        description: ..
        scale_info:
          database:
            scale_level: 0
          proc:
            scale_level: 0

- db_instantiation_levels:
  type: toasca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 2
  targets: [ db ]

- oam_instantiation_levels:
  type: toasca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
  targets: [ oam ]

- processing_instantiation_levels:
  type: toasca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 4
```

```

targets: [ processing ]

- processing_auxiliary_instantiation_levels:
  type: toska.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
    targets: [ processing_auxiliary ]

```

A.7 Service Access Point

A.7.1 General

A SAP may either be a VNF external connection point (or a PNF external connection points or a Sap of a nested NS of this NS) which is exposed as CP of one of the NS constituents (VnfExtCp, PnfExtCp, nested NS Sap externally, or be exposed as a new CP on NS virtual link. This is shown in figure A.7.1-1. In the latter case the Sap node has an association relationship of type VirtualLinksTo to the NsVirtualLink node.

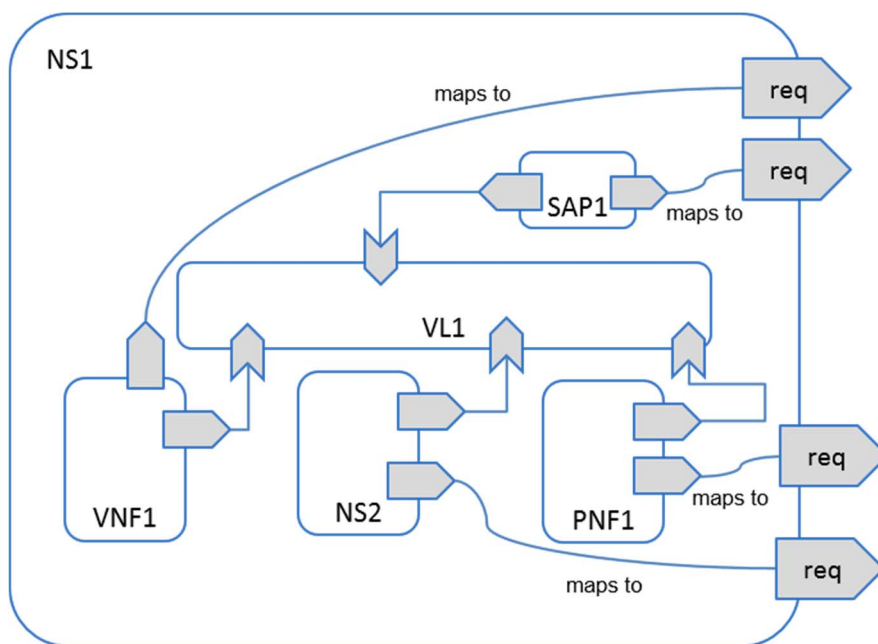


Figure A.7.1-1: Overview of SAP options

The following clauses illustrate the use of both models of SAP, re-exposure of a VNF external connection point and connected to an NS virtual link.

A.7.2 VNF External connection point exposing as a SAP

In this case, Sap node template is not required. When substituting the NSD service template for the abstract NS node, the virtual_link requirement of the abstract NS node is mapped to the VNF external connection point's virtual_link requirement. This is shown in figure A.7.2-1. In this case the NS Service Template does not include a node template of type toska.nodes.nfv.Sap, but the functionality of it is provided by a VNF external connection point.

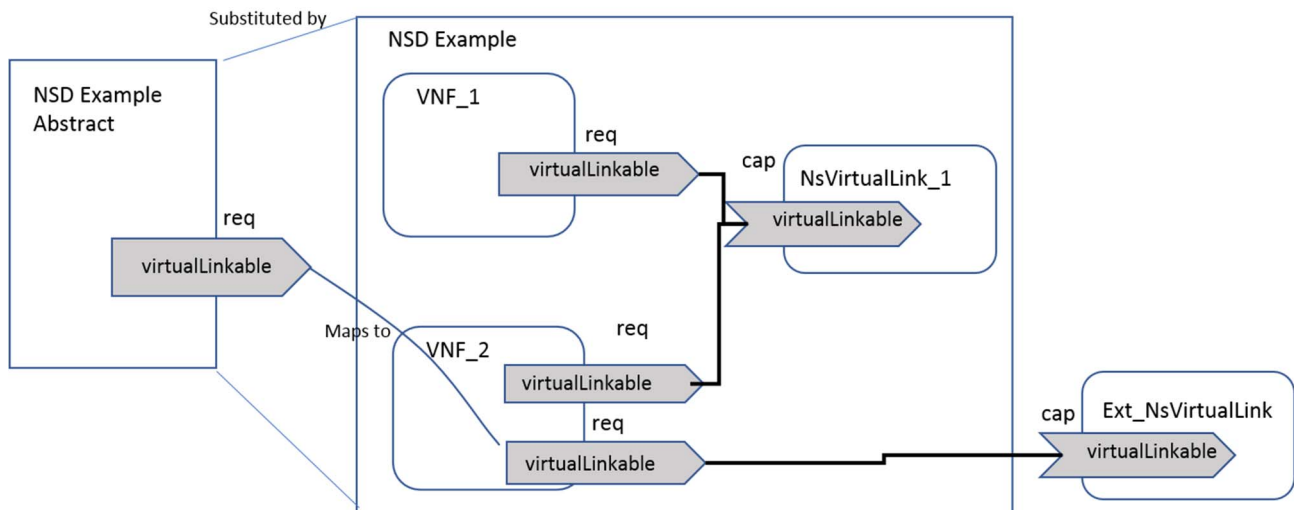


Figure A.7.2-1: VNF_2 External connection point (VnfExtCp) exposed as a Sap

The following snippet shows the relevant part of the NS service template:

```
tosca_definitions_version: tosca_simple_yaml_1_3
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in ETSI
GS NFV-SOL 001
  - example_VNF_2.yaml

node_types:
  tosca.nodes.nfv.exampleNS:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ valid_values: [ b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ] ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer:
        type: string
        constraints: [ valid_values: [ MyCompany ] ]
        default: MyCompany
      name:
        type: string
        constraints: [ valid_values: [ ExampleService ] ]
        default: ExampleService
      version:
        type: string
        constraints: [ valid_values: [ '1.0' ] ]
        default: '1.0'
      invariant_id:
        type: string
        constraints: [ valid_values: [ 1111-2222-aaaa-bbbb ] ]
        default: 1111-2222-aaaa-bbbb
      flavour_id:
        type: string
        constraints: [ valid_values: [ small, big ] ]
        default: small
```

```

    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo

#...

topology_template:

#...

substitution_mappings:
  node_type: tosca.nodes.nfv.exampleNS
  requirements:
    virtual_link: [VNF_2, virtual_link_2] # the External connection point of VNF_2
                                           # will be used as the Sap of this NS

node_templates:
  VNF_2:
    type: tosca.nodes.nfv.example_VNF2
    # properties omitted for brevity
    requirements:
      - virtual_link_1: NsVirtualLink_1 # connects to the External connection
                                         # point which maps to the
                                         # virtual_link_1 requirement of VNF_2
      - virtual_link_2: # mapped to virtual_link requirement of NS node

  NsVirtualLink_1:
    type: tosca.nodes.nfv.NsVirtualLink
    properties:
      vl_profile:
        min_bitrate_requirements:
          root: 100000
        max_bitrate_requirements:
          root: 200000
      connectivity_type:
        layer_protocols: [ ipv4 ]

# other node template definitions
#...

```

A.7.3 SAP connected to an NS virtual link

In this case, a Sap node template is required. When substituting the NSD service template for the abstract NS node type, the `virtual_link` requirement of the abstract NS node maps to the `virtual_link` requirement of the Sap node. This is shown in figure A.7.3-1.

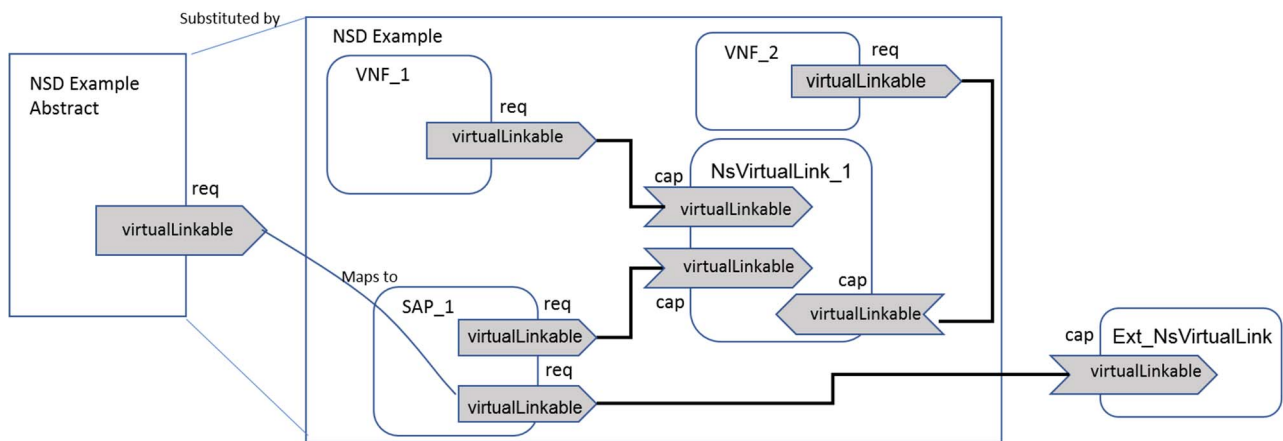


Figure A.7.3-1: SAP_1 exposed as new CP on an internal virtual link

The following snippet shows the corresponding node template definition:

```
tosca_definitions_version: tosca_simple_yaml_1_3
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in ETSI
    GS NFV-SOL 001
node_types:
  tosca.nodes.nfv.exampleNS:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ valid_values: [ b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ] ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer:
        type: string
        constraints: [ valid_values: [ MyCompany ] ]
        default: MyCompany
    name:
      type: string
      constraints: [ valid_values: [ ExampleService ] ]
      default: ExampleService
    version:
      type: string
      constraints: [ valid_values: [ '1.0' ] ]
      default: '1.0'
    invariant_id:
      type: string
      constraints: [ valid_values: [ 1111-2222-aaaa-bbbb ] ]
      default: 1111-2222-aaaa-bbbb
    flavour_id:
      type: string
      constraints: [ valid_values: [ small, big ] ]
      default: small
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
topology_template:
  #...
  substitution_mappings:
```

```

node_type: toska.nodes.nfv.exampleNS
requirements:
  virtual_link: [SAP_1, external_virtual_link] #SAP_1 is the Sap of the NS

node_templates:
  SAP_1:
    type: toska.nodes.nfv.Sap
    properties:
      protocol: [ associated_layer_protocol: ipv4 ]
      trunk_mode: false
      layer_protocols: [ ipv4 ]
    # other properties omitted for brevity
    requirements:
      - internal_virtual_link: NsVirtualLink_1
      # - external_virtual_link: # map to virtual_link requirement of the NS node

  NsVirtualLink_1:
    type: toska.nodes.nfv.NsVirtualLink
    properties:
      vl_profile:
        min_bitrate_requirements:
          root: 100000
        max_bitrate_requirements:
          root: 200000
      connectivity_type:
        layer_protocols: [ ipv4 ]

# Other node templates
#...
```

A.8 NSD with Single deployment flavour modelling design example

In this example, there is one deployment flavour applied to this NSD, and TOSCA-Simple-Profile-YAML-v1.3 [20] is used for designing and processing this NSD TOSCA model.

The one service template design illustrated by this example is only applicable when the NS has only one deployment flavour.

The service template is the main entry point in the NSD file structure, i.e. the Entry-definitions file, and is deployed as a stand-alone service template, i.e. without substituting for a node template. However, the service template still contains substitution_mappings to indicate its ability to substitute for a node template of the specific node type.

example_NS.yaml

This example illustrates an NS which contains two VNF: VNF_1 and VNF_2, they connect through NsVirtualLink_1. One of the VnfExtCp of VNF_2 is exposed as the Sap of this NS. The flavour_id is "simple". In the VNF_2 node template, it defines a dependency requirement to VNF_1, which indicates that all the instances of VNF_1 have to be deployed first before deploy the instances of VNF_2. NsMonitoring and VnfMonitoring policies represent information to be monitored during the lifetime of a network service and VNF instances.

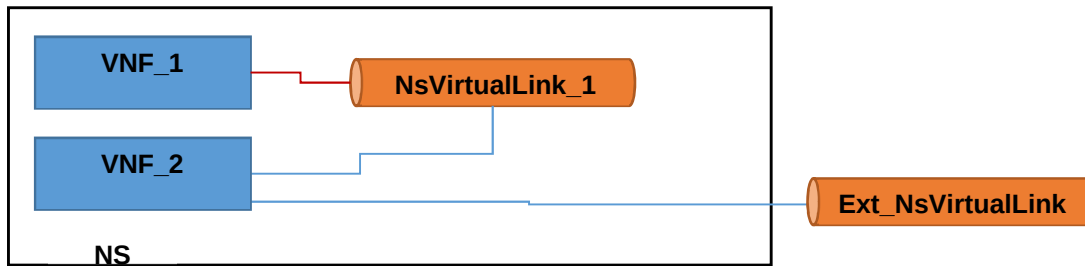


Figure A.8-1: example_NS (simple)

```

tosca_definitions_version: tosca_simple_yaml_1_3

description: Relational database, simple

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of NSD related TOSCA types as defined in ETSI
    GS NFV-SOL 001
  - example_VNF_1.yaml # uri of the yaml file which contains the definition of
    tosca.nodes.nfv.example_VNF1, this file might be included in the NSD file structure
  - example_VNF_2.yaml # uri of the yaml file which contains the definition of
    tosca.nodes.nfv.example_VNF2, this file might be included in the NSD file structure

data_types:
  MyCompany.datatypes.nfv.NsInstantiateAdditionalParameters:
    derived_from: tosca.datatypes.nfv.NsOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
        required: true
        default: value_1
      parameter_2:
        type: string
        required: true
        default: value_2

node_types:
  tosca.example_NS:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
    name:
      type: string
      constraints: [ equal: ExampleService ]
      default: ExampleService
    version:
      type: string
      constraints: [ equal: '1.0' ]
      default: '1.0'
    invariant_id:
      type: string

```



```

    constraints: [ equal: 1111-2222-aaaa-bbbb ]
    default: 1111-2222-aaaa-bbbb
  flavour_id:
    type: string
    constraints: [ equal: simple ]
    default: simple
  interfaces:
    Nslcm:
      type: toska.interfaces.nfv.Nslcm
      operations:
        instantiate:
          inputs:
            additional_parameters:
              type: MyCompany.datatypes.nfv.NsInstantiateAdditionalParameters
              required: false

topology_template:
  substitution_mappings:
    node_type: toska.example_NS
  requirements:
    virtual_link: [ VNF_2, virtual_link_2 ] # the External connection point of
                                           # VNF_2 is exposed as the Sap

node_templates:
  my_service:
    type: toska.example_NS
    properties:
    interfaces:
      Nslcm:
        operations:
          instantiate:
            implementation: instantiate.workflow.yaml
          terminate:
            implementation: terminate.workflow.yaml

  VNF_1:
    type: toska.nodes.nfv.example_VNF1
    properties:
      # no property assignments needed for required properties that have a default
value assigned in the node type definition, e.g. descriptor_id
    vnf_profile:
      instantiation_level: level_1
      min_number_of_instances: 2
      max_number_of_instances: 6
    requirements:
      - virtual_link: NsVirtualLink_1

  VNF_2:
    type: toska.nodes.nfv.example_VNF2
    properties:
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 3
    requirements:
      - virtual_link_1: NsVirtualLink_1
      # - virtual_link_2: # map to virtual_link requirement of the NS node
      - dependency: VNF_1

```

```

NsVirtualLink_1:
  type: toska.nodes.nfv.NsVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ipv4, ethernet]
      flow_pattern: mesh
    vl_profile:
      max_bitrate_requirements:
        root: 1000
      min_bitrate_requirements:
        root: 1000
      virtual_link_protocol_data:
        - associated_layer_protocol: ethernet
          l2_protocol_data:
            name: nsBaseNetwork
            network_type: vlan
            segmentation_id: VLAN100
        - associated_layer_protocol: ipv4
          l3_protocol_data:
            name: mybaseSubnet
            ip_version: ipv4
            cidr: 192.168.0.0/24

policies:
- my_service_NsMonitoring:
  type: toska.policies.nfv.NsMonitoring
  properties:
    ns_monitoring_parameters:
      ns_monitoring_1111: #key map to the id of this monitoring parameter
        name: MyService_byte_incoming_sap
        performance_metric: byte_incoming_sap
        collection_period: 1 s
      ns_monitoring_2222: #key map to the id of this monitoring parameter
        name: MyService_byte_outgoing_sap
        performance_metric: byte_outgoing_sap
        collection_period: 1 s
    targets: [ my_service ]

- VNF_1_VnfMonitoring:
  type: toska.policies.nfv.VnfMonitoring
  properties:
    vnf_monitoring_parameters:
      vnf_1_monitoring_1111: #key map to the id of this monitoring parameter
        name: VNF_1_v_cpu_usage_mean_vnf
        performance_metric: v_cpu_usage_mean_vnf
        collection_period: 1 s
      vnf_1_monitoring_2222: #key map to the id of this monitoring parameter
        name: VNF_1_v_disk_usage_mean_vnf
        performance_metric: v_disk_usage_mean_vnf
        collection_period: 1 s
    targets: [ VNF_1 ]

- VNF_2_VnfMonitoring:
  type: toska.policies.nfv.VnfMonitoring
  properties:
    vnf_monitoring_parameters:
      vnf_2_monitoring_1111: #key map to the id of this monitoring parameter
        name: VNF_2_v_cpu_usage_mean_vnf
        performance_metric: v_cpu_usage_mean_vnf
        collection_period: 1 s
      vnf_2_monitoring_2222: #key map to the id of this monitoring parameter

```

```

name: VNF_2_v_disk_usage_mean_vnf
performance_metric: v_disk_usage_mean_vnf
collection_period: 1 s
vnf_2_monitoring_3333: #key map to the id of this monitoring parameter
name: VNF_2_v_memory_usage_mean_vnf
performance_metric: v_memory_usage_mean_vnf
collection_period: 1 s
targets: [ VNF_2 ]

```

The above example illustrates three policies (1 NS-level policy and 2 VNF-level policies). Each of the policies has different monitoring parameters. In the case, where the values of `vnf_monitoring_parameters` are the same across all constituent VNFs, a single `VnfMonitoring` policy can be used with a `targets` referencing all VNF node types.

A.9 Mapping between NFV IM and TOSCA concepts

A.9.1 Introduction

This clause describes the mapping between the NFV information model and the TOSCA concepts.

A.9.2 Mapping between ETSI GS NFV-IFA 011 IM and TOSCA concepts

Table A.9.2-1 illustrates the mapping between the information model as specified in ETSI GS NFV-IFA 011 [1] and the corresponding TOSCA concepts.

Table A.9.2-1: Mapping between ETSI GS NFV-IFA 011 [1] IM and TOSCA concepts

ETSI GS NFV-IFA 011 [1] information element	TOSCA concept
VNFD (clause 7.1.2)	TOSCA service template(s) in the VNF package
VnfExtCpd (clause 7.1.4)	node template with type <code>tosca.nodes.nfv.VnfExtCp</code> or <code>tosca.nodes.nfv.VduCp</code> or <code>tosca.nodes.nfv.VipCp</code>
VnfLcmOperationsConfiguration (clause 7.1.5)	property of VNF node type with data type <code>tosca.datatypes.nfv.VnfLcmOperationsConfiguration</code> and/or inputs <code>additional_parameters</code> of the corresponding operation in the <code>Vnflcm</code> interface with type <code>tosca.datatypes.nfv.VnfOperationAdditionalParameters</code>
Vdu (clause 7.1.6)	collection of <code>tosca.nodes.nfv.VduCp</code> , <code>tosca.nodes.nfv.Vdu.Compute</code> (for VM based VNFs), <code>tosca.nodes.nfv.Vdu.OsContainerDeployableUnit</code> (for Oscontainer based VNFs), <code>tosca.nodes.nfv.Vdu.OsContainer</code> (for Oscontainer based VNFs), <code>tosca.nodes.nfv.Vdu.VirtualBlockStorage</code> , <code>tosca.nodes.nfv.Vdu.VirtualObjectStorage</code> and <code>tosca.nodes.nfv.Vdu.VirtualFileStorage</code> types
Cpd (clause 7.1.6.3)	node type <code>tosca.nodes.nfv.Cp</code>
VduCpd (clause 7.1.6.4)	node template with type <code>tosca.nodes.nfv.VduCp</code>
SwImageDesc (clause 7.1.6.5)	Artifact with type <code>tosca.artifacts.nfv.SwImage</code>
SecurityGroupRule (clause 7.1.6.9)	policy with type <code>tosca.policies.nfv.SecurityGroupRule</code>

ETSI GS NFV-IFA 011 [1] information element	TOSCA concept
TrunkPortTopology (clause 7.1.6.11)	trunk_binding requirement between node with type <code>tosca.nodes.nfv.VduCp</code> and node with type <code>tosca.nodes.nfv.VduSubCp</code>
Subport (clause 7.1.6.12)	node template with type <code>tosca.nodes.nfv.VduSubCp</code>
VLd (clause 7.1.7)	node template with type <code>tosca.nodes.nfv.VnfVirtualLink</code>
DeploymentFlavour (clause 7.1.8)	lower level service template(s) in the VNF package
VduProfile (clause 7.1.8.3)	property of <code>Vdu.Compute</code> node type with data type <code>tosca.datatypes.nfv.VduProfile</code>
VirtualLinkProfile (clause 7.1.8.4)	property of <code>VnfVirtualLink</code> node type with data type <code>tosca.datatypes.nfv.VIPProfile</code>
VirtualLinkDescFlavour (clause 7.1.8.5)	Only qos attribute in <code>VirtualLinkDescFlavour</code> has been defined as property of the <code>VIPProfile</code> data type
InstantiationLevel (clause 7.1.8.7)	policy with type <code>tosca.policies.nfv.InstantiationLevels</code>
VduLevel (clause 7.1.8.9)	policy with type <code>tosca.policies.nfv.VduInstantiationLevels</code>
LocalAffinityOrAntiAffinityRule (clause 7.1.8.11)	policy with type <code>tosca.policies.nfv.AffinityRule</code> or <code>tosca.policies.nfv.AntiAffinityRule</code>
AffinityOrAntiAffinityGroup (clause 7.1.8.12)	policy with type <code>tosca.policies.nfv.AffinityRule</code> or <code>tosca.policies.nfv.AntiAffinityRule</code>
Dependencies (clause 7.8.1.19)	dependency requirements between different node templates of type <code>Vdu.Compute</code> (or a derived node type)
VirtualComputeDesc (clause 7.1.9.2.2)	VirtualCompute capability of the <code>Vdu.Compute</code> node template
VirtualStorageDesc with BlockStorageData (clause 7.1.9.4.3)	node template with type <code>tosca.nodes.nfv.Vdu.VirtualBlockStorage</code>
VirtualStorageDesc with ObjectStorageData (clause 7.1.9.4.4)	node template with type <code>tosca.nodes.nfv.Vdu.VirtualObjectStorage</code>
VirtualStorageDesc with FileStorageData (clause 7.1.9.4.5)	node template with type <code>tosca.nodes.nfv.Vdu.VirtualFileStorage</code>
ScalingAspect (clause 7.1.10.2)	policy with type <code>tosca.policies.nfv.ScalingAspects</code>
ScalingDelta (clause 7.1.10.4)	policy with type <code>tosca.policies.nfv.VduScalingAspectDeltas</code> and/or <code>tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas</code>
VnfIndicator (clause 7.1.11.2)	interface with type <code>tosca.interfaces.nfv.VnfIndicator</code> and related attributes of the VNF node type.
MonitoringParameter (clause 7.1.11.3)	property of the VNF, <code>Vdu.Compute</code> and <code>VnfVirtualLink</code> node types
VnfConfigurableProperties (clause 7.1.12)	property of VNF node template with data type <code>tosca.datatypes.nfv.VnfConfigurableProperties</code>
LifeCycleManagementScript (clause 7.1.13)	interface with type <code>tosca.interfaces.nfv.Vnflcm</code> or <code>tosca.interfaces.nfv.ChangeCurrentVnfPackage</code>
VnfInfoModifiableAttributes (clause 7.1.14)	property of VNF node template with data type <code>tosca.datatypes.nfv.VnfInfoModifiableAttributes</code>
VnfPackageChangeInfo (clause 7.1.15)	policy with type <code>tosca.policies.nfv.VnfPackageChange</code> , and <code>tosca.interfaces.nfv.ChangeCurrentVnfPackage</code> or <code>tosca.interfaces.nfv.Vnflcm</code>
VnflcmOperationCoordination and LcmCoordinationActionMapping (clause 7.1.16)	policy with type <code>tosca.policies.nfv.LcmCoordinationAction</code> and <code>tosca.policies.nfv.LcmCoordinationsForLcmOperation</code>

ETSI GS NFV-IFA 011 [1] information element	TOSCA concept
VipCpd (clause 7.1.17)	node template with type <code>tosca.nodes.nfv.VipCp</code>
VirtualCpd (clause 7.1.18)	node template with type <code>tosca.nodes.nfv.VirtualCp</code>
CertificateDesc (clause 7.1.19.2)	<code>tosca.nodes.nfv.Certificate</code>
CSRRequirements (clause 7.1.19.3)	<code>tosca.datatypes.nfv.CSRRequirements</code>
CertificateBaseProfile (clause 7.1.19.4)	<code>tosca.datatypes.nfv.CertificateBaseProfile</code>
CertSubjectData (clause 7.1.19.5)	<code>tosca.datatypes.nfv.CertSubjectData</code>
PaasServiceRequest (clause 7.1.21.2)	note template with type <code>tosca.nodes.nfv.PaaSServiceRequest</code>

A.9.3 Mapping between ETSI GS NFV-IFA 014 IM and TOSCA concepts

Table A.9.3-1 illustrates the mapping between the information model as specified in ETSI GS NFV-IFA 014 [2] and the corresponding TOSCA concepts.

Table A.9.3-1: Mapping between ETSI GS NFV-IFA 014 [2] IM and TOSCA concepts

ETSI GS NFV-IFA 014 [2] information element	TOSCA concept
NSD (clause 6.2.2)	TOSCA service template(s) in the NSD file structure
Sapd (clause 6.2.3)	node template with type <code>tosca.nodes.nfv.Sap</code> or the <code>virtual_link</code> requirement in the <code>substitution_mapping</code>
MonitoredData (clause 6.2.6)	policy with type <code>tosca.policies.nfv.VnfMonitoring</code> and <code>tosca.policies.nfv.NsMonitoring</code>
VnfIndicatorData (clause 6.2.7)	interface with type <code>tosca.interfaces.nfv.NsVnfIndicator</code> and related attributes of the NS node type
LifeCycleManagementScript (clause 6.2.9)	Interface with type <code>Nslcm</code>
NsDf (clause 6.3)	lower level service template(s) in the NSD file structure
VnfProfile (clause 6.3.3)	property of VNF node template with data type <code>tosca.datatypes.nfv.VnfProfile</code>
VirtualLinkProfile (clause 6.3.4)	property of <code>NsVirtualLink</code> node template with data type <code>tosca.datatypes.nfv.VLProfile</code>
AffinityOrAntiAffinityGroup (clause 6.3.5)	policy with type <code>tosca.policies.nfv.NsAffinityRule</code> or <code>tosca.policies.nfv.NsAntiAffinityRule</code>
NsVirtualLinkConnectivity (clause 6.3.7)	requirement and capability between VNF or Forwarding or Sap and <code>NsVirtualLink</code> node templates
LocalAffinityOrAntiAffinityRule (clause 6.3.8)	policy with type <code>tosca.policies.nfv.NsAffinityRule</code> or <code>tosca.policies.nfv.NsAntiAffinityRule</code>
NsLevel (clause 6.3.9)	policy with type <code>tosca.policies.nfv.NsInstantiationLevels</code>
NsProfile (clause 6.3.11)	property of NS node template with data type <code>tosca.datatypes.nfv.NsProfile</code>
Dependencies (clause 6.3.12)	dependency requirements
Vnffgd (clause 6.4.2)	group with type <code>tosca.groups.nfv.VNFFG</code>
Nfpgd (clause 6.4.3)	node template with type <code>tosca.nodes.nfv.NFP</code>
NfpPositionElement (clause 6.4.6)	node template with type <code>tosca.nodes.nfv.NfpPositionElement</code>
NfpPositionDesc (clause 6.4.5)	node template with type <code>tosca.nodes.nfv.NfpPosition</code>
CpdInConstituentElement (clause 6.4.8)	node template with type <code>tosca.nodes.nfv.Forwarding</code> or <code>tosca.nodes.nfv.Sap</code>

ETSI GS NFV-IFA 014 [2] information element	TOSCA concept
NsVirtualLinkDesc (clause 6.5.2)	node template with type tosca.nodes.nfv.NsVirtualLink
VirtualLinkDf (clause 6.5.4)	Only the serviceAvailabilityLevel and qos attributes in VirtualLinkDf have been defined as properties of the NsVIPProfile data type. The flavour id is not mapped to a property as there is only one flavour of a particular NS virtual link per NS service template.
NsDataFlowMirroring (clause 6.5.7)	data_flow_mirroring property in NsVIPProfile
DataFlowInfo (clause 6.5.8)	policy with type toasca.policies.nfv.DataFlowInfo
Pnfd (clause 6.6.2)	A standalone TOSCA service template
PnfExtCpd (clause 6.6.4)	node template with type tosca.nodes.nfv.PnfExtCp
NsScalingAspect (clause 6.7.2)	policy with type tosca.policies.nfv.NsScalingAspects
VnfToLevelMapping (clause 6.7.4)	policy with type tosca.policies.nfv.VnfToLevelMapping and tosca.policies.nfv.VnfToInstantiationLevelMapping
VirtualLinkToLevelMapping (clause 6.7.5)	policy with type tosca.policies.nfv.VirtualLinkToLevelMapping and tosca.policies.nfv.VirtualLinkToInstantiationLevelMapping
NsToLevelMapping (clause 6.7.6)	policy with type tosca.policies.nfv.NsToLevelMapping and tosca.policies.nfv.NsToInstantiationLevelMapping
PaasServiceRequest (clause 6.8.2)	node template with type tosca.nodes.nfv.NsPaasServiceRequest
PaasServiceProfile (clause 6.8.3)	node template with type tosca.nodes.nfv.NsPaasServiceProfile

A.10 PNFD modelling design example

In this example, TOSCA-Simple-Profile-YAML-v1.3 [20] is used for designing and processing a PNFD TOSCA model.

The service template contains a node template of type MyCompany.examplePnf.1_0 which represents the main part of the PNF and a node template of type toasca.nodes.nfv.PnfExtCp representing the PNF external connection point.

examplePnf.yaml

This example illustrates a PNF with one external connection point, pnfExtCp_1. All the rest parts of the PNF are described as a single box, e.g. called pnf_mainPart in the PNFD.

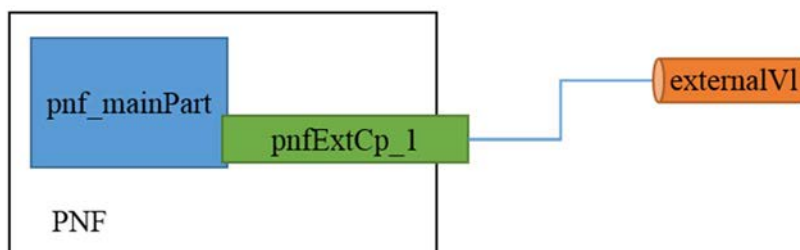


Figure A.10-1: examplePnf

```

tosca_definitions_version: tosca_simple_yaml_1_3

description: the service template of a PNFD

imports:
  - etsi_nfv_sol001_pnfd_types.yaml

node_types:
  MyCompany.examplePnf.1_0:
    derived_from: tosca.nodes.nfv.PNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a2233 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a2233
      function_description:
        type: string
        constraints: [ equal: an example PNF ]
        default: an example PNF
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_invariant_id:
        type: string
        constraints: [ equal: 1111-2222-ccaa-bbdd ]
        default: 1111-2222-ccaa-bbdd
      name:
        type: string
        constraints: [ equal: ExamplePnf ]
        default: ExamplePnf
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable

topology_template:
  substitution_mappings:
    node_type: MyCompany.examplePnf.1_0
  requirements:
    virtual_link: [ pnfExtCp_1, external_virtual_link ]

node_templates:
  pnf_mainPart:
    type: MyCompany.examplePnf.1_0
    properties:

  pnfExtCp_1:
    type: tosca.nodes.nfv.PnfExtCp
    properties:
      trunk_mode: false
      layer_protocols: [ ipv4 ]
      role: leaf
      description: External connection point to access this pnf
    requirements:
      # - external_virtual_link:
      - dependency: pnf_mainPart

```

A.11 NSD with Multiple deployment flavour modelling design example

Deployment flavours are represented as deployable TOSCA topology templates. This way one NS service template represents one NS deployment flavour, and different deployment flavours are described by different NS service templates. This is in line with the concept that different deployment flavours can define different topologies of the same NS, with different scaling aspects, different constituent VNFs, PNFs and nested NSs, and different internal connectivity.

In order to represent an NS, a top-level service template is used. This top-level service template contains a topology template with only an abstract NS node which defines the common parts of the different deployment flavours (such as designer, version and parts of the lifecycle management interface definition). It also sets a constraint on the deployment flavour property (the required value of the flavour_id property); this constraint comes from the NS instantiation request which contains a flavour_id selected among those available in the NSD.

As a result, the NFVO will look into the available further service templates representing the different NS deployment flavours of the NS and use the one that has the matching flavour_id property value to substitute for the abstract NS. These are the low-level service templates.

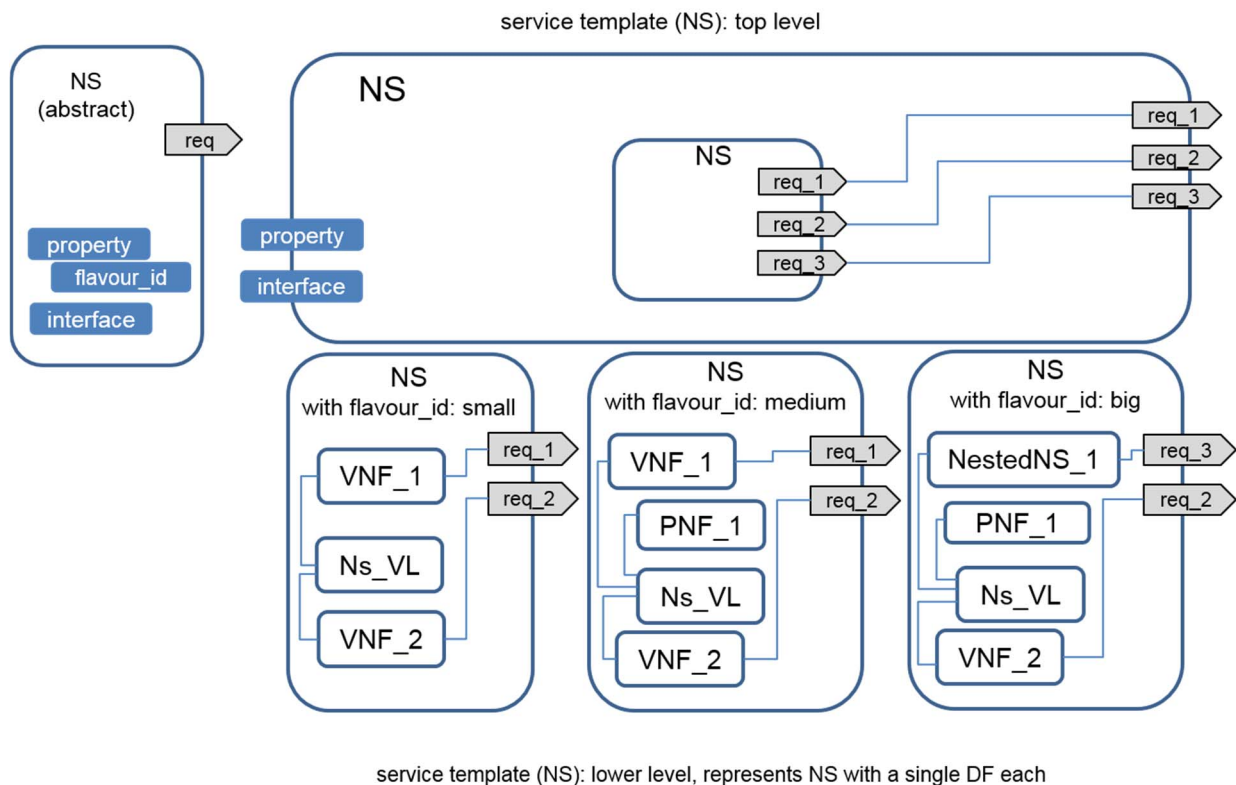


Figure A.11-1: NSD overview with multiple deployment flavour

An NSD contains a TOSCA top-level Service Template as entry point in the NSD file structure and one or more TOSCA low-level Service templates representing the different deployment flavours (see figure A.11-1). The NSD is interpreted by an NFVO. In this example, the templates describe two variants of the NS each corresponding to a deployment flavour: a small and a big one. The small NS consists of two VNFs one NS Virtual link and, the big VNF variant consists of three VNFs and one NS Virtual link.

NSD-top level MyExampleNs.yaml:

```
tosca_definitions_version: tosca_simple_yaml_1_3

description: my service
imports:
  - etsi_nfv_sol001_nsd_types.yaml
  - MyExampleNs_Type.yaml # contains the NS node type definition

topology_template:
  inputs:
    flavour_id:
      type: string
      description: NS deployment flavour selected by the consumer. It is provided in
the SOL005 API

  node_templates:
    myexampleNs:
      type: tosca.MyExampleNS
      directives:
        - substitute
      properties:
        flavour_id: {get_input: flavour_id}
      # requirements:
      #- virtual_link # mapped in lower-level templates
```

The MyExampleNs_Type.yaml file has the following content:

```
tosca_definitions_version: tosca_simple_yaml_1_3

description: type definition of tosca.MyExampleNS

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in ETSI
GS NFV-SOL 001

data_types:
  MyCompany.datatypes.nfv.NsInstantiateNsAdditionalParameters:
    derived_from: tosca.datatypes.nfv.NsOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
        required: true
        default: value_1
      parameter_2:
        type: string
        required: true
        default: value_2

node_types:
  tosca.MyExampleNS:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ valid_values: [ b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ] ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer:
        type: string
```

```

constraints: [ valid_values: [ MyCompany ] ]
default: MyCompany
name:
  type: string
  constraints: [ valid_values: [ ExampleService ] ]
  default: ExampleService
version:
  type: string
  constraints: [ valid_values: [ '1.0' ] ]
  default: '1.0'
invariant_id:
  type: string
  constraints: [ valid_values: [ 1111-2222-aaaa-bbbb ] ]
  default: 1111-2222-aaaa-bbbb
flavour_id:
  type: string
  constraints: [ valid_values: [ small, big ] ]
  default: small
requirements:
- virtual_link:
    capability: tosca.capabilities.nfv.VirtualLinkable
    relationship: tosca.relationships.nfv.VirtualLinksTo
interfaces:
  Nslcm:
    type: tosca.interfaces.nfv.Nslcm
    operations:
      instantiate:
        inputs:
          additional_parameters:
            type: MyCompany.datatypes.nfv.NsInstantiateNsAdditionalParameters
            required: false

```

The NS node template in the **myexample_NS.yaml** file is abstract and is subject to substitution; the lower-level templates in the subsequent sections provide these substitutions. The actual lower-level template is selected based on the node type and a value constraint on the `flavour_id` property.

Each low level service template contains a node template of type `tosca.MyExampleNS` with implementation of the LCM interfaces.

MyExampleNs (small): Lower level, contains 2 VNFs and 1 NS virtual link.

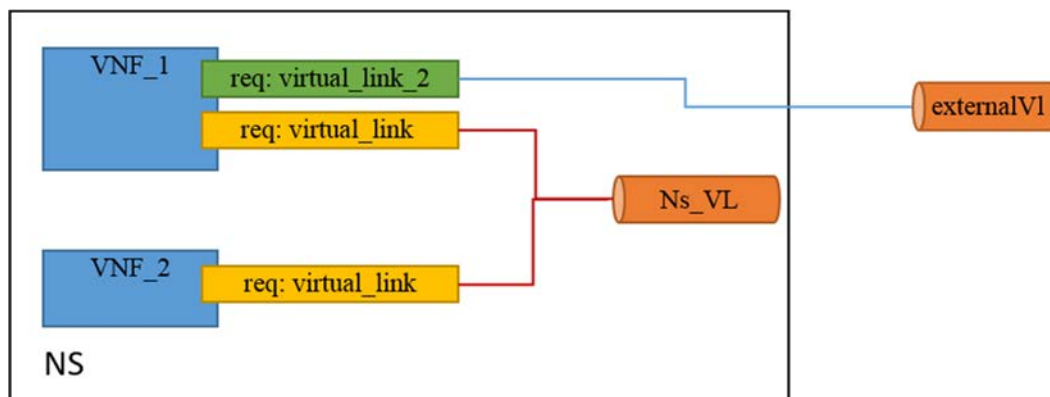


Figure A.11-2: MyExampleNs (simple): Lower level

MyExampleNs_small.yaml

```

tosca_definitions_version: tosca_simple_yaml_1_3

description: myExampleNs with small flavour
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in ETSI
GS NFV-SOL 001
  - MyExampleNs_Type.yaml # contains the NS node type definition
  - example_vnf1.yaml # uri of the yaml file which contains the
tosca.nodes.nfv.example_VNF1 node type definition, this file might be included in the
NSD file structure
  - example_vnf2.yaml # uri of the yaml file which contains the
tosca.nodes.nfv.example_VNF2 node type definition, this file might be included in the
NSD file structure

topology_template:
  substitution_mappings:
    node_type: tosca.MyExampleNS
  substitution_filter:
    properties:
      - flavour_id: { equal: small }
  requirements:
    virtual_link: [ VNF_1, virtual_link_2 ]

node_templates:
  MyExampleNS:
    type: tosca.MyExampleNS
    # properties:
    #
    interfaces:
      Nslcm:
        operations:
          instantiate:
            implementation: instantiate.workflow.yaml
          terminate:
            implementation: terminate.workflow.yaml

  VNF_1:
    type: tosca.nodes.nfv.example_VNF1
    properties:
      # no property assignments needed for required properties that have a default
value assigned in the node type definition, e.g. descriptor_id
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 2
        max_number_of_instances: 6
    requirements:
      - virtual_link: Ns_VL
      # - virtual_link_2: # map to virtual_link requirement of the NS node

  VNF_2:
    type: tosca.nodes.nfv.example_VNF2
    properties:
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 3
    requirements:
      - virtual_link_1: Ns_VL

```

```

Ns_VL:
  type: toska.nodes.nfv.NsVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ipv4]
      flow_pattern: mesh
    vl_profile:
      max_bitrate_requirements:
        root: 1000
      min_bitrate_requirements:
        root: 1000

```

MyExampleNs (big): Lower level, contains 3 VNFs and 1 NS virtual link.

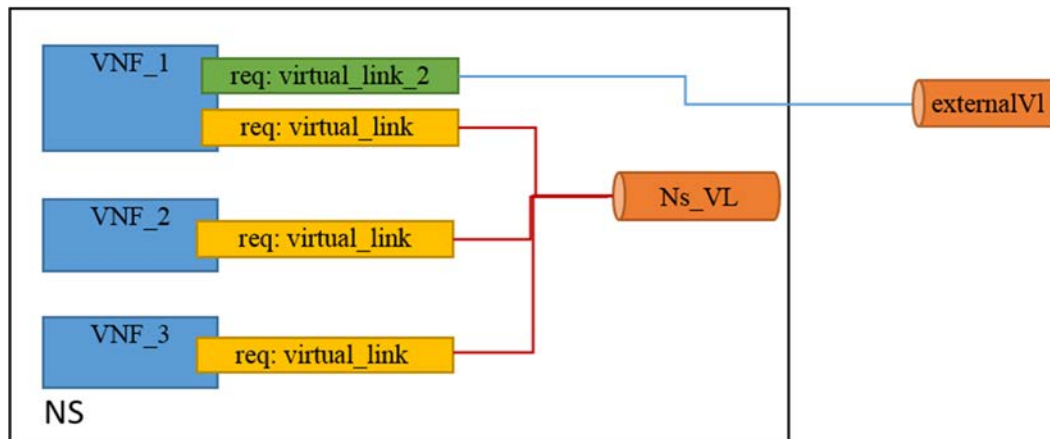


Figure A.11-3: MyExampleNs (big): Lower level

MyExampleNs_big.yaml

```

tosca_definitions_version: toska_simple_yaml_1_3

description: myExampleNs with big flavour
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in ETSI
    GS NFV-SOL 001
  - MyExampleNs_Type.yaml # contains the NS node type definition
  - example_vnf1.yaml # uri of the yaml file which contains the
    toska.nodes.nfv.example_VNF1 node type definition, this file might be included in the
    NSD file structure
  - example_vnf2.yaml # uri of the yaml file which contains the
    toska.nodes.nfv.example_VNF2 node type definition, this file might be included in the
    NSD file structure
  - example_vnf3.yaml # uri of the yaml file which contains the
    toska.nodes.nfv.example_VNF3 node type definition, this file might be included in the
    NSD file structure

topology_template:
  substitution_mappings:
    node_type: toska.MyExampleNS
  substitution_filter:
    properties:
      - flavour_id: { equal: big }
  requirements:
    virtual_link: [ VNF_1, virtual_link_2 ]

```

```

node_templates:
  MyExampleNS:
    type: tosa.MyExampleNS
    properties:
      flavour_id : big
    interfaces:
      Nslcm:
        operations:
          instantiate:
            implementation: instantiate.workflow.yaml
          terminate:
            implementation: terminate.workflow.yaml
          scale:
            implementation: scale.workbook.yaml

  VNF_1:
    type: tosa.nodes.nfv.example_VNF1
    properties:
      # no property assignments needed for required properties that have a default
value assigned in the node type definition, e.g. descriptor_id
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 2
        max_number_of_instances: 6
      requirements:
        - virtual_link_1: Ns_VL
        # - virtual_link_2: # map to virtual_link requirement of the NS node

  VNF_2:
    type: tosa.nodes.nfv.example_VNF2
    properties:
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 3
      requirements:
        - virtual_link_1: Ns_VL

  VNF_3:
    type: tosa.nodes.nfv.example_VNF3
    properties:
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 3
      requirements:
        - virtual_link_1: Ns_VL

  Ns_VL:
    type: tosa.nodes.nfv.NsVirtualLink
    properties:
      connectivity_type:
        layer_protocols: [ipv4]
        flow_pattern: mesh
      vl_profile:
        max_bitrate_requirements:
          root: 1000
        min_bitrate_requirements:
          root: 1000

```

A.12 NSD with nested NS design example

A TOSCA service template representing an NSD may contain a node template of some specific NS node type as one of its constituents. The latter is a nested NS. When the containing NS is deployed, the node template of the nested NS is substituted by the topology template representing the nested NS.

Figure A.12-1 illustrates a network service NS_1 that consists of one VNF (VNF_1), one NsVirtualLink (NS_VL_1) and one nested NS (NS_2).

The nested NS consists of two VNFs (VNF_3 and VNF_4) and one NsVirtualLink (NS_VL_2).

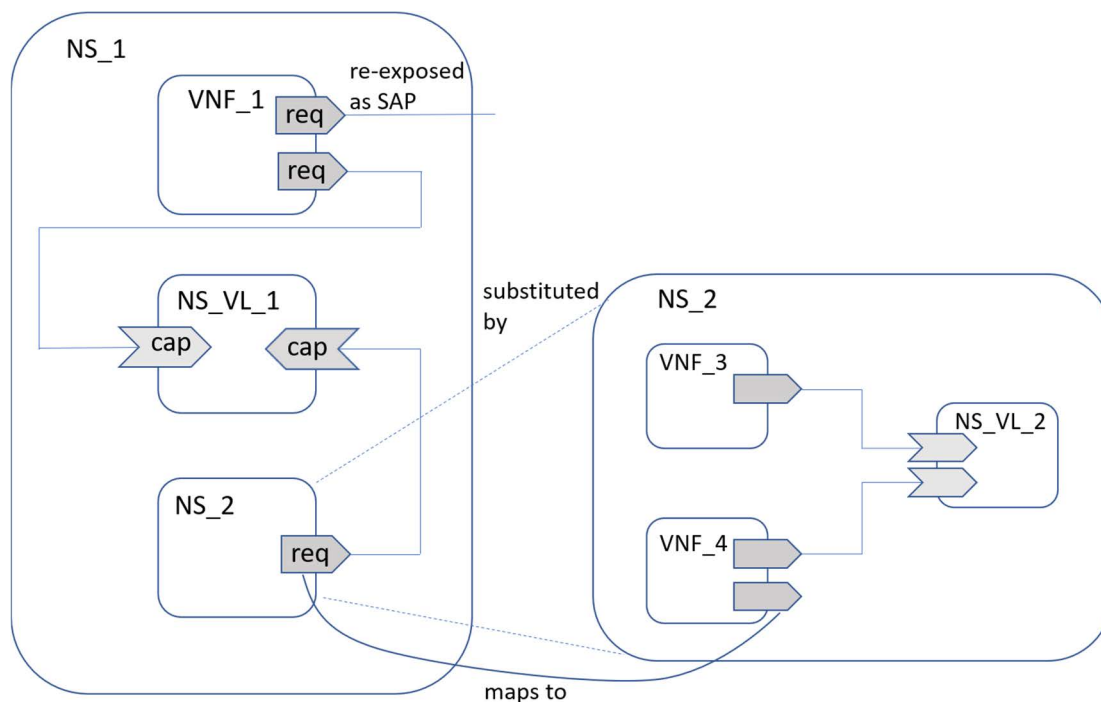


Figure A.12-1: Example of a network service containing a nested network service

```
tosca_definitions_version: toska_simple_yaml_1_3
description: myExampleNs with small flavour

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in ETSI
    GS NFV-SOL 001
  - MyExampleNs_TypeBis.yaml # contains the NS node type definition
  - MyExampleNS_2.yaml # uri of the yaml file which contains the toska.myExample.NS_2
    node type definition, this file might be included in the NSD file structure of NS_1
  - example_vnf1.yaml # uri of the yaml file which contains the
    toska.nodes.nfv.example_VNF1 node type definition, this file might be included in the
    NSD file structure of NS_1

topology_template:
  substitution_mappings:
    node_type: toska.MyExampleNS
  substitution_filter:
    properties:
      - flavour_id: { equal: small }
  requirements:
    virtual_link: [ VNF_1, virtual_link_2 ]

node_templates:
  NS_1:
```

```

    type: toska.MyExampleNS
    interfaces:
      Nslcm:
        operations:
          instantiate:
            implementation: instantiate.workflow.yaml
          terminate:
            implementation: terminate.workflow.yaml

  VNF_1:
    type: toska.nodes.nfv.example_VNF1
    properties:
      # no property assignments needed for required properties that have a default
value assigned in the node type definition, e.g. descriptor_id
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 2
        max_number_of_instances: 6
      requirements:
        - virtual_link_1: NS_VL_1
        # - virtual_link_2: # map to virtual_link requirement of the NS node

  NS_2:
    type: toska.myExample.NS_2
    properties:
      ns_profile:
        ns_instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 3

  NS_VL_1:
    type: toska.nodes.nfv.NsVirtualLink
    properties:
      connectivity_type:
        layer_protocols: [ipv4]
        flow_pattern: mesh
      vl_profile:
        max_bitrate_requirements:
          root: 1000
        min_bitrate_requirements:
          root: 1000

```

The contents of MyExampleNs_TypeBis.yaml file with the node type definition are as follows:

```

tosca_definitions_version: toska_simple_yaml_1_3

description: type definition of toska.MyExampleNS

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA types as defined in ETSI
GS NFV-SOL 001

node_types:
  toska.MyExampleNS:
    derived_from: toska.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ valid_values: [ b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ] ]

```

```

    default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
  designer:
    type: string
    constraints: [ valid_values: [ MyCompany ] ]
    default: MyCompany
  name:
    type: string
    constraints: [ valid_values: [ ExampleService ] ]
    default: ExampleService
  version:
    type: string
    constraints: [ valid_values: [ '1.0' ] ]
    default: '1.0'
  invariant_id:
    type: string
    constraints: [ valid_values: [ 1111-2222-aaaa-bbbb ] ]
    default: 1111-2222-aaaa-bbbb
  flavour_id:
    type: string
    constraints: [ valid_values: [ small, big ] ]
    default: small
  requirements:
    - virtual_link:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
  interfaces:
    Nslcm:
      type: tosca.interfaces.nfv.Nslcm

```

The following snippet shows the service template representing the NSD NS_2. In this example, NS_2 supports one single deployment flavour.

MyExampleNS_2.yaml:

```

tosca_definitions_version: tosca_simple_yaml_1_3

description: Relational database, simple

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of NSD related TOSCA types as defined in ETSI
    GS NFV-SOL 001
  - example_vnf3.yaml # uri of the yaml file which contains the definition of
    tosca.nodes.nfv.example_VNF3, this file might be included in the NSD file structure
  - example_vnf4.yaml # uri of the yaml file which contains the definition of
    tosca.nodes.nfv.example_VNF4, this file might be included in the NSD file structure

node_types:
  tosca.myExample.NS_2:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ valid_values: [ c1bb0ab8-deab-4fa7-95ed-4840d70a3574 ] ]
        default: c1bb0ab8-deab-4fa7-95ed-4840d70a3574
      designer:
        type: string
        constraints: [ valid_values: [ MyCompany ] ]
        default: MyCompany
    name:
      type: string

```



```

    constraints: [ valid_values: [ myExample2Service ] ]
    default: myExample2Service
  version:
    type: string
    constraints: [ valid_values: [ '1.0.0.0' ] ]
    default: '1.0.0.0'
  invariant_id:
    type: string
    constraints: [ valid_values: [ aaaa-bbbb-cccc-dddd ] ]
    default: aaaa-bbbb-cccc-dddd
  flavour_id:
    type: string
    constraints: [ valid_values: [ simple ] ]
    default: simple

topology_template:
  substitution_mappings:
    node_type: tosca.myExample.NS_2
  requirements:
    virtual_link: [ VNF_4, virtual_link_2 ] # the External connection point of
                                           # VNF_2 is exposed as the Sap

node_templates:
  NS_2:
    type: tosca.myExample.NS_2
    interfaces:
      Nslcm:
        operations:
          instantiate:
            implementation: instantiate.workflow.yaml
          terminate:
            implementation: terminate.workflow.yaml

  VNF_3:
    type: tosca.nodes.nfv.example_VNF3
    properties:
      # no property assignments needed for required properties that have a default
value assigned in the node type definition, e.g. descriptor_id
      flavour_id: simple
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 2
        max_number_of_instances: 6
    requirements:
      - virtual_link: NS_VL_2

  VNF_4:
    type: tosca.nodes.nfv.example_VNF4
    properties:
      flavour_id: simple
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 3
    requirements:
      - virtual_link_1: NS_VL_2
      # - virtual_link_2: # map to virtual_link requirement of the NS node

  NS_VL_2:
    type: tosca.nodes.nfv.NsVirtualLink
    properties:

```

```

connectivity_type:
  layer_protocols: [ipv4]
  flow_pattern: mesh
vl_profile:
  max_bitrate_requirements:
    root: 1000
  min_bitrate_requirements:
    root: 1000

```

The following template fragment is part of the content in the example_vnf4.yaml. Template fragments for example_vnf1.yaml, example_vnf2.yaml and example_vnf3.yaml are available in clause A.14.

example_vnf4.yaml

```

tosca_definitions_version: tosca_simple_yaml_1_3

description: Example VNF4 type

imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of VNFD types as defined in NFV SOL 001 GS

node_types:
  tosca.nodes.nfv.example_VNF4:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1184 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1184
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: Example_VNF4 ]
        default: Example_VNF4
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      flavour_id:
        type: string
        constraints: [ equal: simple ]
        default: simple
      flavour_description:
        type: string
        default: ""
      vnfm_info:
        type: list
        entry_schema:
          type: string
          constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
          default: [ '0:MyCompany-1.0.0' ]
    requirements:
      - virtual_link_1:

```

```

capability: tosca.capabilities.nfv.VirtualLinkable
relationship: tosca.relationships.nfv.VirtualLinksTo
occurrences: [ 0, 1 ]
- virtual_link_2:
  capability: tosca.capabilities.nfv.VirtualLinkable
  relationship: tosca.relationships.nfv.VirtualLinksTo
  occurrences: [ 0, 1 ]

```

A.13 Virtual IP address connection point

Virtual IP address connection points (VipCps) are used to allocate one or multiple IP addresses that are shared by other CP instances, which may be instances of the same or of different VduCp or VnfExtCp nodes.

Load balancing

In the following example two or more instances of a particular VNFC are created. The respective instances of the VduCp, in addition to their default IP address which is assigned according to the 'protocol' property, share a virtual IP address. The multiple instances are created for load sharing purposes.

In this particular example the VduCp is re-exposed as VnfExtCp. Therefore the VipCp is also re-exposed as VnfExtCp.

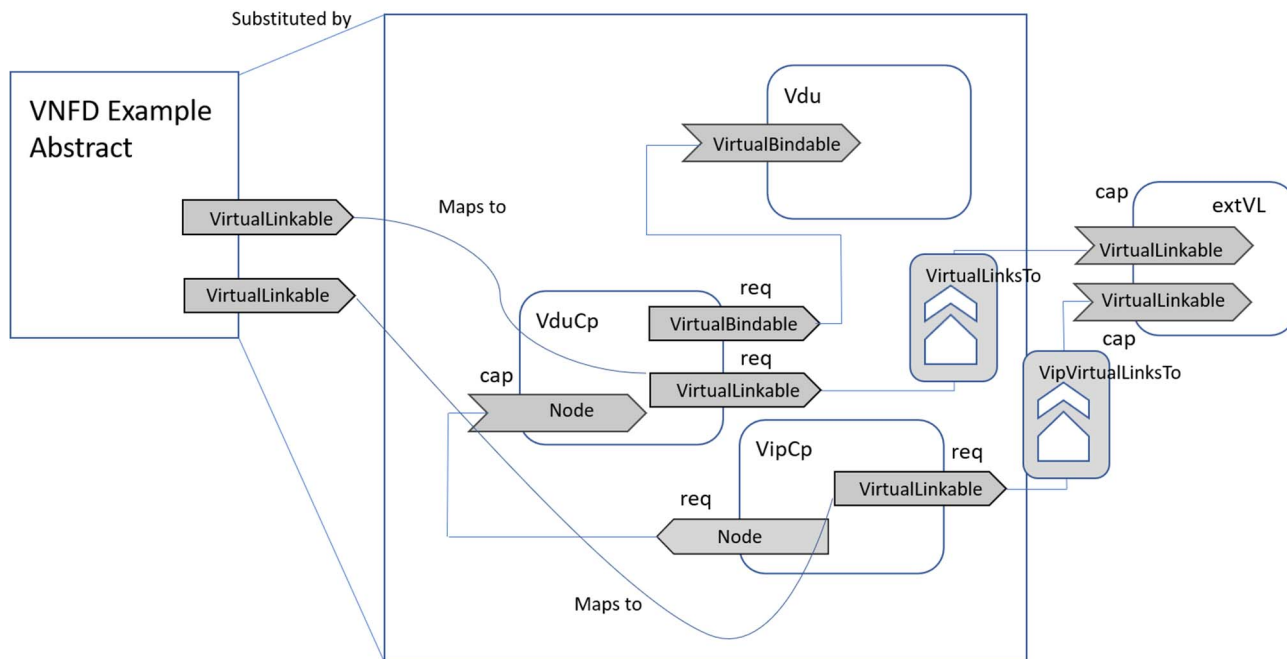


Figure A.13-1: VNFD with a VDU connection point acting as VnfExtCp and sharing a virtual IP address

In this example the VduCp and the VipCp are exposed as VnfExtCps. Thus, the VNF abstract node has two requirements for a VirtualLinkable capability. One of them uses the VirtualLinksTo relationship and the other one uses the VipVirtualLinksTo relationship. Both requirements are considered in the substitution mapping.

The following service template shows the relevant parts of the TOSCA VNFD corresponding to figure A.13-1. For simplicity, a single deployment flavour VNF is assumed.

```

tosca_definitions_version: tosca_simple_yaml_1_3
description: Relational database, simple
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001 for a VNFD
node_types:

```

```

MyCompany.SunshineDB.1_0.1_0:
  derived_from: toska.nodes.nfv.VNF
  properties:
    descriptor_id:
      type: string
      constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
      default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
    provider:
      type: string
      constraints: [ equal: MyCompany ]
      default: MyCompany
    product_name:
      type: string
      constraints: [ equal: SunshineDB ]
      default: SunshineDB
    software_version:
      type: string
      constraints: [ equal: '1.0' ]
      default: '1.0'
    descriptor_version:
      type: string
      constraints: [ equal: '1.0' ]
      default: '1.0'
    flavour_id:
      type: string
      constraints: [ valid_values: [ simple ] ]
      default: simple
    flavour_description:
      type: string
      default: ""
    vnfm_info:
      type: list
      entry_schema:
        type: string
        constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
        default: [ '0:MyCompany-1.0.0' ]
  interfaces:
    Vnflcm:
      operations:
        instantiate: {}
        terminate: {}
  requirements:
    - virtual_link:
        capability: toska.capabilities.nfv.VirtualLinkable
        relationship: toska.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 1 ]
    - virtual_link_vip:
        capability: toska.capabilities.nfv.VirtualLinkable
        relationship: toska.relationships.nfv.VipVirtualLinksTo
        occurrences: [ 0, 1 ]

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
  requirements:
    virtual_link: [ Vdu-A-Cp, virtual_link ]
    virtual_link_vip: [ VipCp, virtual_link ]

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0

```

```
# properties:
# omitted for brevity
# interfaces:
# omitted for brevity
```

VDU-A:

```
type: toasca.nodes.nfv.Vdu.Compute
properties:
  name: VDU-A
  description: VDU-A description
  vdu_profile:
    min_number_of_instances: 2
    max_number_of_instances: 5
  # other properties omitted for brevity
capabilities:
  virtual_compute:
    properties: # following properties are required.
      virtual_memory:
        virtual_mem_size: 1 GB
      virtual_cpu:
        num_virtual_cpu: 1
# requirements:
# omitted for brevity
```

Vdu-A-Cp:

```
type: toasca.nodes.nfv.VduCp
properties:
  protocol: [associated_layer_protocol: ipv4 ]
  trunk_mode: false
  layer_protocols: [ ipv4 ]
  role: leaf
  description: Internal connection point on an VL
requirements:
  - virtual_binding: VDU-A
  #- virtual_link: # the target node is determined in the NSD
```

VipCp:

```
type: toasca.nodes.nfv.VipCp
properties:
  vip_function: load_balance
  protocol:
    - associated_layer_protocol: ipv4
      address_data:
        - address_type: ip_address
          l3_address_data:
            ip_address_assignment: true
            floating_ip_activated: false
            number_of_ip_address: 1
  trunk_mode: false
  layer_protocols: [ ipv4 ]
  description: >
```

Virtual IP connection point. It holds one IP address shared by all instances (between 2 and 5 according to the vdu_profile) of the Vdu-A-Cp node. Floating IP address is not used in the VipCp. Thus, incoming packets are forwarded with unmodified destination address to one of the instances of Vdu-A-Cp. A router external to the VNF with Equal-Cost Multi-Path (ECMP) load balancing functionality is assumed to be properly configured to route the packets accordingly to the available instances applying load balancing, i.e. one packet is only forwarded to one instance.

```
requirements:
```

```
- target: Vdu-A-Cp
# - virtual_link: # the target node is determined in the NSD
```

High availability

In the following example, a VNF uses two VNFCs to provide high availability of a service. One of them is the active one receiving IP packets, the other one is in stand-by mode. The VNF logic determines which VNFC is the active and which is the stand-by. The respective VduCp instances, in addition to their default IP address which is assigned according to the 'protocol' property, share a virtual IP address. At any point in time, only one of the VduCp instances, the one belonging to the active VNFC, is bound to the virtual IP address, i.e. only one receives the packets. During the life of the VNF the binding may change, for example in case of failure of the active VNFC, or if determined by the VNF logic. In order to bind the virtual IP address, the active VNFC sends a Gratuitous ARP (G-ARP) message with the mapping of the VIP address to its MAC address. A router external to the VNF updates its routing tables when receiving the G-ARP and thereafter routes packets that have the virtual IP address as destination address to the active VNFC.

In this particular example the VduCps are re-exposed as VnfExtCps. Therefore the VipCp is also re-exposed as VnfExtCp.

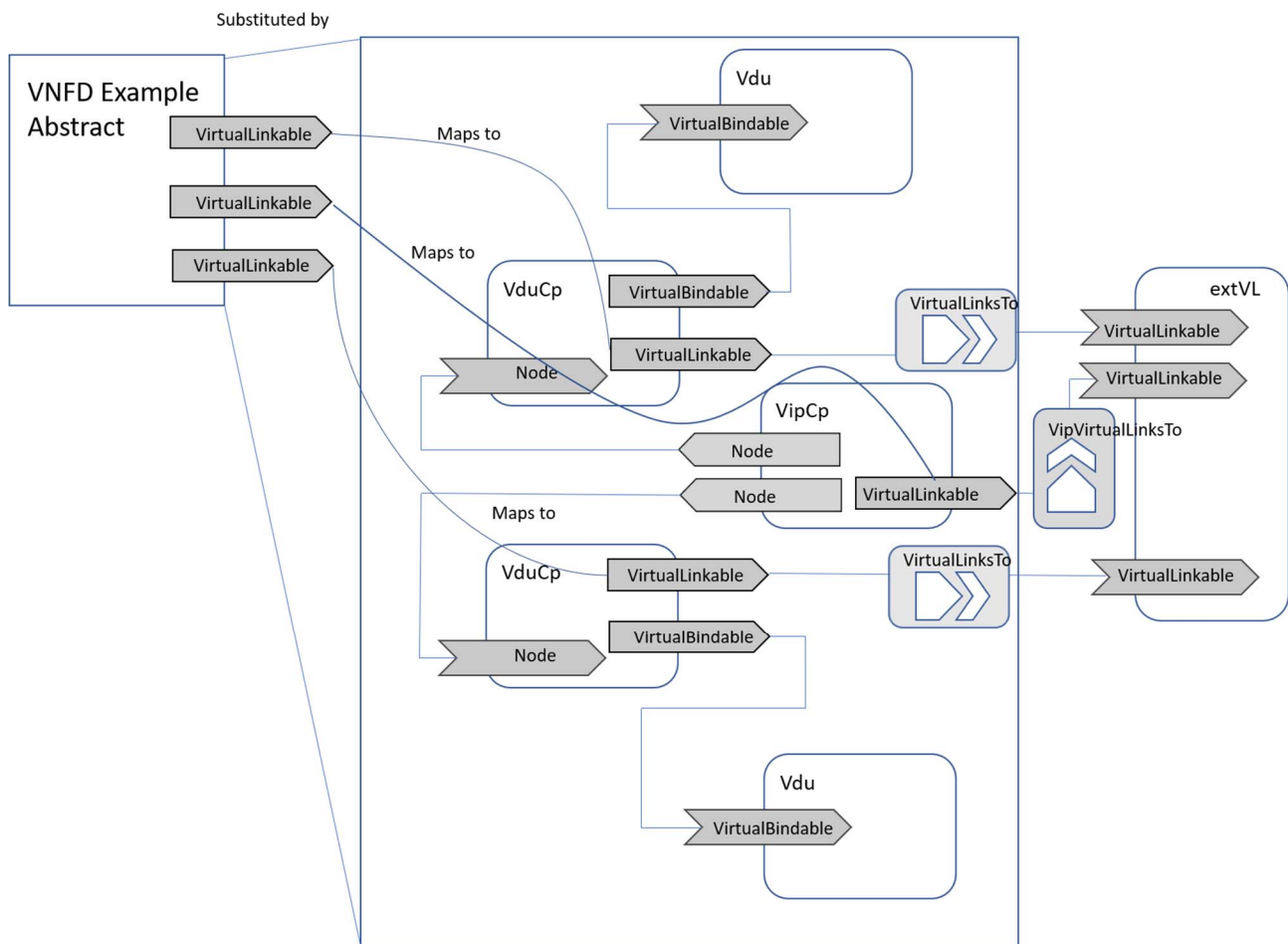


Figure A.13-2: VNFD with two VDU connection points acting as VnfExtCps and sharing a virtual IP address

In this example the two VduCps and the VipCp are exposed as VnfExtCps. Therefore the VNF abstract node has three requirements for a VirtualLinkable capability. Two of them use the VirtualLinksTo relationship and the third one uses the VipVirtualLinksTo relationship. The three of them are considered in the substitution mapping.

The following service template shows the relevant parts of the TOSCA VNFD corresponding to figure A.13-2. For simplicity, a single deployment flavour VNF is assumed.

```

tosca_definitions_version: toska_simple_yaml_1_3
description: Relational database, simple
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001 for a VNFD

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: toska.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: SunshineDB ]
        default: SunshineDB
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      flavour_id:
        type: string
        constraints: [ valid_values: [ simple ] ]
        default: simple
      flavour_description:
        type: string
        default: ""
      vnfm_info:
        type: list
        entry_schema:
          type: string
          constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
          default: [ '0:MyCompany-1.0.0' ]
    # interfaces:
    #   # omitted for brevity
  requirements:
    - virtual_link:
        capability: toska.capabilities.nfv.VirtualLinkable
        relationship: toska.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 1 ]
    - virtual_link_sby:
        capability: toska.capabilities.nfv.VirtualLinkable
        relationship: toska.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 1 ]
    - virtual_link_vip:
        capability: toska.capabilities.nfv.VirtualLinkable
        relationship: toska.relationships.nfv.VipVirtualLinksTo
        occurrences: [ 0, 1 ]

```

```
topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
  requirements:
    virtual_link: [ Vdu-A-Cp, virtual_link ]
    virtual_link_sby: [ Vdu-B-Cp, virtual_link ]
    virtual_link_vip: [ VipCp, virtual_link ]

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    # properties:
    #   omitted for brevity
    # interfaces:
    #   omitted for brevity

  VDU-A:
    type: toasca.nodes.nfv.Vdu.Compute
    properties:
      name: VDU-A
      description: VDU-A description

    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 1
      # other properties omitted for brevity
    capabilities:
      virtual_compute:
        properties: # following properties are required.
          virtual_memory:
            virtual_mem_size: 1 GB
          virtual_cpu:
            num_virtual_cpu: 1

      # requirements:
      #   omitted for brevity

  VDU-B:
    type: toasca.nodes.nfv.Vdu.Compute
    properties:
      name: VDU-B
      description: VDU-B description
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 1
      # other properties omitted for brevity
    capabilities:
      virtual_compute:
        properties: # following properties are required.
          virtual_memory:
            virtual_mem_size: 1 GB
          virtual_cpu:
            num_virtual_cpu: 1

      # requirements:
      #   omitted for brevity

  Vdu-A-Cp:
    type: toasca.nodes.nfv.VduCp
```



```

properties:
  protocol: [associated_layer_protocol: ipv4 ]
  trunk_mode: false
  layer_protocols: [ ipv4 ]
  role: leaf
  description: Internal connection point on an VL
requirements:
  - virtual_binding: VDU-A
  #- virtual_link: # the target node is determined in the NSD

```

Vdu-B-Cp:

```

type: tosca.nodes.nfv.VduCp
properties:
  protocol: [associated_layer_protocol: ipv4 ]
  trunk_mode: false
  layer_protocols: [ ipv4 ]
  role: leaf
  description: Internal connection point on an VL
requirements:
  - virtual_binding: VDU-B
  #- virtual_link: # the target node is determined in the NSD

```

VipCp:

```

type: tosca.nodes.nfv.VipCp
properties:
  vip_function: high_availability
  protocol:
    - associated_layer_protocol: ipv4
      address_data:
        - address_type: ip_address
          l3_address_data:
            ip_address_assignment: true
            floating_ip_activated: true
            number_of_ip_address: 1
      trunk_mode: false
      layer_protocols: [ ipv4 ]
  description: >

```

Virtual IP connection point. It holds one IP address shared by the instances of the Vdu-A-Cp and the Vdu-B-Cp nodes (one instance of each). Floating IP address is used. Thus, incoming packets are first NATed to the virtual IP address and then forwarded with the virtual IP address as destination address to the instance of Vdu-A-Cp or Vdu-B-Cp that currently has the address binding.

```

requirements:
  - target: Vdu-A-Cp
  - target: Vdu-B-Cp
  # - virtual_link: # the target node is determined in the NSD

```

In the example above, the VipCp uses a floating IP address. Thus, the incoming packets are expected to have the floating IP as destination address and they are first NATed to the virtual IP address and then forwarded to the instance of the VduCp that currently has the binding to the virtual IP address.

If the VipCp does not use floating IP address, the incoming packets are expected to have the virtual IP address as destination address.

A.14 NSD VNF Forwarding Graph design example

The following template fragment illustrates a VNF FG data model for a Network Service. The NS consists of VNF_1, VNF_2, VNF_3 and NsVirtualLink_1 as its constituents. VNF_1, VNF_2 and VNF_3 node templates have virtual link requirements pointing to node templates of the type `tosca.nodes.nfv.Forwarding` defined in clause 7.8.8 which in turn have virtual link requirements pointing to the NS virtual links *or* to external virtual links (i.e. *transport links beyond the SAPs*) to which these VNFs are attached.

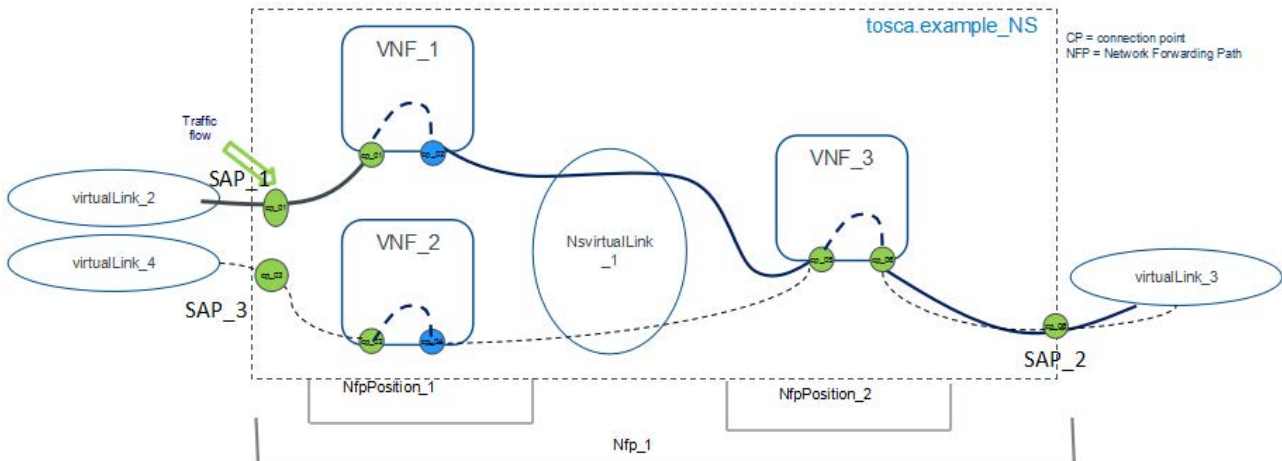


Figure A.14-1: Example Network Forwarding Path

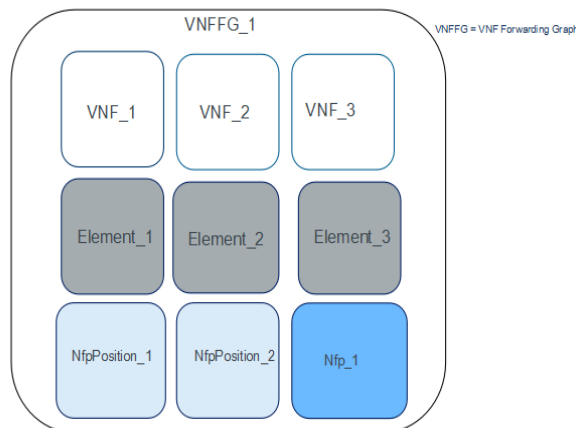


Figure A.14-2: Example VNFFG_1 group with constituent elements

```
tosca_definitions_version: toska_simple_yaml_1_3
description: VNF FG Model for example_NS
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of NSD related TOSCA types as defined in
    ETSI GS NFV-SOL 001
  - example_vnf1.yaml # uri of the yaml file which contains the
    toska.nodes.nfv.example_VNF1 node type definition, this file might be included in
    the NSD file structure
  - example_vnf2.yaml # uri of the yaml file which contains the
    toska.nodes.nfv.example_VNF2 node type definition, this file might be included in
    the NSD file structure
```

- example_vnf3.yaml # uri of the yaml file which contains the
 toska.nodes.nfv.example_VNF3 node type definition, this file might be included in
 the NSD file structure

```
data_types:
  MyCompany.datatypes.nfv.NsInstantiateAdditionalParameters:
    derived_from: toska.datatypes.nfv.NsOperationAdditionalParameters

node_types:
  toska.example_NS:
    derived_from: toska.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
    name:
      type: string
      constraints: [ equal: ExampleService ]
      default: ExampleService
    version:
      type: string
      constraints: [ equal: '1.0' ]
      default: '1.0'
    invariant_id:
      type: string
      constraints: [ equal: 1111-2222-aaaa-bbbb ]
      default: 1111-2222-aaaa-bbbb
    flavour_id:
      type: string
      constraints: [ equal: simple ]
      default: simple
    interfaces:
      Nslcm:
        type: toska.interfaces.nfv.Nslcm
        operations:
          instantiate:
            inputs:
              additional_parameters:
                type: MyCompany.datatypes.nfv.NsInstantiateAdditionalParameters
                required: false
    requirements:
      - virtual_link:
          capability: toska.capabilities.nfv.VirtualLinkable
          relationship: toska.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 0 ]
      - virtual_link_2:
          capability: toska.capabilities.nfv.VirtualLinkable
          relationship: toska.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 1 ]
      - virtual_link_3:
          capability: toska.capabilities.nfv.VirtualLinkable
          relationship: toska.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 1 ]
      - virtual_link_4:
```

```

        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 1 ]

topology_template:
  substitution_mappings:
    node_type: tosca.example_NS
    requirements:
      virtual_link_2: [ VNF_1_forward_1, virtual_link ] # the requirement of SAP_1
      virtual_link_3: [ VNF_3_forward_6, virtual_link ] # the requirement of SAP_2
      virtual_link_4: [ VNF_2_forward_3, virtual_link ] # the requirement of SAP_3

  node_templates:
    my_service:
      type: tosca.example_NS
      properties:
        # . . .
      interfaces:
        Nslcm:
          operations:
            instantiate:
              implementation: instantiate.workflow.yaml
            terminate:
              implementation: terminate.workflow.yaml

    NsVirtualLink_1:
      type: tosca.nodes.nfv.NsVirtualLink
      properties:
        vl_profile:
          min_bitrate_requirements:
            root: 100000
          max_bitrate_requirements:
            root: 200000
        connectivity_type:
          layer_protocols: [ ipv4 ]
        # . . .

    VNF_1:
      type: tosca.nodes.nfv.example_VNF1
      properties:
        # . . .
      requirements:
        - virtual_link: VNF_1_forward_1 # related to cp_01 in VNF_1
        - virtual_link_2: VNF_1_forward_2 # related to cp_02 in VNF_1

    VNF_1_forward_1:
      type: tosca.nodes.nfv.Forwarding

    VNF_1_forward_2:
      type: tosca.nodes.nfv.Forwarding
      requirements:
        - virtual_link: NsVirtualLink_1

    VNF_2:
      type: tosca.nodes.nfv.example_VNF2
      properties:
        # . . .

```

```

    requirements:
      - virtual_link: VNF_2_forward_3 # related to cp_03 in VNF_2
      - virtual_link_4: VNF_2_forward_4 # related to cp_04 in VNF_2

VNF_2_forward_3:
  type: tosa.nodes.nfv.Forwarding

VNF_2_forward_4:
  type: tosa.nodes.nfv.Forwarding
  requirements:
    - virtual_link: NsVirtualLink_1

VNF_3:
  type: tosa.nodes.nfv.example_VNF3
  properties:
    # . . .
  requirements:
    - virtual_link: VNF_3_forward_5 # related to cp_05 in VNF_3
    - virtual_link_3: VNF_3_forward_6 # related to cp_06 in VNF_3

VNF_3_forward_5:
  type: tosa.nodes.nfv.Forwarding
  requirements:
    - virtual_link: NsVirtualLink_1

VNF_3_forward_6:
  type: tosa.nodes.nfv.Forwarding

# NfpPositionElement (Service Function) for VNF_1
Element_1:
  type: tosa.nodes.nfv.NfpPositionElement
  requirements:
    - profile_element:
        node: VNF_1_forward_1
        capability: forwarding
    - profile_element:
        node: VNF_1_forward_2
        capability: forwarding

# NfpPositionElement (Service Function) for VNF_2
Element_2:
  type: tosa.nodes.nfv.NfpPositionElement
  requirements:
    - profile_element:
        node: VNF_2_forward_3
        capability: forwarding
    - profile_element:
        node: VNF_2_forward_4
        capability: forwarding

# NfpPositionElement (Service Function) for VNF_3
Element_3:
  type: tosa.nodes.nfv.NfpPositionElement
  requirements:
    - profile_element:
        node: VNF_3_forward_5
        capability: forwarding

```

```

    - profile_element:
        node: VNF_3_forward_6
        capability: forwarding

# NfpPosition_1 with Element_1 and Element_2 as constituents
NfpPosition_1:
    type: toasca.nodes.nfv.NfpPosition
    properties:
        forwarding_behaviour: lb
    requirements:
        - element: Element_1
        - element: Element_2

# NfpPosition_2 with Element_3 as constituents
NfpPosition_2:
    type: toasca.nodes.nfv.NfpPosition
    properties:
        forwarding_behaviour: all
    requirements:
        - element: Element_3

Nfp_1:
    type: toasca.nodes.nfv.NFP
    requirements:
        - nfp_position: NfpPosition_1
        - nfp_position: NfpPosition_2

policies:
    - NfpRule_1:
        type: toasca.policies.nfv.NfpRule
        properties:
            ether_destination_address: 00:0a:95:9d:68:16
            ether_source_address: 00:A0:C9:14:C8:29
            ether_type: ipv4
            vlan_tag:
                - "10"
                - "20"
                - "30"
            protocol: tcp
            dscp: "101111"
            source_port_range: [ 5000, 15000 ]
            destination_port_range: [ 800, 8080 ]
            source_ip_address_prefix: 10.10.10.0
            destination_ip_address_prefix: 125.1.12.111
            extended_criteria:
                - starting_point: 3
                  length: 4
                  value: "1000"
            targets: [ Nfp_1 ]

groups:
    VNFFG_1:
        type: toasca.groups.nfv.VNFFG
        properties:
            description: VNF Forwarding Graph for example_NS
            members: [ Nfp_1, VNF_1, VNF_2, VNF_3, NsVirtualLink_1, Element_1, Element_2,
Element_3 ]

```

The following template fragment is part of the content in the example_vnf1.yaml.

```
tosca_definitions_version: tosca_simple_yaml_1_3
description: VNF Descriptor for VNF1
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of VNFD related TOSCA types as defined in
ETSI GS NFV-SOL 001
node_types:
  tosca.nodes.nfv.example_VNF1:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-2222-4fa7-95ed-4840d70a1179 ]
        default: b1bb0ce7-2222-4fa7-95ed-4840d70a1179
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: Example_VNF1 ]
        default: Example_VNF1
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      flavour_id:
        type: string
        constraints: [ equal: simple ]
        default: simple
      flavour_description:
        type: string
        default: ""
      vnfm_info:
        type: list
        entry_schema:
          type: string
          constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
          default: [ '0:MyCompany-1.0.0' ]
    requirements:
      - virtual_link_1:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 1 ]
      - virtual_link_2:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 1 ]
  topology_template:
    substitution_mappings:
      node_type: tosca.nodes.nfv.example_VNF1
    requirements:
      virtual_link: [ cp_01, external_virtual_link ]
      virtual_link_2: [ cp_02, external_virtual_link ]
```

```

node_templates:
  cp_01:
    type: toska.nodes.nfv.VnfExtCp
    properties:
      layer_protocols: [ ipv4 ]
      # . . . . .
    requirements:
      - internal_virtual_link: intVL-A

  cp_02:
    type: toska.nodes.nfv.VnfExtCp
    properties:
      layer_protocols: [ ipv4 ]
      # . . . . .
    requirements:
      - internal_virtual_link: intVL-A

  intVL-A:
    type: toska.nodes.nfv.VnfVirtualLink
    properties:
      connectivity_type:
        layer_protocols: [ ipv4 ]
      vl_profile:
        max_bitrate_requirements:
          root: 1000000
          leaf: 100000
        min_bitrate_requirements:
          root: 100000
          leaf: 10000

```

The following template fragment is part of the content in the example_vnf2.yaml.

```

tosca_definitions_version: toska_simple_yaml_1_3
description: VNF Descriptor for VNF2
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of VNFD related TOSCA types as defined in
    ETSI GS NFV-SOL 001
node_types:
  toska.nodes.nfv.example_VNF2:
    derived_from: toska.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-2222-4fa7-95ed-4840d70a1182 ]
        default: b1bb0ce7-2222-4fa7-95ed-4840d70a1182
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: Example_VNF2 ]
        default: Example_VNF2
      software_version:
        type: string
        constraints: [ equal: '1.0.0' ]
        default: '1.0.0'
      descriptor_version:

```



```

    type: string
    constraints: [ equal: '1.0.0' ]
    default: '1.0.0'
  flavour_id:
    type: string
    constraints: [ equal: simple ]
    default: simple
  flavour_description:
    type: string
    default: ""
  vnfm_info:
    type: list
    entry_schema:
      type: string
      constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
      default: [ '0:MyCompany-1.0.0' ]
  requirements:
    - virtual_link_1:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 1 ]
    - virtual_link_2:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 1 ]
    - virtual_link_4:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 1 ]

topology_template:
  substitution_mappings:
    node_type: tosca.nodes.nfv.example_VNF2
  requirements:
    virtual_link: [ cp_03, external_virtual_link ]
    virtual_link_4: [ cp_04, external_virtual_link ]

node_templates:
  cp_03:
    type: tosca.nodes.nfv.VnfExtCp
    properties:
      layer_protocols: [ ipv4 ]
      # . . . . .
    requirements:
      - internal_virtual_link: intVL-A

  cp_04:
    type: tosca.nodes.nfv.VnfExtCp
    properties:
      layer_protocols: [ ipv4 ]
      # . . . . .
    requirements:
      - internal_virtual_link: intVL-A

  intVL-A:
    type: tosca.nodes.nfv.VnfVirtualLink
    properties:
      connectivity_type:
        layer_protocols: [ ipv4 ]

```

```

    vl_profile:
      max_bitrate_requirements:
        root: 1000000
        leaf: 100000
      min_bitrate_requirements:
        root: 100000
        leaf: 10000

```

The following template fragment is part of the content in the example_vnf3.yaml.

```

tosca_definitions_version: tosca_simple_yaml_1_3
description: VNF Descriptor for VNF3
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of VNFD related TOSCA types as defined in
ETSI GS NFV-SOL 001
node_types:
  tosca.nodes.nfv.example_VNF3:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-2222-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-2222-4fa7-95ed-4840d70a1177
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: Example_VNF3 ]
        default: Example_VNF3
      software_version:
        type: string
        constraints: [ equal: '1.0.0' ]
        default: '1.0.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0.0' ]
        default: '1.0.0'
      flavour_id:
        type: string
        constraints: [ equal: simple ]
        default: simple
      flavour_description:
        type: string
        default: ""
      vnfm_info:
        type: list
        entry_schema:
          type: string
          constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
          default: [ '0:MyCompany-1.0.0' ]
    requirements:
      - virtual_link_1:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 1 ]
      - virtual_link_2:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo

```

```

        occurrences: [ 0, 1 ]
    - virtual_link_3:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 1 ]

topology_template:
  substitution_mappings:
    node_type: tosca.nodes.nfv.example_VNF3
  requirements:
    virtual_link: [ cp_05, external_virtual_link ]
    virtual_link_3: [ cp_06, external_virtual_link ]

  node_templates:
    cp_05:
      type: tosca.nodes.nfv.VnfExtCp
      properties:
        layer_protocols: [ ipv4 ]
        # . . .
      requirements:
        - internal_virtual_link: intVL-A

    cp_06:
      type: tosca.nodes.nfv.VnfExtCp
      properties:
        layer_protocols: [ ipv4 ]
        # . . .
      requirements:
        - internal_virtual_link: intVL-A

    intVL-A:
      type: tosca.nodes.nfv.VnfVirtualLink
      properties:
        connectivity_type:
          layer_protocols: [ ipv4 ]
        vl_profile:
          max_bitrate_requirements:
            root: 1000000
            leaf: 100000
          min_bitrate_requirements:
            root: 100000
            leaf: 10000

```

A.15 Auto-scale and auto-heal design

A.15.1 Introduction

This clause describes the Auto-scale and auto-heal policy design in a VNFD or NSD.

A.15.2 Auto-scale and auto-heal design with use of VNF indicator in VNFD

This clause illustrates an example of a VNF with two VNF indicators, one VNF indicator related to the utilization of the VNF and one related to the VNF health.

The modelling and handling of the indicator in the VNFD service template involves the following definitions:

- A VNF specific VNF indicator interface type definition that includes one notification for each VNF indicator.
- New attribute definition included in the VNF specific node type definition for each of the indicators.
- Two attributes to hold the values of the current scale level of the 'call_proc' and 'database' scaling aspects.

NOTE: It is assumed that this VNF has the same scaling aspects as defined in the example in clause A.6 ('call_proc' and 'database').

- An interface definition of the VNF specific VnfIcm type with input definitions of additional_parameters for the heal operation.
- An interface definition of the VNF specific VNF indicator type with an output definition for each notification indicating the attribute modified by the output of the notification.
- A VNF specific auto_scale and auto_heal policy definitions of tosa.policies.nfv.VnfIndicator type Each policy includes a trigger with an event associated to the notification, as well as condition and action.

```
tosca_definitions_version: tosa_simple_yaml_1_3
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001

interface_types:
  MyCompany.nfv.interfaces.VnfIndicator:
    derived_from: tosa.interfaces.nfv.VnfIndicator
    notifications:
      utilization:
        description: this notification is used to received asynchronous
information of value change of the utilization_vnf_indicator
      health:
        description: this notification is used to received asynchronous
information of value change of the health_vnf_indicator

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosa.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: SunshineDB ]
        default: SunshineDB
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      flavour_id:
        type: string
```

```

        constraints: [ valid_values: [ simple ] ]
        default: simple
    flavour_description:
        type: string
        default: ""
    vnfm_info:
        type: list
        entry_schema:
            type: string
            constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
            default: [ '0:MyCompany-1.0.0' ]
    attributes:
        utilization_vnf_indicator:
            type: float
            description: holds the value of the utilization VNF indicator. It is
assigned the output value of the utilization notification.
        health_vnf_indicator:
            type: string
            description: holds the value of the health VNF indicator. It is assigned
the output value of the health notification.
        call_proc_scale_level:
            type: integer
        database_scale_level:
            type: integer
    interfaces:
        Vnflcm:
            operations:

#       scale: {}
        heal:
            inputs:
                additional_parameters:
                    type: MyCompany.datatypes.nfv.HealAdditionalParameters
                    required: false
        VnfIndicator:
            type: MyCompany.nfv.interfaces.VnfIndicator
            notifications:
                utilization:
                    outputs:
                        utilization_vnf_indicator: [ SELF, utilization_vnf_indicator ]
                health:
                    outputs:
                        health_vnf_indicator: [ SELF, health_vnf_indicator ]

data_types:
    MyCompany.datatypes.nfv.HealAdditionalParameters:
        derived_from: tosca.datatypes.nfv.VnfOperationAdditionalParameters
        properties:
            parameter_1:
                type: string
                required: false

policy_types:
    MyCompany.policies.nfv.VnfIndicator:
        derived_from: tosca.policies.nfv.VnfIndicator

topology_template:
    substitution_mappings:
        node_type: MyCompany.SunshineDB.1_0.1_0
#    ..

```

```

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    attributes:
      call_proc_scale_level: { get_attribute: [ SELF, scale_status, call_proc,
scale_level ] }
      database_scale_level: { get_attribute: [ SELF, scale_status, database,
scale_level ] }

  policies:
    - scaling_aspects:
        type: toska.policies.nfv.ScalingAspects
        properties:
          aspects:
            database:
              name: ..
              description: ..
              max_scale_level: 2
              step_deltas:
                - delta_1
            call_proc:
              name: ..
              description: ..
              max_scale_level: 4
              step_deltas:
                - delta_1
    - auto_scale:
        type: toska.policies.nfv.VnfIndicator
        properties:
          source: vnf
        triggers:
          scale_out:
            event: toska.interfaces.nfv.VnfIndicator.utilization # full name of a
notification in the VnfIndicator interface
            condition:
              - utilization_vnf_indicator: [ { greater_or_equal: 60.0 } ]
              - call_proc_scale_level: [ { less_than: 3 } ]
            action:
              - call_operation:
                  operation: Vnflcm.scale
                  inputs:
                    type:
                      value: scale_out
                    aspect:
                      value: call_proc
          scale_in:
            event: toska.interfaces.nfv.VnfIndicator.utilization # full name of a
notification in the VnfIndicator interface
            condition:
              - utilization_vnf_indicator: [ { less_or_equal: 20.0 } ]
              - call_proc_scale_level: [ { greater_than: 0 } ]
            action:
              - call_operation:
                  operation: Vnflcm.scale
                  inputs:
                    type:
                      value: scale_in
                    aspect:
                      value: call_proc

```

```

    targets: [SunshineDB ]

- auto_heal:
  type: toska.policies.nfv.VnfIndicator
  properties:
    source: vnf
  triggers:
    red:
      event: toska.interfaces.nfv.VnfIndicator.health # full name of a
notification in the VnfIndicator interface
      condition:
        health_vnf_indicator: [ { equal: red } ]
      action:
        - call_operation:
            operation: Vnflcm.heal
            inputs:
              cause:
                value: no_service
    yellow:
      event: toska.interfaces.nfv.VnfIndicator.health
      condition:
        health_vnf_indicator: [ { equal: yellow } ]
      action:
        - call_operation:
            operation: Vnflcm.heal
            inputs:
              cause:
                value: degraded_service
    targets: [SunshineDB ]

- auto_heal_with_additional_parameters:
  type: MyCompany.policies.nfv.VnfIndicator
  properties:
    source: vnf
  triggers:
    red_or_yellow:
      event: toska.interfaces.nfv.VnfIndicator.health # full name of a
notification in the VnfIndicator interface
      condition:
        or:
          - health_vnf_indicator: [ { equal: red } ]
          - health_vnf_indicator: [ { equal: yellow } ]
      action:
        - call_operation:
            operation: Vnflcm.heal
            inputs:
              cause:
                value: problem
                additional_parameters:
                  value:
                    parameter_1: value_1
    targets: [SunshineDB ]
#..

```

A.15.3 Auto-scale design with use of VNF indicator in NSD

This clause illustrates an example of a NS with two VNF indicators, one VNF indicator related to the utilization of the VNF_1 and one related to the utilization of the VNF_2.

The modelling and handling of the VNF indicator in the NSD service template involves the following definitions:

- A NS specific VNF indicator interface type definition that includes one notification for VNF indicators.
- New attribute definition included in the NS specific node type definition for each of the VNF indicators.
- Two attributes to hold the values of the current scale level of the 'call_proc' and 'database' scaling aspects.
- A NS specific auto_scale policy definitions of `tosca.policies.nfv.NsAutoScale` type, which includes a trigger with an event associated to the notification, as well as condition and action.

Example_NS.yaml:

```
tosca_definitions_version: toska_simple_yaml_1_3

description: Relational database, simple

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of NSD related TOSCA types as defined in ETSI
    GS NFV-SOL 001
  - example_VNF_1.yaml # uri of the yaml file which contains the definition of
    toska.nodes.nfv.example_VNF1, this file might be included in the NSD file structure
  - example_VNF_2.yaml # uri of the yaml file which contains the definition of
    toska.nodes.nfv.example_VNF2, this file might be included in the NSD file structure

data_types:
  MyCompany.datatypes.nfv.NsInstantiateAdditionalParameters:
    derived_from: toska.datatypes.nfv.NsOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
        required: true
        default: value_1
      parameter_2:
        type: string
        required: true
        default: value_2

interface_types:
  MyCompany.nfv.interfaces.NsVnfIndicator:
    derived_from: toska.interfaces.nfv.NsVnfIndicator
    notifications:
      utilization:
        description: this notification is used to received asynchronous information of
        value change of the utilization_vnf_indicator

node_types:
  toska.example_NS:
    derived_from: toska.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
    name:
      type: string
      constraints: [ equal: ExampleService ]
```



```

    default: ExampleService
  version:
    type: string
    constraints: [ equal: '1.0' ]
    default: '1.0'
  invariant_id:
    type: string
    constraints: [ equal: 1111-2222-aaaa-bbbb ]
    default: 1111-2222-aaaa-bbbb
  flavour_id:
    type: string
    constraints: [ equal: simple ]
    default: simple
  attributes:
    vnf_1_utilization_vnf_indicator:
      type: float
      description: holds the value of the vnf_1_utilization VNF indicator. It is
assigned the output value of the utilization notification.
    vnf_2_utilization_vnf_indicator:
      type: float
      description: holds the value of the vnf_1_utilization VNF indicator. It is
assigned the output value of the utilization notification.
    call_proc_scale_level:
      type: integer
    database_scale_level:
      type: integer
  interfaces:
    Nslcm:
      type: toska.interfaces.nfv.Nslcm
      operations:
        instantiate:
          inputs:
            additional_parameters:
              type: MyCompany.datatypes.nfv.NsInstantiateAdditionalParameters
              required: false
    NsVnfIndicator:
      type: MyCompany.nfv.interfaces.NsVnfIndicator
      notifications:
        utilization: {}

topology_template:
  substitution_mappings:
    node_type: toska.example_NS
  requirements:
    virtual_link: [ VNF_2, virtual_link_2 ] # the External connection point of
# VNF_2 is exposed as the Sap

node_templates:
  my_service:
    type: toska.example_NS
    properties:
    attributes:
      call_proc_scale_level: { get_attribute: [ SELF, scale_status, call_proc ] }
      vnf_1_utilization_vnf_indicator: { get_attribute: [ VNF_1,
utilization_vnf_indicator ] } # required Vnf indicator for VNF_1 of this NS
      vnf_2_utilization_vnf_indicator: { get_attribute: [ VNF_2,
utilization_vnf_indicator ] } # required Vnf indicator for VNF_1 of this NS

    interfaces:
      Nslcm:

```

```

    operations:
      instantiate:
        implementation: instantiate.workflow.yaml
      terminate:
        implementation: terminate.workflow.yaml

VNF_1:
  type: tosca.nodes.nfv.example_VNF1
  properties:
    # no property assignments needed for required properties that have a default
    value assigned in the node type definition, e.g. descriptor_id
    vnf_profile:
      instantiation_level: level_1
      min_number_of_instances: 2
      max_number_of_instances: 6
  requirements:
    - virtual_link: NsVirtualLink_1

VNF_2:
  type: tosca.nodes.nfv.example_VNF2
  properties:
    vnf_profile:
      instantiation_level: level_1
      min_number_of_instances: 1
      max_number_of_instances: 3

  requirements:
    - virtual_link_1: NsVirtualLink_1
    # - virtual_link_2: # map to virtual_link requirement of the NS node
    - dependency: VNF_1

NsVirtualLink_1:
  type: tosca.nodes.nfv.NsVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ipv4]
      flow_pattern: mesh
    vl_profile:
      max_bitrate_requirements:
        root: 1000
      min_bitrate_requirements:
        root: 1000

policies:
  - scaling_aspects:
      type: tosca.policies.nfv.NsScalingAspects
      properties:
        aspects:
          database:
            name: DatabaseScalingAspect
            description: This scaling aspect is for database
            ns_scale_levels:
              0:
                description: first NS level for the database scaling aspect
              1:
                description: second NS level for the database scaling aspect
          call_proc:
            name: ProcessorScalingAspect
            description: This scaling aspect is for Processor

```

```

        ns_scale_levels:
          0:
            description: first NS level for the processor scaling aspect
          1:
            description: second NS level for the processor scaling aspect
          2:
            description: third NS level for the processor scaling aspect

- auto_scale:
  type: toska.policies.nfv.NsAutoScale
#  properties:
#    # none
  triggers:
    scale_out:
      event: toska.interfaces.nfv.NsVnfIndicator.utilization # full name of a
notification in the VnfIndicator interface
      condition:
        - vnf_1_utilization_vnf_indicator: [ { greater_or_equal: 80.0 } ]
        - vnf_2_utilization_vnf_indicator: [ { greater_or_equal: 80.0 } ]
        - call_proc_scale_level: [ { less_than: 2 } ]
      action:
        - call_operation:
            operation: Nslcm.scale
            inputs:
              scale_ns_by_steps_data:
                scaling_direction: scale_out
                aspect: call_proc
      scale_in:
        event: toska.interfaces.nfv.NsVnfIndicator.utilization # full name of a
notification in the VnfIndicator interface
        condition:
          - vnf_1_utilization_vnf_indicator: [ { less_or_equal: 10.0 } ]
          - vnf_2_utilization_vnf_indicator: [ { less_or_equal: 10.0 } ]
          - call_proc_scale_level: [ { greater_than: 2 } ]
        action:
          - call_operation:
              operation: Nslcm.scale
              inputs:
                scale_ns_by_steps_data:
                  scaling_direction: scale_out
                  aspect: call_proc

  targets: [ my_service ]

```

The contents of example_VNF_1.yaml file with the VNF_1 node type definition are as follows:

```

tosca_definitions_version: toska_simple_yaml_1_3
description: VNFD for VNF_1

imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI GS
NFV-SOL 001

interface_types:
  MyCompany.nfv.interfaces.VnfIndicator:
    derived_from: toska.interfaces.nfv.VnfIndicator
    notifications:
      utilization:
        description: this notification is used to received asynchronous information of

```

```

value change of the utilization_vnf_indicator
  health:
    description: this notification is used to received asynchronous information of
value change of the health_vnf_indicator

node_types:
  tosca.nodes.nfv.example_VNF1:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: SunshineDB ]
        default: SunshineDB
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      flavour_id:
        type: string
        constraints: [ valid_values: [ simple ] ]
        default: simple
      flavour_description:
        type: string
        default: ""
      vnfm_info:
        type: list
        entry_schema:
          type: string
          constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
          default: [ '0:MyCompany-1.0.0' ]
    attributes:
      utilization_vnf_indicator:
        type: float
        description: holds the value of the utilization VNF indicator. It is assigned
the output value of the utilization notification.
      health_vnf_indicator:
        type: string
        description: holds the value of the health VNF indicator. It is assigned the
output value of the health notification.
      call_proc_scale_level:
        type: integer
      database_scale_level:
        type: integer
    interfaces:
      Vnflcm:
        operations:
          scale: {}
      VnfIndicator:
        type: MyCompany.nfv.interfaces.VnfIndicator

```

```

    notifications:
      utilization:
        outputs:
          utilization_vnf_indicator: [ SELF, utilization_vnf_indicator ]
      health:
        outputs:
          health_vnf_indicator: [ SELF, health_vnf_indicator ]

```

The contents of example_VNF_2.yaml file with the VNF_2 node type definition are as follows:

```

tosca_definitions_version: tosca_simple_yaml_1_3
description: VNFD for VNF_2

imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI GS
    NFV-SOL 001

interface_types:
  MyOtherCompany.nfv.interfaces.VnfIndicator:
    derived_from: tosca.interfaces.nfv.VnfIndicator
    notifications:
      utilization:
        description: this notification is used to received asynchronous information of
value change of the utilization_vnf_indicator
      health:
        description: this notification is used to received asynchronous information of
value change of the health_vnf_indicator

node_types:
  tosca.nodes.nfv.example_VNF2:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a2233 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a2233
      provider:
        type: string
        constraints: [ equal: MyOtherCompany ]
        default: MyOtherCompany
      product_name:
        type: string
        constraints: [ equal: MyProduct ]
        default: MyProduct
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      flavour_id:
        type: string
        constraints: [ valid_values: [ simple ] ]
        default: simple
      flavour_description:
        type: string
        default: ""

```

```

vnfm_info:
  type: list
  entry_schema:
    type: string
    constraints: [ valid_values: [ [ '0:MyCompany-1.0.0' ] ] ]
    default: [ '0:MyCompany-1.0.0' ]
  attributes:
    utilization_vnf_indicator:
      type: float
      description: holds the value of the utilization VNF indicator. It is assigned
the output value of the utilization notification.
    health_vnf_indicator:
      type: string
      description: holds the value of the health VNF indicator. It is assigned the
output value of the health notification.
    call_proc_scale_level:
      type: integer
    database_scale_level:
      type: integer
  interfaces:
    Vnflcm:
      operations:
        scale: {}
    VnfIndicator:
      type: MyOtherCompany.nfv.interfaces.VnfIndicator
      notifications:
        utilization:
          outputs:
            utilization_vnf_indicator: [ SELF, utilization_vnf_indicator ]
        health:
          outputs:
            health_vnf_indicator: [ SELF, health_vnf_indicator ]
  requirements:
    - virtual_link:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 0 ]
    - virtual_link_1:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 1 ]
    - virtual_link_2:
        capability: tosca.capabilities.nfv.VirtualLinkable
        relationship: tosca.relationships.nfv.VirtualLinksTo
        occurrences: [ 0, 1 ]

```

A.16 VDU connection point in trunk mode

With supporting trunk mode, a VM can differentiate between traffic of many networks by different encapsulation types and IDs, instead of using many vNICs. Connection points of a VDU are combined into one VIF by turning one connection point into a trunk parent port and the other connection points into trunk subports of the same trunk. A trunk logical model is a collection of connection points. Different infrastructure technologies may have different way to implement trunk logical model.

The main concept of difference between subport and parent port:

- The parent port can associate to the VM and a network.
- The parent port can be associated by subports.

- The subport cannot associate directly to the VM but through the parent port and a network.
- The traffic of sub-port can be encapsulated as required.

In the following example, there are three connection points are needed in initial deployment stage, one VdpCp node taken as trunk port and two VduSubCp nodes taken as trunk subports. With respect to VduSubCp node, requirement virtual_binding, if existing, is useless because the trunk_binding have priority over it.

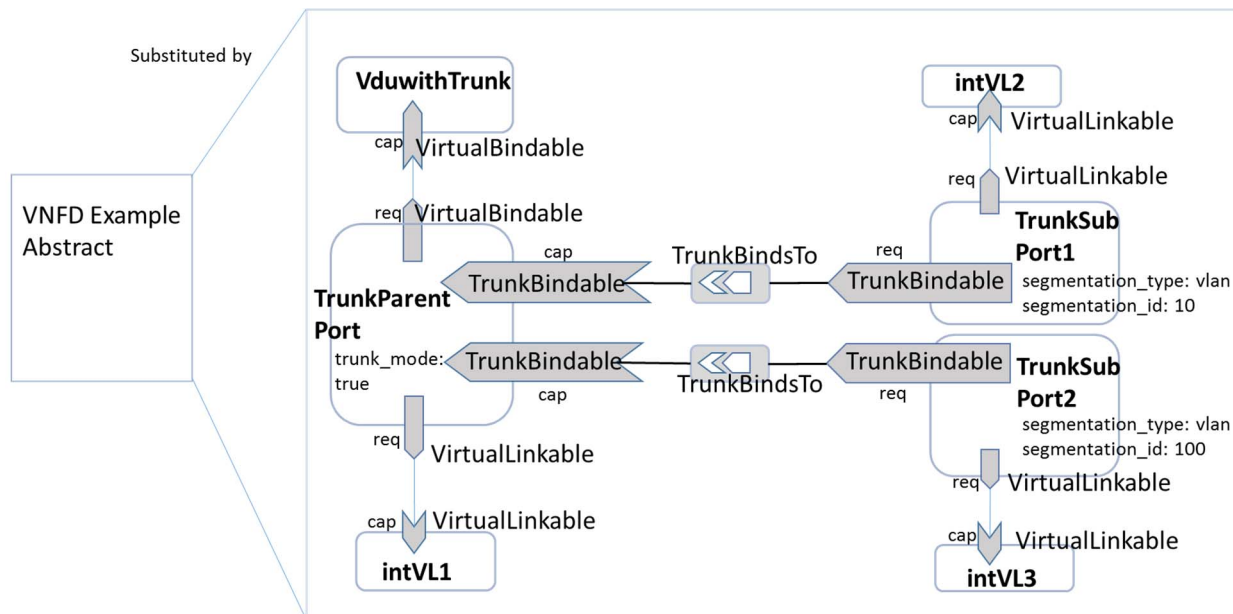


Figure A.16-1: VNFD with a VDU has three connection points group as trunk

The following service template shows the relevant parts of the TOSCA VNFD corresponding to figure A.16-1:

```
tosca_definitions_version: tosca_simple_yaml_1_3
# ...
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001

topology_template:
  node_templates:
    #...
    VduwithTrunk:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: VduwithTrunk
        description: compute node with trunk port
        vdu_profile:
          min_number_of_instances: 3
          max_number_of_instances: 4
      capabilities:
        virtual_compute:
          properties:
            virtual_memory:
              virtual_mem_size: 8192 MiB
            virtual_cpu:
              num_virtual_cpu: 2
# omitted for brevity

    TrunkParentPort:
      type: tosca.nodes.nfv.VduCp
      properties:
```

```

    layer_protocols: [ ipv4 ]
    trunk_mode: true
    # omitted for brevity
    requirements:
      - virtual_binding: VduwithTrunk
      - virtual_link: intVL1

TrunkSubPort1:
  type: toasca.nodes.nfv.VduSubCp
  properties:
    # omitted for brevity
    layer_protocols: [ ipv4 ]
    segmentation_type: vlan
    segmentation_id: 10
  requirements:
    - trunk_binding: TrunkParentPort
    - virtual_link: intVL2

TrunkSubPort2:
  type: toasca.nodes.nfv.VduSubCp
  properties:
    # omitted for brevity
    layer_protocols: [ ipv4 ]
    segmentation_type: vlan
    segmentation_id: 100
  requirements:
    - trunk_binding: TrunkParentPort
    - virtual_link: intVL3

intVL1:
  type: toasca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
    vl_profile:
      max_bitrate_requirements:
        root: 100000
        leaf: 20000
      min_bitrate_requirements:
        root: 10000
        leaf: 10000 # omitted for brevity

intVL2:
  type: toasca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
    vl_profile:
      max_bitrate_requirements:
        root: 100000
        leaf: 20000
      min_bitrate_requirements:
        root: 10000
        leaf: 10000
    # omitted for brevity

intVL3:
  type: toasca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
    vl_profile:
      max_bitrate_requirements:

```



```

    root: 100000
    leaf: 20000
    min_bitrate_requirements:
      root: 10000
      leaf: 10000
    # omitted for brevity

```

A.17 NS scaling

Clause A.17 shows the use of the NS policies to define the scaling aspects and NS scale levels.

The NSD NS_1 consists of the following components:

- Three VNFDs specified with the VNF node templates: FrontEnd, CentralProcess and Database
- One nested NSD specified with the NS node template: SupportFrontEnd
- One NsVirtualLinkDesc specified with the NS VL node template: NS_VL_1

Figure A.17-1 shows the two scaling aspects and the two instantiation levels defined in this NS.

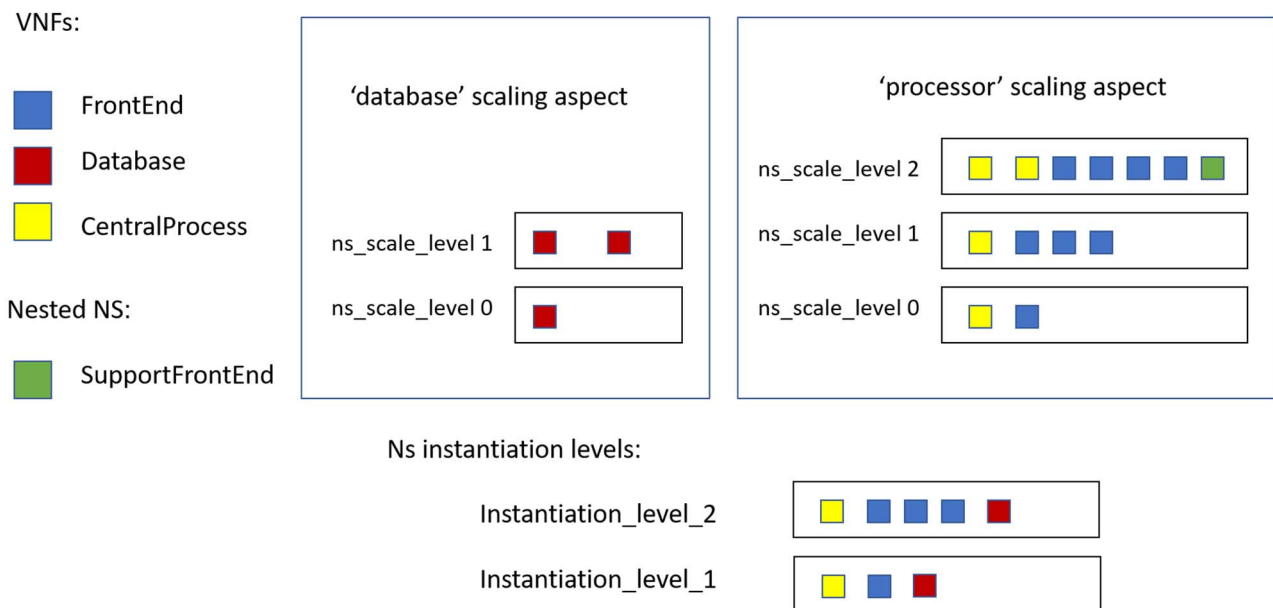


Figure A.17-1: NS scaling aspects and instantiation levels

```

tosca_definitions_version: tosca_simple_yaml_1_3
description: myExampleNs with scaling aspects
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in ETSI
    GS NFV-SOL 001
  - MyScalableNs_Type.yaml # contains the NS node type definition.
  - SupportFrontEndNs.yaml # uri of the yaml file which contains the
    MyProviderCompany.NS.SupportFrontEndNs node type definition, this file might be
    included in the NSD file structure of NS_1.
  - FrontEnd_1_0.yaml # uri of the yaml file which contains the
    MyCompany.VNF.FrontEnd_1_0 node type definition, this file might be included in the NSD
    file structure of NS_1.

```

```

- Database_1_0.yaml # uri of the yaml file which contains the
MyCompany.VNF.Database_1_0 node type definition, this file might be included in the NSD
file structure of NS_1.
- CentralProcess_1_0.yaml # uri of the yaml file which contains the
MyCompany.VNF.CentralProcess_1_0 node type definition, this file might be included in
the NSD file structure of NS_1.

topology_template:
  substitution_mappings:
    node_type: MyProviderCompany.NS.MyScalableNS
  substitution_filter:
    properties:
      - flavour_id: { equal: small }
  requirements:
    virtual_link: [ FrontEnd, virtual_link_2 ]

node_templates:
  NS_1:
    type: MyProviderCompany.NS.MyScalableNS
    interfaces:
      Nslcm:
        operations:
          instantiate:
            implementation: instantiate.workflow.yaml
          terminate:
            implementation: terminate.workflow.yaml

  FrontEnd:
    type: MyCompany.VNF.FrontEnd_1_0
    properties:
      # no property assignments needed for required properties that have a default
value assigned in the node type definition, e.g. descriptor_id
      flavour_id: simple
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 4
    requirements:
      - virtual_link_1: NS_VL_1
      # - virtual_link_2: # map to virtual_link requirement of the NS node

  Database:
    type: MyCompany.VNF.Database_1_0
    properties:
      # no property assignments needed for required properties that have a default
value assigned in the node type definition, e.g. descriptor_id
      flavour_id: simple
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 2
    requirements:
      - virtual_link_1: NS_VL_1

  CentralProcess:
    type: MyCompany.VNF.CentralProcess_1_0
    properties:
      # no property assignments needed for required properties that have a default
value assigned in the node type definition, e.g. descriptor_id
      flavour_id: simple
      vnf_profile:

```

```

    instantiation_level: level_1
    min_number_of_instances: 1
    max_number_of_instances: 2
  requirements:
    - virtual_link_1: NS_VL_1

```

```

SupportFrontEnd:
  type: MyProviderCompany.NS.SupportFrontEndNs
  properties:
    descriptor_id: c1bb0ab8-deab-4fa7-95ed-4840d70a3574
    designer: MyCompany
    version: 1.0.0.0
    name: myExample2Service
    invariant_id: aaaa-bbbb-cccc-dddd
    ns_profile:
      ns_instantiation_level: level_1
      min_number_of_instances: 0
      max_number_of_instances: 1
  requirements:
    - virtual_link_1: NS_VL_1

```

```

NS_VL_1:
  type: toska.nodes.nfv.NsVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ipv4]
      flow_pattern: mesh
    vl_profile:
      max_bitrate_requirements:
        root: 1000
      min_bitrate_requirements:
        root: 1000

```

```

policies:
  - scaling_aspects:
      type: toska.policies.nfv.NsScalingAspects
      properties:
        aspects:
          database:
            name: DatabaseScalingAspect
            description: This scaling aspect is used to scale the Database VNF
            ns_scale_levels:
              0:
                description: first NS level for the database scaling aspect
              1:
                description: second NS level for the database scaling aspect
          processor:
            name: ProcessorScalingAspect
            description: This scaling aspect is used to scale the FrontEnd and
CentralProcess VNFs and the SupportFrontEnd nested NS
            ns_scale_levels:
              0:
                description: first NS level for the processor scaling aspect
              1:
                description: second NS level for the processor scaling aspect
              2:
                description: second NS level for the processor scaling aspect
        - FrontEndVnfToLevelMapping:
            type: toska.policies.nfv.VnfToLevelMapping
            properties:

```

```

    aspect: processor
    number_of_instances:
      0: 1
      1: 3
      2: 4
    targets: [ FrontEnd ]

- CentralProcessVnfToLevelMapping:
  type: tosca.policies.nfv.VnfToLevelMapping
  properties:
    aspect: processor
    number_of_instances:
      0: 1
      1: 1
      2: 2
    targets: [ CentralProcess ]

- DatabaseVnfToLevelMapping:
  type: tosca.policies.nfv.VnfToLevelMapping
  properties:
    aspect: database
    number_of_instances:
      0: 1
      1: 2
    targets: [ Database ]

- SupportFrontEndNsToLevelMapping:
  type: tosca.policies.nfv.NsToLevelMapping
  properties:
    aspect: processor
    number_of_instances:
      0: 0
      1: 0
      2: 1
    targets: [ SupportFrontEnd ]

- ns_instantiation_levels:
  type: tosca.policies.nfv.NsInstantiationLevels
  properties:
    ns_levels:
      instantiation_level_1:
        description: low capacity instantiation level
      instantiation_level_2:
        description: high capacity instantiation level
      default_level: instantiation_level_1

- FrontEnd_instantiation_levels:
  type: tosca.policies.nfv.VnfToInstantiationLevelMapping
  properties:
    number_of_instances:
      instantiation_level_1: 1
      instantiation_level_2: 3
    targets: [ FrontEnd ]

- CentralProcess_instantiation_levels:
  type: tosca.policies.nfv.VnfToInstantiationLevelMapping
  properties:
    number_of_instances:
      instantiation_level_1: 1
      instantiation_level_2: 1

```

```

    targets: [ CentralProcess ]

- Database_instantiation_levels:
  type: tosa.policies.nfv.VnfToInstantiationLevelMapping
  properties:
    number_of_instances:
      instantiation_level_1: 1
      instantiation_level_2: 1
  targets: [ Database ]

- SupportFrontEnd_instantiation_levels:
  type: tosa.policies.nfv.NsToInstantiationLevelMapping
  properties:
    number_of_instances:
      instantiation_level_1: 0
      instantiation_level_2: 0
  targets: [ SupportFrontEnd ]

```

The contents of MyScalableNs_Type.yaml file with the node type definition are as follows:

```

tosca_definitions_version: tosa_simple_yaml_1_3

description: type definition of tosa.MyScalableNS

imports:
- etsi_nfv_sol001_nsd_types.yaml # all of TOSCA types as defined in ETSI
  GS NFV-SOL 001

node_types:
  MyProviderCompany.NS.MyScalableNS:
    derived_from: tosa.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ valid_values: [ b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ] ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer:
        type: string
        constraints: [ valid_values: [ MyCompany ] ]
        default: MyCompany
      name:
        type: string
        constraints: [ valid_values: [ ExampleService ] ]
        default: ExampleService
      version:
        type: string
        constraints: [ valid_values: [ '1.0' ] ]
        default: '1.0'
      invariant_id:
        type: string
        constraints: [ valid_values: [ 1111-2222-aaaa-bbbb ] ]
        default: 1111-2222-aaaa-bbbb
      flavour_id:
        type: string
        constraints: [ valid_values: [ small, big ] ]
        default: small
    requirements:
      - virtual_link:
          capability: tosa.capabilities.nfv.VirtualLinkable
    interfaces:

```

```

Nslcm:
  type: tosca.interfaces.nfv.Nslcm

```

A.18 VNFD illustrating OsContainer modelling example

The following TOSCA service template provides an VNFD example for a containerized VNF. In this example the VNF comprises two kinds of VNFCs represented by two `Vdu.OsContainerDeployableUnit` nodes. One kind of the VNFC (`vdu_1`) contains a single container while the second one (`vdu_2`) contains two containers and has storage resources. Each VNFC has an associated connection point (`VduCp`) connected to a VNF virtual link intended to be deployed as a secondary container cluster internal or external network. The service provided by the first VNFC is exposed using a virtual Cp. Each VNFC has also an additional associated connection point (`VduCp`) to be connected to an external NS virtual link for external management purposes. The logical structure of the VNF is illustrated in figure A.18-1.

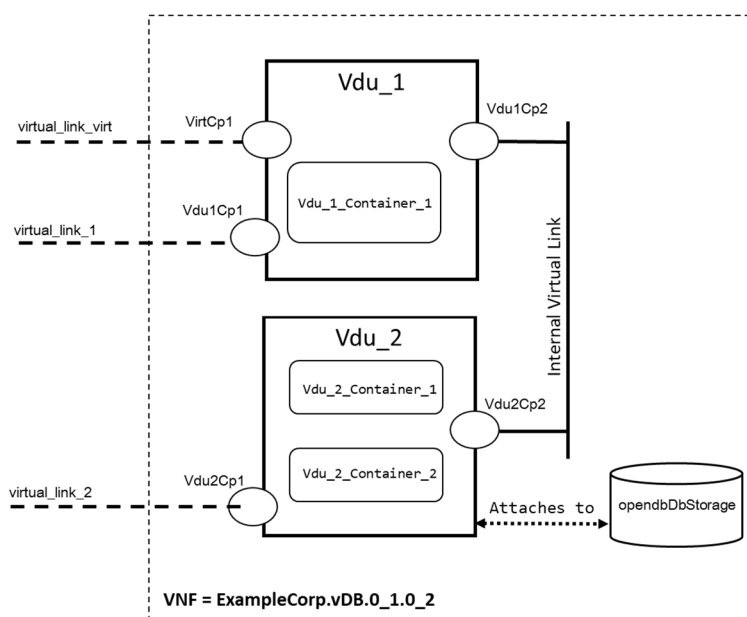


Figure A.18-1: Containerized VNF example

```

tosca_definitions_version: tosca_simple_yaml_1_3
description: A sample VNFD descriptor with containerized VDUs
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # VNFD type definitions

data_types:
  Example.datatypes.nfv.VnfAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfAdditionalConfigurableProperties
    properties:
      name_prefix_in_cism:
        type: string
        required: false
      dns_server:
        type: list
        entry_schema:
          type: string
        required: false
  Example.datatypes.nfv.VnfConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfConfigurableProperties
    properties:
      additional_configurable_properties:
        type: Example.datatypes.nfv.VnfAdditionalConfigurableProperties

```

```

node_types:
  ExampleCorp.vDB.0_1.0_2:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        default: 5af09567-fc5f-4be9-b372-0cc431ad5c03
        constraints: [ equal: 5af09567-fc5f-4be9-b372-0cc431ad5c03 ]
      descriptor_version:
        default: "0.2"
        constraints: [ equal: "0.2" ]
      provider:
        default: "ExampleCorp"
        constraints: [ equal: "ExampleCorp" ]
      product_name:
        default: "vDB"
        constraints: [ equal: "vDB" ]
      software_version:
        default: "0.1"
        constraints: [ equal: "0.1" ]
      vnfm_info:
        type: list
        required: true
        description: Identifies VNFM(s) compatible with the VNF
        entry_schema:
          type: string
          constraints: [ valid_values: [ [ 'etsivnfm:v0.1.0' ] ] ]
          default: [ 'etsivnfm:v0.1.0' ]
      flavour_id:
        default: simple
        constraints: [ valid_values: [ simple, complex ] ]
      flavour_description:
        default: ""
      configurable_properties:
        type: Example.datatypes.nfv.VnfConfigurableProperties
        description: Describes the configurable properties of the VNF
        required: false
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [0, 0]
      - virtual_link_1:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [0, 1]
      - virtual_link_2:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [0, 1]
      - virtual_link_virt:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [0, 1]

topology_template:
  substitution_mappings:
    node_type: ExampleCorp.vDB.0_1.0_2
  requirements:
    virtual_link_1: [ vdu1Cp1, virtual_link ]
    virtual_link_2: [ vdu2Cp1, virtual_link ]

```

```

    virtual_link_virt: [ VirtCp1, virtual_link ]

node_templates:
  VNF:
    type: ExampleCorp.vDB.0_1.0_2
    properties:
      product_info_name: "Firewall"
      product_info_description: "Firewall with new functionality"
      configurable_properties: # All of the VNF configurable properties should
go here
        is_autoscale_enabled: true
        is_autoheal_enabled: true
        additional_configurable_properties:
          name_prefix_in_cism: "ExampleContainerizedVNF" # Default
          dns_server: [ "8.8.8.8", "8.8.4.4" ] #Default

        # modifiable_attributes: {} # All of the VNF modifiable properties
should go here
      flavour_id: simple
      flavour_description: "A specimen copy of the VNF Descriptor for containers"
    interfaces:
      Vnflcm:
        operations:
          # instantiate_start:
          # instantiate: # A sample illustration to run an instantiation script
without any inputs.
          instantiate_end: helm_test

          scale: #Invoked upon receipt of a Scale VNF request
          inputs:
            type: "scale_out"
            aspect: "default"

    artifacts:
      helm_test:
        description: Post instantiation test script
        type: toska.artifacts.Implementation.Bash
        file: Artifacts/Tests/helm_test.sh

opendb_mciop:
  type: toska.nodes.nfv.Mciop
  requirements:
    - associatedVdu: Vdu_2
  artifacts:
    opendb_helm:
      description: Helm Chart for opendb Pod
      type: toska.artifacts.nfv.HelmChart
      file: Artifacts/Scripts/opendb.tgz
    opendb_mapping_script:
      description: executable artifact to be invoked for any operation towards Helm
related to this Mciop, except for the Query VNF operation.
      type: toska.artifacts.nfv.HelmMappingScript
      file: Artifacts/Scripts/ExampleCorp_vDB_0_1_0_2.sh # In this example the same
mapping script is used for both Mciops.
    properties:
      language: bash
    opendb_mapping_rule:
      description: contains mapping rules specific to this Mciop to be interpreted
by the opendb_mapping_script artifact.
      type: toska.artifacts.nfv.HelmMappingRule

```



```

    file: Artifacts/opendb_mapping_rules.txt

lb_mciop:
  type: toska.nodes.nfv.Mciop
  requirements:
    - associatedVdu: Vdu_1
    - dependency: opendb_mciop
  artifacts:
    lb_helm:
      description: Helm Chart for lb Pod
      type: toska.artifacts.nfv.HelmChart
      file: Artifacts/Scripts/lb.tgz
    lb_mapping_script:
      description: executable artifact to be invoked for any operation towards Helm
related to this Mciop, except for the Query VNF operation.
      type: toska.artifacts.nfv.HelmMappingScript
      file: Artifacts/Scripts/ExampleCorp_vDB_0_1_0_2.sh # In this example the same
mapping script is used for both Mciops.
      properties:
        language: bash
    lb_mapping_rule:
      description: contains mapping rules specific to this Mciop to be interpreted
by the lb_mapping_script artifact.
      type: toska.artifacts.nfv.HelmMappingRule
      file: Artifacts/lb_mapping_rules.txt

vdu1Cp1:
  type: toska.nodes.nfv.VduCp
  description: "External CP for managing the VNFC"
  properties:
    layer_protocols: [ ethernet ]
    vnic_type: normal
    order: 1
  requirements:
    - virtual_binding: Vdu_1
    # - virtual_link: the target node is determined in the NSD

vdu1Cp2:
  type: toska.nodes.nfv.VduCp
  description: "CP for intra-VNF communication"
  properties:
    layer_protocols: [ ethernet ]
    vnic_type: macvlan
    bitrate_requirement: 10000000000 #10 GiB
    order: 2
    trunk_mode: true # vdu1Cp2 is capable of carrying traffic for several VLANs.
  requirements:
    - virtual_binding: Vdu_1
    - virtual_link: InternalV1

VirtCp1:
  type: toska.nodes.nfv.VirtualCp
  properties:
    layer_protocols: [ ipv6 ]
    protocol:
      - address_data:
          - address_type: ip_address
            l3_address_data:
              floating_ip_activated: true
              ip_address_assignment: false
              ip_address_type: ipv6

```

```

        number_of_ip_address: 1
        associated_layer_protocol: ipv6
    additionalServiceData:
        - portData:
            - name: DBAccessService
              protocol: tcp
              port: 8001
              portConfigurable: false
            - name: DBConfigService
              protocol: tcp
              port: 443
              portConfigurable: false
    requirements:
        - target: Vdu_1
    # - virtual_link: the target node is determined in the NSD

Vdu_1:
    type: toasca.nodes.nfv.Vdu.OsContainerDeployableUnit
    properties:
        name: "lb"
        description: "Advanced Software Load Balancer"
        vdu_profile:
            min_number_of_instances: 1
            max_number_of_instances: 4
        mcio_identification_data:
            name: lb_mcio
            type: Deployment
    requirements:
        - container: Vdu_1_Container_1

Vdu_1_Container_1:
    type: toasca.nodes.nfv.Vdu.OsContainer
    properties:
        name: "lb Container"
        description: "Advanced Software Load Balancer"
        requested_cpu_resources: 100 # In Milli-Cpus, ie 0.1 CPU
        cpu_resource_limit: 1000 # In Milli-Cpus, ie 1 CPU, single threaded
        requested_memory_resources: 10 MiB
        memory_resource_limit: 100 MiB
    artifacts:
        sw_image:
            type: toasca.artifacts.nfv.SwImage
            file: Artifacts/Images/lbimage
            properties:
                name: lb
                version: "7.3"
                checksum:
                    algorithm: sha-256
                    hash: a411cafee2f0f702572369da0b765e2
                container_format: docker
                size: "1024MB"

vdu2Cp1:
    type: toasca.nodes.nfv.VduCp
    description: "External CP for managing the VNFC"
    properties:
        layer_protocols: [ ethernet ]
        vnic_type: normal
    requirements:
        - virtual_binding: Vdu_2

```

```
# - virtual_link: the target node is determined in the NSD

vdu2Cp2:
  type: toska.nodes.nfv.VduCp
  description: "CP for intra-VNF communication"
  properties:
    layer_protocols: [ ethernet ]
    vnic_type: macvlan
    bitrate_requirement: 10000000000 #10 GiB
    order: 2
    trunk_mode: true # vdu2Cp2 is capable of carrying traffic for several VLANs.
  requirements:
    - virtual_binding: Vdu_2
    - virtual_link: InternalV1

opendbDbStorage:
  type: toska.nodes.nfv.Vdu.VirtualBlockStorage
  properties:
    virtual_block_storage_data:
      size_of_storage: 100 GB
      rdma_enabled: false

Vdu_2:
  type: toska.nodes.nfv.Vdu.OsContainerDeployableUnit
  properties:
    name: "opendb"
    description: "DB Server"
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 1
    mcio_identification_data:
      name: opendb_mcio
      type: StatefulSet
  requirements:
    - container: Vdu_2_Container_1
    - container: Vdu_2_Container_2
    - virtual_storage: opendbDbStorage

Vdu_2_Container_1:
  type: toska.nodes.nfv.Vdu.OsContainer
  properties:
    name: "opendb FE Container"
    description: "DB Server Container"
    requested_cpu_resources: 1000 # In Milli-Cpus, ie 1 CPU
    cpu_resource_limit: 2500 # In Milli-Cpus, ie 2.5 CPU
    requested_memory_resources: 1 GiB
    memory_resource_limit: 10 GiB
  artifacts:
    sw_image:
      type: toska.artifacts.nfv.SwImage
      file: Artifacts/Images/opendbFEimage
      properties:
        name: opendb
        version: "4.3"
        checksum:
          algorithm: sha-256
          hash: a411cafee2f0f702572369da0b765e2
        container_format: docker
        size: "1024MB"
```

```

Vdu_2_Container_2:
  type: toska.nodes.nfv.Vdu.OsContainer
  properties:
    name: "opendb Container"
    description: "DB Server Container"
    requested_cpu_resources: 1000 # In Milli-Cpus, ie 1 CPU
    cpu_resource_limit: 2500 # In Milli-Cpus, ie 2.5 CPU
    requested_memory_resources: 1 GiB
    memory_resource_limit: 10 GiB
  artifacts:
    sw_image:
      type: toska.artifacts.nfv.SwImage
      file: Artifacts/Images/opendbBEimage
      properties:
        name: opendb
        version: "4.3"
        checksum:
          algorithm: sha-256
          hash: a411cafee2f0f702572369da0b765e2
        container_format: docker
        size: "1024MB"

```

```

InternalVl:
  type: toska.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ethernet ]
    description: Internal VL # internal virtual link meant to be implemented as a
secondary container cluster internal or external network
    vl_profile:
      max_bitrate_requirements: #in bits per second
        root: 100000
        leaf: 20000
      min_bitrate_requirements: #in bits per second
        root: 10000
        leaf: 10000

```

A.19 NSD with data flow mirroring information

This clause shows the use of data flow mirroring policies in a NSD service template.

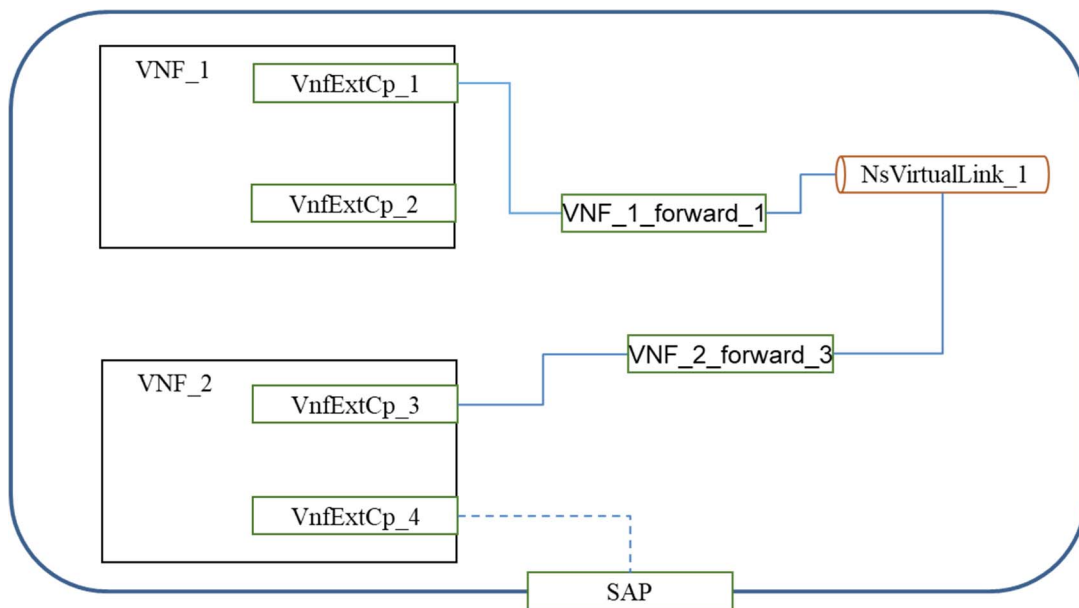


Figure A.19-1: A NSD Example with 2 VNFs

```

tosca_definitions_version: tosa_simple_yaml_1_3
description: NSD example with data flow mirroring information
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of NSD related TOSCA types as defined in ETSI
    GS NFV-SOL 001
  - example_vnf1.yaml # uri of the yaml file which contains the
    tosa.nodes.nfv.example_VNF1 node type definition, this file might be included in the
    NSD file structure. See the definition in clause A.14.
  - example_vnf2.yaml # uri of the yaml file which contains the
    tosa.nodes.nfv.example_VNF2 node type definition, this file might be included in the
    NSD file structure. See the definition in clause A.14.

```

```

node_types:
  tosa.example_NS:
    derived_from: tosa.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
    name:
      type: string
      constraints: [ equal: ExampleService ]
      default: ExampleService
    version:
      type: string
      constraints: [ equal: '1.0' ]
      default: '1.0'
    invariant_id:
      type: string
      constraints: [ equal: 1111-2222-aaaa-bbbb ]
      default: 1111-2222-aaaa-bbbb
    flavour_id:

```

```

    type: string
    constraints: [ equal: simple ]
    default: simple
requirements:
  - virtual_link:
      capability: tosca.capabilities.nfv.VirtualLinkable
      relationship: tosca.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 0 ]
  - virtual_link_1:
      capability: tosca.capabilities.nfv.VirtualLinkable
      relationship: tosca.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 1 ]

topology_template:
  substitution_mappings:
    node_type: tosca.example_NS
  requirements:
    virtual_link_1: [ VNF_2, virtual_link_4 ] # the requirement of SAP of the NS

node_templates:
  my_service:
    type: tosca.example_NS
    properties:
      descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer: MyCompany
      name: ExampleService
      # . . .
    interfaces:
      Nslcm:
        operations:
          instantiate:
            implementation: instantiate.workflow.yaml
          terminate:
            implementation: terminate.workflow.yaml

  NsVirtualLink_1:
    type: tosca.nodes.nfv.NsVirtualLink
    properties:
      vl_profile:
        min_bitrate_requirements:
          root: 100000
        max_bitrate_requirements:
          root: 200000
        data_flow_mirroring:
          - mirroring_job_name: myNs_mirroring_Job_1
            description: mirroring job 1 of my NS
      connectivity_type:
        layer_protocols: [ ipv4 ]
      # . . .

  VNF_1:
    type: tosca.nodes.nfv.example_VNF1
    properties:
      descriptor_id: b1bb0ce7-2222-4fa7-95ed-4840d70a1179
      descriptor_version: "1.0"
      # . . .
    requirements:
      - virtual_link: VNF_1_forward_1 # related to VnfExtCp_1 in VNF_1

  VNF_1_forward_1:
    type: tosca.nodes.nfv.Forwarding

```

```

requirements:
  - virtual_link: NsVirtualLink_1

VNF_2:
  type: toska.nodes.nfv.example_VNF2
  properties:
    descriptor_id: b1bb0ce7-2222-4fa7-95ed-4840d70a1182
    descriptor_version: "1.0.0"
    # . . .
  requirements:
    - virtual_link: VNF_2_forward_3 # related to VnfExtCp_3 in VNF_2
    # - virtual_link_4: # map to virtual_link requirement of the NS node

VNF_2_forward_3:
  type: toska.nodes.nfv.Forwarding
  requirements:
    - virtual_link: NsVirtualLink_1

policies:

- DataFlow_a:
  type: toska.policies.nfv.DataFlowInfo
  properties:
    data_flow_info_id: myNs_flow_a
    associated_mirroring_job_name: myNs_mirroring_Job_1
    direction: in
    targets: [ VNF_1_forward_1 ]

- DataFlow_b:
  type: toska.policies.nfv.DataFlowInfo
  properties:
    data_flow_info_id: myNs_flow_b
    associated_mirroring_job_name: myNs_mirroring_Job_1
    direction: in
    targets: [ VNF_2_forward_3 ]

```

A.20 NSD example with l2_network scope used in NsAffinityRule/NsAntiAffinityRule

This clause shows the use of NsAffinityRule/NsAntiAffinityRule policies with l2_network scope in a NSD service template.

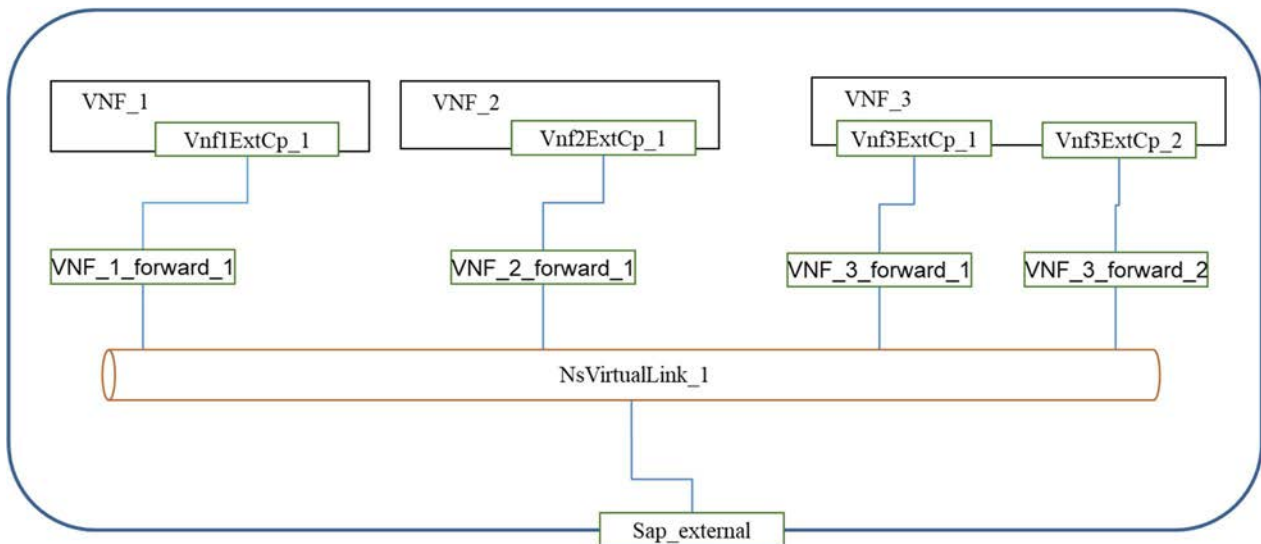


Figure A.20-1: A NSD Example with 3 VNFs

```

tosca_definitions_version: tosca_simple_yaml_1_3
description: NSD example with l2_network scope used in
NsAffinityRule/NsAntiAffinityRule
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of NSD related TOSCA types as defined in ETSI
    GS NFV-SOL 001
  - example_vnf1.yaml # uri of the yaml file which contains the
    toska.nodes.nfv.example_VNF1 node type definition, this file might be included in the
    NSD file structure. See the definition in clause A.14.
  - example_vnf2.yaml # uri of the yaml file which contains the
    toska.nodes.nfv.example_VNF2 node type definition, this file might be included in the
    NSD file structure. See the definition in clause A.14.
  - example_vnf3.yaml # uri of the yaml file which contains the
    toska.nodes.nfv.example_VNF3 node type definition, this file might be included in the
    NSD file structure. See the definition in clause A.14.

node_types:
  myCompany.example_NS:
    derived_from: toska.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1199 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1199
      designer:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      name:
        type: string
        constraints: [ equal: ExampleService ]
        default: ExampleService
      version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      invariant_id:
        type: string
        constraints: [ equal: 1111-2222-aaaa-cccc ]

```



```

    default: 1111-2222-aaaa-cccc
    flavour_id:
      type: string
      constraints: [ equal: simple ]
      default: simple
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 0 ]
      - virtual_link_1:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 1 ]

topology_template:
  substitution_mappings:
    node_type: myCompany.example_NS
  requirements:
    virtual_link_1: [ Sap_external, external_virtual_link ] # the requirement of SAP
of the NS

node_templates:
  my_service:
    type: myCompany.example_NS
    interfaces:
      Nslcm:
        operations:
          instantiate:
            implementation: instantiate.workflow.yaml
          terminate:
            implementation: terminate.workflow.yaml

VNF_1:
  type: tosca.nodes.nfv.example_VNF1
  requirements:
    - virtual_link: VNF_1_forward_1 # related to Vnf1ExtCp_1 in VNF_1

VNF_1_forward_1: # represents Vnf1ExtCp_1
  type: tosca.nodes.nfv.Forwarding
  requirements:
    - virtual_link: NsVirtualLink_1

VNF_2:
  type: tosca.nodes.nfv.example_VNF2
  requirements:
    - virtual_link: VNF_2_forward_1 # related to Vnf2ExtCp_1 in VNF_2

VNF_2_forward_1: # represents Vnf2ExtCp_1
  type: tosca.nodes.nfv.Forwarding
  requirements:
    - virtual_link: NsVirtualLink_1

VNF_3:
  type: tosca.nodes.nfv.example_VNF3
  requirements:
    - virtual_link: VNF_3_forward_1 # related to Vnf3ExtCp_1 in VNF_3
    - virtual_link_3: VNF_3_forward_2 # related to Vnf3ExtCp_2 in VNF_3

VNF_3_forward_1: # represents Vnf3ExtCp_1
  type: tosca.nodes.nfv.Forwarding

```

```

requirements:
  - virtual_link: NsVirtualLink_1

VNF_3_forward_2: # represents Vnf3ExtCp_2
type: toska.nodes.nfv.Forwarding
requirements:
  - virtual_link: NsVirtualLink_1

NsVirtualLink_1:
type: toska.nodes.nfv.NsVirtualLink
properties:
  connectivity_type:
    layer_protocols: [ ipv4 ]
  vl_profile:
    min_bitrate_requirements:
      root: 100000
    max_bitrate_requirements:
      root: 200000

Sap_external:
type: toska.nodes.nfv.Sap
properties:
  layer_protocols: [ ipv4 ]
requirements:
  - internal_virtual_link: NsVirtualLink_1

policies:
  - policy_affinity_group_1: # it shows that when VNF_1_forward_1 and VNF_3_forward_1
connected to NsVirtualLink_1, they should connect to the same l2 network.
    type: toska.policies.nfv.NsAffinityRule
    targets: [ VNF_1_forward_1, VNF_3_forward_1 ]
    properties:
      scope: l2_network

  - policy_affinity_group_2: # it shows that when VNF_2_forward_1, VNF_3_forward_2
and Sap_external connected to NsVirtualLink_1, they should connect to the same l2
network.
    type: toska.policies.nfv.NsAffinityRule
    targets: [ VNF_2_forward_1, VNF_3_forward_2, Sap_external ]
    properties:
      scope: l2_network

  - policy_antiaffinity_group_3: # it shows that when VNF_3_forward_1 and
VNF_3_forward_2 connected to NsVirtualLink_1, they should not connect to the same l2
network.
    type: toska.policies.nfv.NsAntiAffinityRule
    targets: [ VNF_3_forward_1, VNF_3_forward_2 ]
    properties:
      scope: l2_network

```

Figure A.20-2 shows the run-time NS instance model after instantiation based on the above NSD service template.

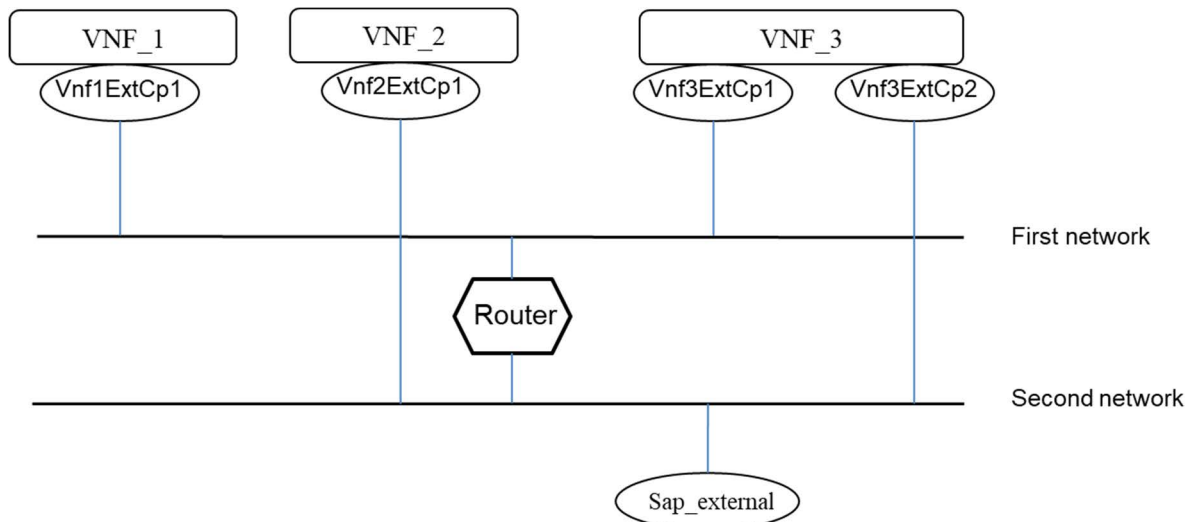


Figure A.20-2: NS instance run-time model after instantiation

A.21 VNFD with deployable modules design example

This annex illustrates the use of deployable modules in a VNFD. The descriptor of VNF1 consists of a single deployment flavour with the VDUs shown in figure A.21-1.

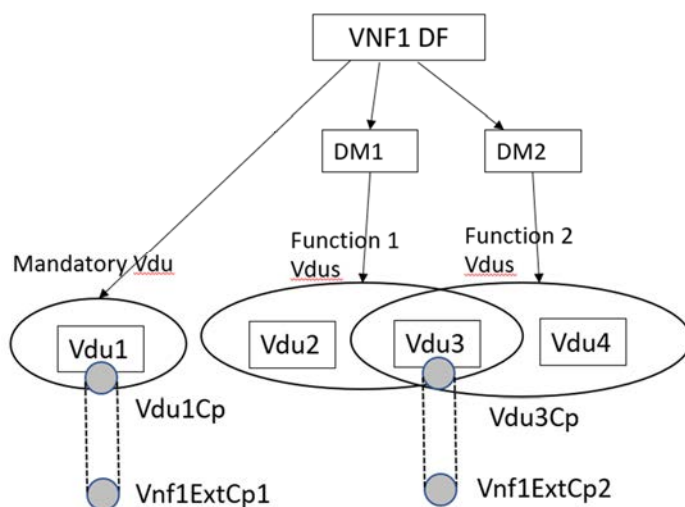


Figure A.21-1: VNF DF with deployable modules

VNF1 has the following VDUs:

- Vdu1, which does not belong to any deployable module, i.e. it is mandatory to deploy.
- Vdu2, which belongs to deployable module DM1, i.e. it will only be deployed when DM1 is selected.
- Vdu4, which belongs to deployable module DM2, i.e. it will only be deployed when DM2 is selected.
- Vdu3, which belongs to deployable modules DM1 and DM2, i.e. it will only be deployed when either DM1 or DM2, or both, are selected.

Vdu1 and Vdu3 have VduCps that are exposed as VnfExtCps.

The VNF1.yaml file contains the VNF1 node type definition:

```
tosca_definitions_version: tosca_simple_yaml_1_3

imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
    GS NFV-SOL 001

node_types:
  MyCompany.VNF1:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider: MyCompany
      product_name: VNF1_product
      software_version: '1.0'
      descriptor_version: '1.0'
      flavour_id: normal
      flavour_description: "VNF1 only flavour"
      vnfm_info: [ MyCompanyVnfm ]
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 0 ]
      - virtual_link_OAM
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 1 ]
      - virtual_link_service:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 1 ]
```

The following file contains the topology descriptor of VNF1:

```
tosca_definitions_version: tosca_simple_yaml_1_3
description: A sample VNF descriptor with containerized VDUs
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # VNFD type definitions
  - VNF1.yaml

topology_template:
  substitution_mappings:
    node_type: MyCompany.VNF1
  requirements:
    virtual_link_OAM: [ Vdu1Cp, virtual_link ]
    virtual_link_service: [ Vdu3Cp, virtual_link ]

  node_templates:
    VNF1:
      type: MyCompany.VNF1
      properties:
        change_selected_deployable_modules_op: [ CHANGE_CURRENT_VNF_PACKAGE,
SELECT_DEP_MOD ]
      # interfaces:
      # omitted for brevity
```

```
Vdu1:
  type: tosa.nodes.nfv.Vdu.OsContainerDeployableUnit
  # properties: omitted for brevity
  requirements:
    - container: Vdu1_Container

Vdu2:
  type: tosa.nodes.nfv.Vdu.OsContainerDeployableUnit
  # properties: omitted for brevity
  requirements:
    - container: Vdu2_Container

Vdu3:
  type: tosa.nodes.nfv.Vdu.OsContainerDeployableUnit
  # properties: omitted for brevity
  requirements:
    - container: Vdu3_Container

Vdu4:
  type: tosa.nodes.nfv.Vdu.OsContainerDeployableUnit
  # properties: omitted for brevity
  requirements:
    - container: Vdu4_Container

Vdu1_Container:
  type: tosa.nodes.nfv.Vdu.OsContainer
  # properties: omitted for brevity
  artifacts:
    sw_image_1:
      type: tosa.artifacts.nfv.SwImage
      file: Artifacts/Images/SwImage1
      # properties: omitted for brevity

Vdu2_Container:
  type: tosa.nodes.nfv.Vdu.OsContainer
  # properties: omitted for brevity
  artifacts:
    sw_image_2:
      type: tosa.artifacts.nfv.SwImage
      file: Artifacts/Images/SwImage2
      # properties: omitted for brevity

Vdu3_Container:
  type: tosa.nodes.nfv.Vdu.OsContainer
  # properties: omitted for brevity
  artifacts:
    sw_image_3:
      type: tosa.artifacts.nfv.SwImage
      file: Artifacts/Images/SwImage3
      # properties: omitted for brevity

Vdu4_Container:
  type: tosa.nodes.nfv.Vdu.OsContainer
  # properties: omitted for brevity
  artifacts:
    sw_image_4:
      type: tosa.artifacts.nfv.SwImage
      file: Artifacts/Images/SwImage4
      # properties: omitted for brevity
```

```
Vdu1Cp:
  type: toasca.nodes.nfv.VduCp
  # properties: omitted for brevity
  requirements:
    - virtual_binding: Vdu1

Vdu3Cp:
  type: toasca.nodes.nfv.VduCp
  # properties: omitted for brevity
  requirements:
    - virtual_binding: Vdu3

mciop1:
  type: toasca.nodes.nfv.Mciop
  requirements:
    - associatedVdu: Vdu1

mciop2:
  type: toasca.nodes.nfv.Mciop
  requirements:
    - associatedVdu: Vdu2

mciop3:
  type: toasca.nodes.nfv.Mciop
  requirements:
    - associatedVdu: Vdu3

mciop4:
  type: toasca.nodes.nfv.Mciop
  requirements:
    - associatedVdu: Vdu4

DM1:
  type: toasca.nodes.nfv.DeployableModule
  properties:
    name: service1
  requirements:
    - member: mciop2
    - member: mciop3

DM2:
  type: toasca.nodes.nfv.DeployableModule
  properties:
    name: service2
  requirements:
    - member: mciop3
    - member: mciop4
```

A.22 NSD with SAP to deployable modules mapping

A.22.1 Deployable modules in a constituent VNF of the NS

This annex illustrates the use of the SapToDeployableModuleMapping policy in an NSD. In this example the deployable modules are defined in a VNF which is a direct constituent of the NS, i.e. without any level of nesting NS between the NS and the VNF.

The descriptor of NS1 consists of a single NS deployment flavour. The NS flavour contains VNF1, described in the example in A.21 and other VNFs, not described further for simplicity. NS1 re-exposes the two VnfExtCps of VNF1 as SAPs.

This is represented in figure A.22.1-1.

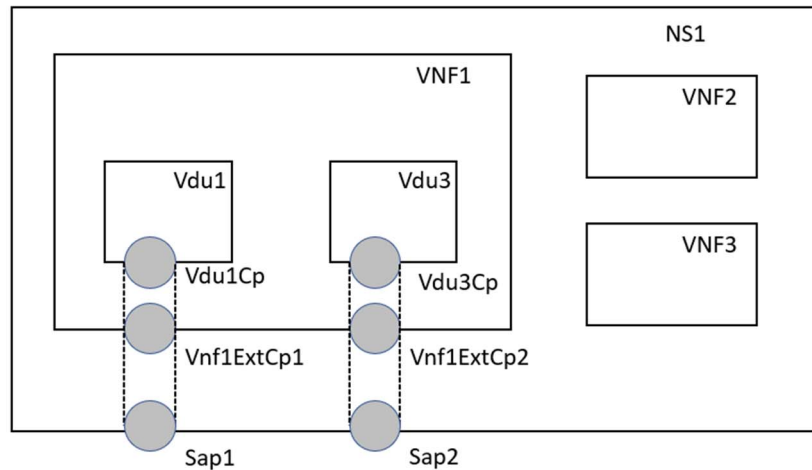


Figure A.22.1-1: NS DF with a constituent VNF with deployable modules

The NS1.yaml file contains the NS1 node type definition:

```
tosca_definitions_version: tosa_simple_yaml_1_3

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in ETSI GS NFV-
    SOL 001

node_types:
  MyServiceProviderCompany.NS1:
    derived_from: tosa.nodes.nfv.NS
    properties:
      descriptor_id: c65906ba-5165-11ee-be56-0242ac120002
      designer: MyServiceProvideCompany
      name: ExampleService
      version: '1.0'
      invariant_id: 1111-2222-aaaa-bbbb
      flavour_id: simple
# interfaces:
#   omitted for brevity
requirements:
  - virtual_link:
      capability: tosa.capabilities.nfv.VirtualLinkable
      relationship: tosa.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 0 ]
  - virtual_link_Sap1: # represents Sap1
      capability: tosa.capabilities.nfv.VirtualLinkable
      relationship: tosa.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 1 ]
  - virtual_link_Sap2: # represents Sap2
      capability: tosa.capabilities.nfv.VirtualLinkable
      relationship: tosa.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 1 ]
```

The following file contains the topology descriptor of NS1. Since Sap2 is exposing a VnfExtCp which in turn exposes a VduCp that belongs to one or multiple deployable modules in VNF1, a constituent of the NS, an instance of the SapToDeployableModuleMapping policy is included in the NSD to expose the information to a user of the NSD.

```
tosca_definitions_version: tosca_simple_yaml_1_3
description: A sample VNF descriptor with containerized VDUs
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # VNFD type definitions
  - NS1.yaml
  - VNF1.yaml # File containing VNF1 node type definition
# - VNF2.yaml
# - VNF3.yaml

topology_template:
  substitution_mappings:
    node_type: MyServiceProviderCompany.NS1
  requirements:
    virtual_link_Sap1: [ VNF1, virtual_link_OAM ]
    virtual_link_Sap2: [ VNF1, virtual_link_service ]

  node_templates:
    NS1:
      type: MyServiceProviderCompany.NS1
#   interfaces:
#     omitted for brevity

    VNF1:
      type: MyCompany.VNF1
      requirements:
        - virtual_link_OAM
        - virtual_link_service

# Other node templates omitted for brevity

policies:
  - SapToVNF1DeployableModulemapping
    type: tosca.policies.nfv.SapToDeployableModuleMapping
    properties:
      sap_to_depl_mod_mapping:
        virtual_link_Sap2: [ DM1, DM2 ]
    targets: [ VNF1 ]
```

It can be observed that no SAP to deployable module mapping related to Sap1 exists in the policy, as Sap1 is re-exposing Vnf1ExtCp1, which is a re-exposure of Vdu1Cp and this Cp does not belong to any deployable module in VNF1.

A.22.2 Deployable modules in a constituent nested NS of a composite NS

In the following example NS1 is used as nested NS of a composite NS, called NSComp. NSComp has one single NS flavour. In the composite NS, the SAPs of the nested NS are re-exposed, once more, as SAPs. This is illustrated in figure A.22.2-1.

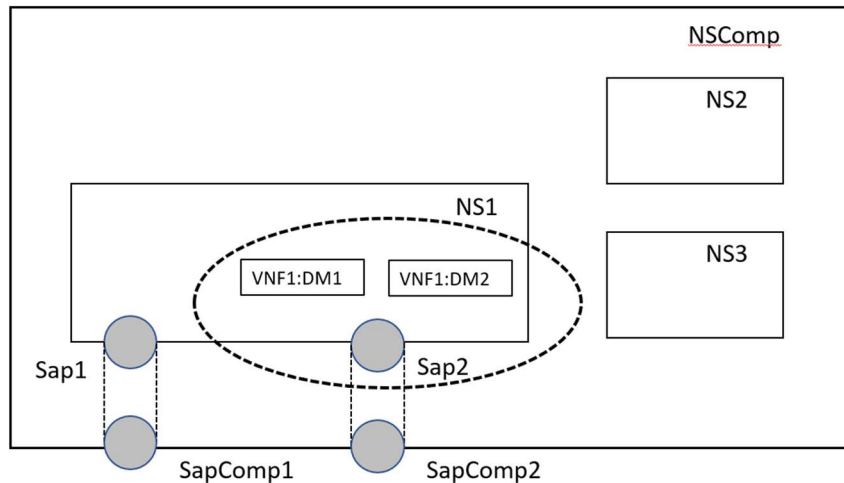


Figure A.22.2-1: Composite NS with a nested NS with deployable modules in its constituents

The NSComp.yaml file contains the NSComp node type definition:

```
tosca_definitions_version: tosca_simple_yaml_1_3

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in ETSI GS NFV-
    SOL 001

node_types:
  AnotherServiceProviderCompany.NSComp:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id: c415219e-4b26-11ee-be56-0242ac120002
      designer: OtherServiceProvideCompany
      name: ExampleCompositeService
      version: '1.0'
      invariant_id: 0011-2222-3333-bbbb
      flavour_id: simple
# interfaces:
#   omitted for brevity
requirements:
  - virtual_link:
      capability: tosca.capabilities.nfv.VirtualLinkable
      relationship: tosca.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 0 ]
  - virtual_link_SapComp1:
      capability: tosca.capabilities.nfv.VirtualLinkable
      relationship: tosca.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 1 ]
  - virtual_link_SapComp2:
      capability: tosca.capabilities.nfv.VirtualLinkable
      relationship: tosca.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 1 ]
```

The following file contains the topology descriptor of NSComp. Since the descriptor of NS1, which is a constituent of NSComp, includes a SapToDeployableModuleMapping policy that indicates that one of the SAPs in NS1 is related to a deployable module of an NS1 constituent, and this SAP is re-exposed as SAP in the composite NSComp, the NS descriptor of NSComp needs to include also a policy to indicate that SapComp2 is related to a deployable module of a constituent of NSComp.

It can be observed that in this policy, the values of the elements in the list in the 'sap_to_depl_mod_mapping' property that indicate the deployable modules, are the same as in the policy in the nested NS but preceded by the name of the node template in the NSD of the nested NS NS1, targeted by the policy, i.e. VNF1, and separated by a colon ':':

```
tosca_definitions_version: tosca_simple_yaml_1_3
description: A sample VNF descriptor with containerized VDUs
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # VNFD type definitions
  - NSComp.yaml
  - NS1.yaml # File containing NS1 node type definition
# - NS2.yaml
# - NS3.yaml

topology_template:
  substitution_mappings:
    node_type: OtherServiceProviderCompany.NSComp
  requirements:
    virtual_link_SapComp1: [ NS1, virtual_link_Sap1 ]
    virtual_link_SapComp2: [ NS1, virtual_link_Sap2 ]

  node_templates:
    NSComp:
      type: OtherServiceProviderCompany.NSComp
#   interfaces:
#     omitted for brevity

    NS1:
      type: MyServiceProviderCompany.NS1
      requirements:
        - virtual_link_sap1
        - virtual_link_sap2

# Other node templates omitted for brevity

policies:
  - SapToNS1DeployableModulemapping
    type: tosca.policies.nfv.SapToDeployableModuleMapping
    properties:
      sap_to_depl_mod_mapping:
        virtual_link_SapComp2: [ VNF1:DM1, VNF1:DM2 ]
    targets: [ NS1 ]
```

A.23 VNFD example with simplified design by using MCIOP

This annex illustrates a VNFD example with simplified design to support VNF realized as OS containers by using MCIOPs. In order to avoid duplicated information included in the VNFD, in this example, the OS container related resources are only included in the associated MCIOP node(s) (i.e. in the helm chart), the Vdu.OsContainer node is not used. The logical structure of the descriptor with a single deployment flavour is shown in figure A.23-1.

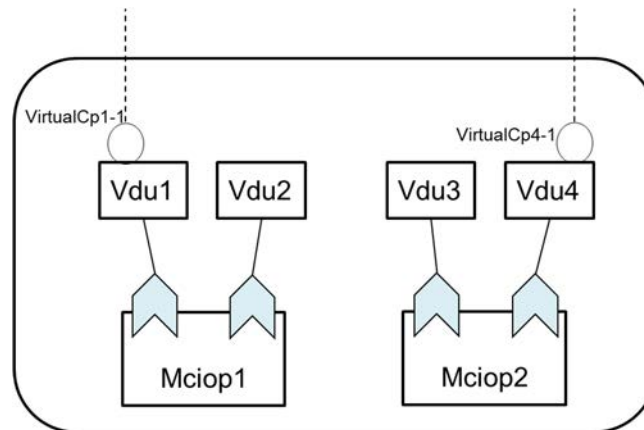


Figure A.23-1: VNFD with simplified design by using MCIOP

In the example, the VNFD includes the following VDUs:

- Vdu1 and Vdu2 which associate with Mciop1 (e.g. represents helm chart 1).
- Vdu3 and Vdu4 which associate with Mciop2 (e.g. represents helm chart 2).
- Vdu1 and Vdu4 have VirtualCps that are exposed as VnfExtCps.

The VNF_node_definition.yaml file contains the VNF node type definition:

```
tosca_definitions_version: tosca_simple_yaml_1_3

imports:
- etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in ETSI
  GS NFV-SOL 001

data_types:
  MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters:
    derived_from: tosca.datatypes.nfv.VnfOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
        required: true
        default: value_1
      parameter_2:
        type: string
        required: true
        default: value_2

node_types:
  MyCompany.ExampleVNF:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id: b1aa0ce2-esss-5fb7-96eg-5830d40bb137
      provider: MyCompany
      product_name: NF1_product
      software_version: '1.0'
      descriptor_version: '1.0'
      flavour_id: normal
      flavour_description: ""
      vnfm_info: [ MyCompanyVnfm ]
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
```

```

        occurrences: [ 0, 0 ]
    - virtual_link_1
      capability: toasca.capabilities.nfv.VirtualLinkable
      relationship: toasca.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 1 ]
    - virtual_link_2:
      capability: toasca.capabilities.nfv.VirtualLinkable
      relationship: toasca.relationships.nfv.VirtualLinksTo
      occurrences: [ 0, 1 ]
  interfaces:
    Vnflcm:
      type: toasca.interfaces.nfv.Vnflcm
      operations:
        instantiate:
          inputs:
            additional_parameters:
              type: MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters
              required: false

```

The following file contains the topology descriptor of the VNFD:

```

tosca_definitions_version: toasca_simple_yaml_1_3
description: A sample VNF descriptor with containerized VDUs
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # VNFD type definitions
  - VNF_node_definition.yaml

topology_template:
  substitution_mappings:
    node_type: MyCompany.ExampleVNF
  requirements:
    virtual_link_1: [ VirtualCp1_1, virtual_link ]
    virtual_link_2: [ VirtualCp4_1, virtual_link ]

  node_templates:
    MyVnf:
      type: MyCompany.ExampleVNF
      properties:
        flavour_description: 'vnf simple flavour description'

    Vdu1:
      type: toasca.nodes.nfv.Vdu.OsContainerDeployableUnit
      properties:
        name: "lb"
        description: "Advanced Software Load Balancer"
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 4
        mcio_identification_data:
          name: lb_mcio
          type: Deployment

    Vdu2:
      type: toasca.nodes.nfv.Vdu.OsContainerDeployableUnit
      properties:
        name: "MainFunction1"
        description: "The main part of function1"
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 10

```

```
    mcio_identification_data:
      name: db_mcio
      type: Deployment

Vdu3:
  type: toasca.nodes.nfv.Vdu.OsContainerDeployableUnit
  properties:
    name: "db"
    description: "An DB server"
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 4
    mcio_identification_data:
      name: db_mcio
      type: StatefulSet

Vdu4:
  type: toasca.nodes.nfv.Vdu.OsContainerDeployableUnit
  properties: omitted for brevity
    name: "MainFunction2"
    description: "The main part of function2"
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 10
    mcio_identification_data:
      name: db_mcio
      type: Deployment

VirtualCp1_1:
  type: toasca.nodes.nfv.VirtualCp
  properties:
    layer_protocols: [ ipv6 ]
    trunk_mode: true
    description: virtual CP for data base communication
    additionalServiceData:
      - portData:
          - name: DBService
            protocol: tcp
            port: 8001
            portConfigurable: false
  requirements:
    - target: Vdu_1
    # - virtual_link: the target node is determined in the NSD

VirtualCp4_1:
  type: toasca.nodes.nfv.VirtualCp
  properties:
    layer_protocols: [ ipv6 ]
    trunk_mode: true
    description: virtual CP for session communication
    additionalServiceData:
      - portData:
          - name: SessionService
            protocol: tcp
            port: 443
            portConfigurable: false
  requirements:
    - target: Vdu_4
    # - virtual_link: the target node is determined in the NSD

mciop1:
```

```
type: tosa.nodes.nfv.Mciop
requirements:
  - associatedVdu: Vdu1
  - associatedVdu: Vdu2
artifacts:
  helm_1:
    description: Helm Chart for Function1
    type: tosa.artifacts.nfv.HelmChart
    file: Artifacts/Chart/Function1.tgz

mciop2:
type: tosa.nodes.nfv.Mciop
requirements:
  - associatedVdu: Vdu3
  - associatedVdu: Vdu4
artifacts:
  helm_2:
    description: Helm Chart for Function2
    type: tosa.artifacts.nfv.HelmChart
    file: Artifacts/Chart/Function2.tgz
```

Annex B (normative): etsi_nfv_sol001_type definitions

B.1 Purpose

All type definitions specified in clauses 6, 7, 8 and 9 of the present document are gathered in four definition files.

The file names are structured as follows:

- etsi_nfv_sol001_common_types.yaml, for the common type definitions provided in clause 9 which are used by at least two types of deployment templates among those identified in clause 5.1;
- etsi_nfv_sol001_vnfd_types.yaml, for the definitions provided in clause 6 and only used in a VNFD service template design;
- etsi_nfv_sol001_nsd_types.yaml, for the definitions provided in clause 7 and only used in an NSD service template design;
- etsi_nfv_sol001_pnfd_types.yaml, for the definitions provided in clause 8 and only used in a PNFD service template design.

B.2 VNFD type definitions file

All type definitions specified in clause 6 of the present document are contained in the file etsi_nfv_sol001_vnfd_types.yaml which is available at the following URL:

- https://forge.etsi.org/rep/nfv/SOL001/raw/v5.2.1/etsi_nfv_sol001_vnfd_types.yaml.

NOTE 1: The file etsi_nfv_sol001_vnfd_types.yaml includes a TOSCA import definition referencing etsi_nfv_sol001_common_types.yaml file. If the later file is included in the VNF package, the import definition can reference the local file using appropriate path in the VNF package.

This file is a TOSCA service template that only contains definitions. The template_version in the metadata section within this template is structured as x.y.z, where x, y and z represent the version of this file and are set respectively to "5", "2" and "1" for this version of the present document. In subsequent versions of the present document, "x", "y" and "z" in the template_version will be incremented only if there are changes in the VNFD type definitions.

A TOSCA service template representing a VNFD complying with the present document shall contain import statement referencing this file, as defined in clause 5.6.1.

NOTE 2: This file may, but need not, be included in the VNF Package.

B.3 NSD type definitions file

All type definitions specified in clause 7 of the present document are contained in the file etsi_nfv_sol001_nsd_types.yaml which is available at the following URL:

- https://forge.etsi.org/rep/nfv/SOL001/raw/v5.2.1/etsi_nfv_sol001_nsd_types.yaml.

NOTE 1: The file etsi_nfv_sol001_nsd_types.yaml includes a TOSCA import definition referencing etsi_nfv_sol001_common_types.yaml file. If the later file is included in the NSD file archive, the import definition can reference the local file using appropriate path in the NSD file archive.

This file is a TOSCA service template that only contains definitions. The template_version in the metadata section within this template is structured as x.y.z, where x, y and z represent the version of this file and are set respectively to "5", "2" and "1" for this version of the present document. In subsequent versions of the present document, "x", "y" and "z" in the template_version will be incremented only if there are changes in the NSD type definitions.

A TOSCA service template representing an NSD complying with the present document shall contain import statement referencing this file, as defined in clause 5.6.2.

NOTE 2: This file may, but need not, be included in the NSD file archive.

B.4 PNFD type definitions file

All type definitions specified in clause 8 of the present document are contained in the file `etsi_nfv_sol001_pnfd_types.yaml` which is available at the following URL:

- https://forge.etsi.org/rep/nfv/SOL001/raw/v5.2.1/etsi_nfv_sol001_pnfd_types.yaml.

NOTE: The file `etsi_nfv_sol001_pnfd_types.yaml` includes a TOSCA import definition referencing `etsi_nfv_sol001_common_types.yaml` file.

This file is a TOSCA service template that only contains definitions. The `template_version` in the metadata section within this template is structured as `x.y.z`, where `x`, `y` and `z` represent the version of this file and are set respectively to "5", "2" and "1" for this version of the present document. In subsequent versions of the present document, "x", "y" and "z" in the `template_version` will be incremented only if there are changes in the PNFD type definitions.

A TOSCA service template representing a PNFD complying with the present document shall contain import statement referencing this file, as defined in clause 5.6.3.

B.5 Common type definitions file

The type definitions as specified in clause 9 and used by at least two types of deployment templates are contained in the file `etsi_nfv_sol001_common_types.yaml` which is available at the following URL:

- https://forge.etsi.org/rep/nfv/SOL001/raw/v5.2.1/etsi_nfv_sol001_common_types.yaml.

This file is a TOSCA service template that only contains definitions. The `template_version` in the metadata section within this template is structured as `x.y.z`, where `x`, `y` and `z` represent the version of this file and are set respectively to "5", "2" and "1" for this version of the present document. In subsequent versions of the present document, "x", "y" and "z" in the `template_version` will be incremented only if there are changes in the common type definitions.

NOTE: This file may, but need not, be included in the VNF package or NSD file archive.

Annex C (normative): Conformance

C.1 Purpose

The present document specifies a data model for the VNFD, the NSD and the PNFD, by using the grammar defined in the TOSCA-Simple-Profile-YAML-v1.3 [20]. This annex specifies the requirements to be fulfilled for claiming conformance to the present document.

C.2 NFV TOSCA YAML service template

A VNFD, an NSD or a PNFD conforms to the present document if it complies with all the requirements below:

- 1) A VNFD conformant to the present document shall comply with the requirements in clause 6 of the present document and to the specification of the elements of the TOSCA-Simple-Profile-YAML-v1.3 [20] it uses, unless otherwise stated in clause 6 of the present document.
- 2) An NSD conformant to the present document shall comply with the requirements in clause 7 of the present document and to the specification of the elements of the TOSCA-Simple-Profile-YAML-v1.3 [20] it uses, unless otherwise stated in clause 7 of the present document.
- 3) A PNFD conformant to the present document shall comply with the requirements in clause 8 of the present document and to the specification of the elements of the TOSCA-Simple-Profile-YAML-v1.3 [20] it uses, unless otherwise stated in clause 8 of the present document.
- 4) When using or referring to the TOSCA normative types listed in table C.2-1, it is valid according to the definitions given in clauses 6, 7, 8 and 9 of the present document and to section 5 of the TOSCA-Simple-Profile-YAML-v1.3 [20].

Table C.2-1: TOSCA normative types used in the present document

Types	VNFD	NSD	PNFD
tosca.datatypes.Root	X	X	X
tosca.artifacts.Deployment.Image	X		
tosca.artifacts.Implementation	X		
tosca.capabilities.Root	X	X	
tosca.capabilities.Node	X	X	X
tosca.relationships.Root	X	X	
tosca.relationships.DependsOn	X	X	X
tosca.interfaces.Root	X	X	
tosca.nodes.Root	X	X	X
tosca.groups.Root	X	X	
tosca.policies.Root	X	X	X
tosca.policies.Placement	X	X	

- 5) A VNFD conformant to the present document shall comply with VNFD TOSCA service template design specified in clause 6.11 of the present document.
- 6) A NSD conformant to the present document shall comply with NSD TOSCA service template design specified in clause 7.11 of the present document.
- 7) A PNFD conformant to the present document shall comply with PNFD TOSCA service template design specified in clause 8.11 of the present document.
- 8) A VNFD and NSD conformant to the present document shall comply with rules for Type extension defined in clause 5.7 of the present document.

- 9) When using the TOSCA functions listed in table 5.9-1, it is valid according to section 4 of the TOSCA-Simple-Profile-YAML-v1.3 [20].

C.3 NFV TOSCA processor

A processor or program conforms to the present document as NFV TOSCA processor for VNFD, NFV TOSCA processor for NSD or NFV TOSCA processor for PNFD if it complies with all the requirements below:

- 1) It can parse and recognize the elements of any VNFD, NSD or PNFD that conform to the present document, and shall generate errors for those documents that fail to conform to the present document.
- 2) It shall comply with all requirements and implement the semantics associated with the definitions specified in clauses 6, 7, 8 and 9 of the present document.
- 3) It shall resolve the import definitions, as described in clause 5.6 of the present document.

Annex D (informative): Mapping between properties of TOSCA types and API attributes

D.1 Introduction

This annex provides the mapping between properties of TOSCA types defined in the present document and defined in the following API specifications: ETSI GS NFV-SOL 002 [22], ETSI GS NFV-SOL 003 [25], and ETSI GS NFV-SOL 005 [i.10].

NOTE: See also Annex A "Mapping operations to protocol elements" of ETSI GS NFV-SOL 002 [22], ETSI GS NFV-SOL 003 [25] and ETSI GS NFV-SOL 005 [i.10] for each operation.

D.2 VNFD-related constructs

Table D.2-1 provides the mapping between the properties of TOSCA types related to the VNFD and API attributes, which include: resource or notification data type (and referenced structured data type when available), attribute name and type in the resource or notification data type, and the interface operation in which the data type is used.

NOTE: In the "Data model" column of the table, an arrow "->" indicates the navigation through the resource, notification and referenced structured data types.

Table D.2-1: Mapping of API attributes and TOSCA constructs

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
	Type and attribute name	Data model	Operation (see note 1)
tosca.nodes.nfv.VNF -> descriptor_id	(Identifier) vnfdId	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfInfoModificationRequest	Modify VNF Information (see SOL002 and SOL003)
		VnfInfoModifications	
		VnfLcmOpOcc -> VnfInfoModifications	Get Operation Status (see SOL002 and SOL003)
		VnfLcmOperationOccurrenceN otification -> VnfInfoModifications	Notify about VNF LCM (see SOL002 and SOL003)
		VnfPkgInfo	Create VNF package (see SOL005) Query/Read VNF Package Info (see SOL003 and SOL005)
		VnfPackageOnboardingNotifica tion	Notify about VNF Package (see SOL003 and SOL005)
		VnfPackageChangeNotification	Notify about VNF Package (see SOL003 and SOL005)
		NsInstance -> VnfInstance	Query NS (see SOL005)
		UpdateNsRequest -> InstantiateVnfData	Update NS (see SOL005)

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
		NsLcmOperationOccurrenceNo tification -> AffectedVnf	Notify about NS LCM (see SOL005)
		PkgmSubscriptionRequest -> PkgmNotificationsFilter	Subscription about VNF Package (see SOL003 and SOL005)
		PkgmSubscription -> PkgmNotificationsFilter	Subscription about VNF Package (see SOL003 and SOL005)
tosca.nodes.nfv.VNF -> provider	(String) vnfProvider	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfPkgInfo	Create VNF package (see SOL005) Query/Read VNF Package Info (see SOL003 and SOL005)
		NsInstance -> VnfInstance	Query NS (see SOL005)
		PkgmSubscriptionRequest -> PkgmNotificationsFilter -> vnfProductsFromProviders	Subscription about VNF Package (see SOL003 and SOL005)
		PkgmSubscription -> PkgmNotificationsFilter -> vnfProductsFromProviders	Subscription about VNF Package (see SOL003 and SOL005)
tosca.nodes.nfv.VNF -> product_name	(String) vnfProductName	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfPkgInfo	Create VNF package (see SOL005) Query/Read VNF Package Info (see SOL003 and SOL005)
		NsInstance -> VnfInstance	Query NS (see SOL005)
		PkgmSubscriptionRequest -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts	Subscription about VNF Package (see SOL003 and SOL005)
		PkgmSubscription -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts	Subscription about VNF Package (see SOL003 and SOL005)
tosca.nodes.nfv.VNF -> software_version	(Version) vnfSoftwareVersion	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfPkgInfo	Create VNF package (see SOL005)

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
			Query/Read VNF Package Info (see SOL003 and SOL005)
		NsInstance -> VnfInstance	Query NS (see SOL005)
		PkgmSubscriptionRequest -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts -> versions	Subscription about VNF Package (see SOL003 and SOL005)
		PkgmSubscription -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts -> versions	Subscription about VNF Package (see SOL003 and SOL005)
tosca.nodes.nfv.VNF -> descriptor_version	(Version) vnfdVersion	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfInfoModifications	Modify VNF Information (see SOL002 and SOL003)
		VnfLcmOpOcc -> VnfInfoModifications	Get Operation Status (see SOL002 and SOL003)
		VnfLcmOperationOccurrenceNotification -> VnfInfoModifications	Notify about VNF LCM (see SOL002 and SOL003)
		VnfPkgInfo	Create VNF package (see SOL005) Query/Read VNF Package Info (see SOL003 and SOL005)
		NsInstance -> VnfInstance	Query NS (see SOL005)
		PkgmSubscriptionRequest -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts -> versions	Subscription about VNF Package (see SOL003 and SOL005)
		PkgmSubscription -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts -> versions	Subscription about VNF Package (see SOL003 and SOL005)
tosca.nodes.nfv.VNF -> configurable_properties	(KeyValuePairs) vnfConfigurableProperties	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnfInstance	Query NS (see SOL005)
		VnfInfoModificationRequest VnfInfoModifications	Modify VNF Information (see SOL002 and SOL003)
		VnfLcmOpOcc -> VnfInfoModifications	Get Operation Status (see SOL002 and SOL003)

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
	Type and attribute name	Data model	Operation (see note 1)
		VnfLcmOperationOccurrenceNotification -> VnfInfoModifications	Notify about VNF LCM (see SOL002 and SOL003)
		ChangeCurrentVnfPkgRequest	Change current VNF package (see SOL002 and SOL003)
tosca.nodes.nfv.VNF ->flavour_id	(IdentifierInVnfd) flavourId	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		InstantiateVnfRequest	Instantiate VNF (see SOL002 and SOL003)
		GrantRequest	Grant Lifecycle Operation (see SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo	Query NS (see SOL005)
	(IdentifierInVnfd) vnfFlavourId	UpdateNsRequest -> instantiateVnfData	Update NS (see SOL005)
	(IdentifierInVnfd) newFlavourId	ChangeVnfFlavourRequest	Change VNF Flavour (see SOL002 and SOL003)
		UpdateNsRequest -> ChangeVnfFlavourData	Update NS (see SOL005)
tosca.policies.nfv.ScalingAspects ->aspects[key]	(IdentifierInVnfd) aspectId	VnfInstance -> instantiatedVnfInfo -> ScaleInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		ScaleVnfRequest	Scale VNF (see SOL002 and SOL003)
		ScaleVnfToLevelRequest -> ScaleInfo	Scale VNF to level (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> ScaleInfo	Query NS (see SOL005)
tosca.datatypes.nfv.ScaleInfo ->scale_level	(Integer) scaleLevel	VnfInstance -> instantiatedVnfInfo -> ScaleInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		ScaleVnfToLevelRequest -> ScaleInfo	Scale VNF to level (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> ScaleInfo	Query NS (see SOL005)
Node template name of type tosca.nodes.nfv.VnfExtCp	(IdentifierInVnfd) cpdId	VnfInstance -> instantiatedVnfInfo -> VnfExtCpInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		InstantiateVnfRequest	Instantiate VNF

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
		-> ExtVirtualLinkData -> VnfExtCpData	(see SOL002 and SOL003)
		ChangeVnfFlavourRequest -> ExtVirtualLinkData -> VnfExtCpData	Change VNF Flavour (see SOL002 and SOL003)
		ChangeExtVnfConnectivityReq uest -> ExtVirtualLinkData -> VnfExtCpData	Change External VNF Connectivity (see SOL002 and SOL003)
		Grant ->ExtVirtualLinkData ->VnfExtCpData	Grant Lifecycle Operation (see SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> VnfExtCpInfo	Query NS (see SOL005)
Node template name of type tosca.nodes.nfv.VnfVirtual Link	(IdentifierInVnfd) resourceTemplateId vnfVirtualLinkDescId	GrantRequest -> ResourceDefinition	Grant Lifecycle Operation (see SOL003)
		VnfInstance -> instantiatedVnfInfo -> ExtManagedVirtualLinkInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfInstance -> instantiatedVnfInfo -> VnfVirtualLinkResourceInfo	Query VNF (see SOL003)
		InstantiateVnfRequest -> ExtManagedVirtualLinkData	Instantiate VNF (see SOL002 and SOL003)
		ChangeVnfFlavourRequest -> ExtManagedVirtualLinkData	Change VNF Flavour (see SOL002 and SOL003)
		VnfLcmOpOcc -> resourceChanges -> AffectedVirtualLink	Get Operation Status (see SOL002 and SOL003)
		Grant -> ExtManagedVirtualLinkData	Grant Lifecycle Operation (see SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> ExtManagedVirtualLinkInfo	Query NS (see SOL005)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> VnfVirtualLinkResourceInfo	Query NS (see SOL005)
		(IdentifierInVnfd) vnfVirtualLinkDescId	Update NS (see SOL005)
tosca.datatypes.nfv.VnfMo nitoringParameter	(IdentifierInVnfd) id	VnfInstance -> instantiatedVnfInfo -> MonitoringParameter	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> MonitoringParameter	Query NS (see SOL005)
tosca.datatypes.nfv.VnfMo nitoringParameter ->name	(String) name	VnfInstance -> instantiatedVnfInfo -> MonitoringParameter	Create VNF Identifier (see SOL002 and SOL003) Query VNF

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
			(see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> MonitoringParameter	Query NS (see SOL005)
tosca.nodes.nfv.VNF -> localization_languages	(String) localizationLanguage	VnfInstance -> instantiatedVnfInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo	Query NS (see SOL005)
Node template name of type tosca.nodes.nfv.Vdu.Compute Node template name of type tosca.nodes.nfv.Vdu.OsC ontainerDeployableUnit	(IdentifierInVnfd)vduId	VnfInstance -> instantiatedVnfInfo -> VnfcResourceInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfInstance -> instantiatedVnfInfo -> VnfcInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfLcmOpOcc -> resourceChanges -> AffectedVnfc	Get Operation Status (see SOL002 and SOL003)
		VnfLcmOperationOccurrenceN otification -> AffectedVnfc	Notify about VNF LCM (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> VnfcResourceInfo	Query NS (see SOL005)
	(IdentifierInVnfd) vnfdVirtualComputeDescId	Grant -> vimAssets -> VimComputeResourceFlavour	Grant Lifecycle Operation (see SOL003)
Node template name of type tosca.nodes.nfv.VduCp	(IdentifierInVnfd) cpId	VnfInstance -> instantiatedVnfInfo -> VnfcResourceInfo -> vnfcCplInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfInstance -> instantiatedVnfInfo -> VnfExtCplInfo See note 3.	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		InstantiateVnfRequest -> ExtVirtualLinkData -> VnfExtCpData See note 3.	Instantiate VNF (see SOL002 and SOL003)
		ChangeVnfFlavourRequest -> ExtVirtualLinkData -> VnfExtCpData See note 3.	Change VNF Flavour (see SOL002 and SOL003)
		ChangeExtVnfConnectivityReq uest	Change External VNF Connectivity

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
		-> ExtVirtualLinkData -> VnfExtCpData See note 3.	(see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> VnfExtCpInfo See note 3.	Query NS (see SOL005)
	(IdentifierInVnfd) resourceTemplateId See note 3.	GrantRequest -> ResourceDefinition See note 3.	Grant Lifecycle Operation (see SOL003)
Node template name of type tosca.nodes.nfv.Vdu.VirtualBlockStorage	(IdentifierInVnfd) virtualStorageDescId	VnfInstance -> instantiatedVnfInfo -> VirtualStorageResourceInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
Node template name of type tosca.nodes.nfv.Vdu.VirtualObjectStorage		VnfLcmOpOcc -> resourceChanges -> AffectedVirtualStorage	Get Operation Status (see SOL002 and SOL003)
Node template name of type tosca.nodes.nfv.Vdu.VirtualFileStorage		VnfLcmOperationOccurrenceNotification -> AffectedVirtualStorage	Notify about VNF LCM (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> VirtualStorageResourceInfo	Query NS (see SOL005)
	(IdentifierInVnfd) resourceTemplateId	GrantRequest -> ResourceDefinition	Grant Lifecycle Operation (see SOL003)
tosca.datatypes.nfv.VnfInfoModifiableAttributes ->metadata	(KeyValuePairs) metadata	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnfInstance	Query NS (see SOL005)
		CreateVnfRequest	Create VNF Identifier (see SOL002 and SOL003)
		VnfInfoModificationRequest	Modify VNF Information (see SOL002 and SOL003)
		VnfInfoModifications	
		VnfLcmOpOcc -> VnfInfoModifications	Get Operation Status (see SOL002 and SOL003)
		VnfLcmOperationOccurrenceNotification -> VnfInfoModifications	Notify about VNF LCM (see SOL002 and SOL003)
tosca.datatypes.nfv.VnfInfoModifiableAttributes ->extensions	(KeyValuePairs) extensions	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnfInstance	Query NS (see SOL005)
		InstantiateVnfRequest	Instantiate VNF (see SOL002 and SOL003)
		VnfInfoModificationRequest	Modify VNF Information (see SOL002 and SOL003)
		VnfInfoModifications	

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
		VnfLcmOpOcc -> VnfInfoModifications	Get Operation Status (see SOL002 and SOL003)
		VnfLcmOperationOccurrenceN otification -> VnfInfoModifications	Notify about VNF LCM (see SOL002 and SOL003)
		ChangeCurrentVnfPkgRequest	Change current VNF package (see SOL002 and SOL003)
tosca.policies.nfv.Instantia tionLevel ->levels[key]	(IdentifierInVnfd) instantiationLevelId	InstantiateVnfRequest	Instantiate VNF (see SOL002 and SOL003)
		ScaleVnfToLevelRequest	Scale VNF to level (see SOL002 and SOL003)
		ChangeVnfFlavourRequest	Change VNF Flavour (see SOL002 and SOL003)
		GrantRequest	Grant Lifecycle Operation (see SOL003)
		UpdateNsRequest -> changeVnfFlavourData See note 4.	Update NS (see SOL005)
	(IdentifierInVnfd) vnfInstantiationLevelId	UpdateNsRequest -> InstantiateVnfData See note 4.	Update NS (see SOL005)
		ScaleNsRequest -> ScaleVnfData -> ScaleToLevelData	Scale NS (see SOL005)
Properties of tosca.datatypes.nfv.VnfOp erationAdditionalParamete rs in the inputs of the instantiate operation of the Vnflcm interface See note 2	(KeyValuePairs) additionalParams	InstantiateVnfRequest	Instantiate VNF (see SOL002 and SOL003)
Properties of tosca.datatypes.nfv.VnfOp erationAdditionalParamete rs in the inputs of the scale operation of the Vnflcm interface See note 2	(KeyValuePairs) additionalParams	ScaleVnfRequest	Scale VNF (see SOL002 and SOL003)
Properties of tosca.datatypes.nfv.VnfOp erationAdditionalParamete rs in the inputs of the 'scale to level' operation of the Vnflcm interface See note 2	(KeyValuePairs) additionalParams	ScaleVnfToLevelRequest	Scale VNF to level (see SOL002 and SOL003)
Properties of tosca.datatypes.nfv.VnfOp erationAdditionalParamete rs in the inputs of the 'change vnf flavour operation' of the Vnflcm interface See note 2	(KeyValuePairs) additionalParams	ChangeVnfFlavourRequest	Change VNF Flavour (see SOL002 and SOL003)
Properties of	(KeyValuePairs)	OperateVnfRequest	Operate VNF

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
tosca.datatypes.nfv.VnfOperationAdditionalParameters in the inputs of the operate operation of the Vnflcm interface See note 2	additionalParams		(see SOL002 and SOL003)
Properties of tosca.datatypes.nfv.VnfOperationAdditionalParameters in the inputs of the heal operation of the Vnflcm interface See note 2	(KeyValuePairs) additionalParams	HealVnfRequest	Heal VNF (see SOL002 and SOL003)
Properties of tosca.datatypes.nfv.VnfOperationAdditionalParameters in the inputs of the 'change external vnf connectivity' operation of the Vnflcm interface See note 2	(KeyValuePairs) additionalParams	ChangeExtVnfConnectivityRequest	Change External VNF Connectivity (see SOL002 and SOL003)
Properties of tosca.datatypes.nfv.VnfOperationAdditionalParameters in the inputs of the terminate operation of the Vnflcm interface See note 2	(KeyValuePairs) additionalParams	TerminateVnfRequest	Terminate VNF (see SOL002 and SOL003)
Properties of tosca.datatypes.nfv.VnfOperationAdditionalParameters in the inputs of the create snapshot operation of the Vnflcm interface See note 2	(KeyValuePairs) additionalParams	CreateVnfSnapshotRequest	CreateVNFSnapshot (see SOL002 and SOL003)
Properties of tosca.datatypes.nfv.VnfOperationAdditionalParameters in the inputs of the revert to snapshot operation of the Vnflcm interface See note 2	(KeyValuePairs) additionalParams	RevertToVnfSnapshotRequest	RevertToVNFSnapshot (see SOL002 and SOL003)
Properties of tosca.datatypes.nfv.VnfOperationAdditionalParameters in the inputs of the Change current VNF operation of the Vnflcm or ChangeCurrentVnfPackage interface. See note 2	(KeyValuePairs) additionalParams	ChangeCurrentVnfPkgRequest	Change current VNF package (see SOL002 and SOL003)
Node template name of type tosca.nodes.nfv.Vdu.Compute	(IdentifierInVnfd) id	VnfPkgInfo -> VnfPackageSoftwareImageInfo	Create VNF Package Info (see SOL005) Query VNF Package Info (see SOL005)

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
	Type and attribute name	Data model	Operation (see note 1)
Node template name of type tosca.nodes.nfv.Vdu.VirtualBlockStorage			
Node template name of type tosca.nodes.nfv.Vdu.OsContainer			
Name of attribute for VNF indicator in VNF node	(IdentifierInVnfd) Id	VnfIndicator	Get indicator value (see SOL002 and SOL003)
		VnfIndicatorValueChangeNotification	Notify about VNF indicator value change (see SOL002 and SOL003).
Name of attribute for VNF indicator in VNF	(String) Name	VnfIndicator	Get indicator value (see SOL002 and SOL003)
	(IdentifierInVnfd) vnfIndicatorId	VnfIndicatorValueChangeNotification	Notify about VNF indicator value change (see SOL002 and SOL003).
Value of attribute for VNF indicator in VNF node	(Object) value	VnfIndicator	Get indicator value (see SOL002 and SOL003)
		VnfIndicatorValueChangeNotification	Notify about VNF indicator value change (see SOL002 and SOL003).
Node template name of type tosca.nodes.nfv.Vdu.Compute	(IdentifierInVnfd) vnfSoftwareImageId	Grant -> vimAssets -> VimSoftwareImage	Grant Lifecycle Operation (see SOL003)
Node template name of type tosca.nodes.nfv.Vdu.VirtualBlockStorage			
Node template name of type tosca.nodes.nfv.Vdu.OsContainer			
tosca.datatypes.nfv.VnfPackageChangeSelector -> destination_descriptor_id		ChangeCurrentVnfPkgRequest	Change current VNF package (see SOL002 and SOL003)
tosca.policies.nfv.LcmCoordinationsForLcmOperation ->actions -> referenced_coordination_actions	coordinationActionName	LcmCoordRequest LcmCoord	CoordinateLcmOperation for VNF LCM (see SOL002)
tosca.policies.nfv.LcmCoordinationsForLcmOperation ->actions -> referenced_coordination_actions	coordinationActionName	LcmCoordRequest LcmCoord	CoordinateLcmOperation for Change current VNF package (see SOL002)

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
Properties of tosca.datatypes.nfv.Input OpCoordParams See note 2	inputParameters	LcmCoordRequest	CoordinateLcmOperation for VNF LCM and Change current VNF package (see SOL002)
Properties of tosca.datatypes.nfv.Output OpCoordParams See note 2	outputParameters	LcmCoord	CoordinateLcmOperation for VNF LCM and Change current VNF package (see SOL002)
Properties of tosca.datatypes.nfv.VnfcC onfigurableProperties	(KeyValuePairs)	VnflInfoModificationRequest	Modify VNF Information (see SOL002)
	vnfcConfigurableProperties	VnflInfoModifications	
tosca.datatypes.nfv.L3Prot ocolData -> cidr -> ip_allocation_pools -> gateway_ip -> dhcp_enabled -> ipv6_address_mode	n/a (see note 5)	n/a (see note 5)	n/a (see note 5)
Node template name of type tosca.nodes.nfv.Deployabl eModule	(IdentifierInVnfd) selectedDeployableModule	InstantiateVnfRequest	Instantiate VNF (see SOL002 and SOL003)
		ChangeVnfFlavourRequest	Change VNF Flavour (see SOL002 and SOL003)
		ChangeCurrentVnfPkgRequest	Change current VNF package (see SOL002 and SOL003)
		SelectVnfDeployableModulesR equest	Select VNF Deployable modules (see SOL002 and SOL003)
		GrantRequest	Grant Lifecycle Operation (see SOL003)
		VnflInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		InstantiateNsRequest -> ParamsForVnf InstantiateNsRequest -> ParamsForNestedNs -> DeployableModulesInConstitue ntVnf	Instantiate NS (see SOL005)
		UpdateNsRequest -> InstantiateVnfData UpdateNsRequest -> AssocNewNsdVersionData -> ParamsForVnf UpdateNsRequest -> AssocNewNsdVersionData -> DeployableModulesInConstitue ntVnf	Update NS (see SOL005)

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
		UpdateNsRequest -> ChangeVnfFlavourData UpdateNsRequest -> ChangeVnfPackageData UpdateNsRequest -> ChangeNsFlavourData -> DeployableModulesInConstituentVnf UpdateNsRequest -> SelectVnfDeployableModuleData	
		NsInstance -> VnfInstance	Query NS (see SOL005)
Node template name of type tosca.nodes.nfv.PaaS ServiceRequest	(IdentifierinVnfd) paasServiceRequestId	VnfInstance -> PaasServiceInfo	Query VNF (see SOL002 and SOL003)
	(IdentifierinVnfd) vnfPaasServiceRequestId	NsInstance -> VnfInstance -> instantiatedVnfInfo -> PaasServiceInfo	Query NS (see SOL005)
tosca.datatypes.nfv.CertificateBaseProfile -> id	(Identifier) id	CreateVnfRequest -> CertificateConfigurationData -> CertificateBaseProfile	Create VNF identifier (see SOL002 and SOL003)
		ChangeVnfFlavourRequest -> CertificateConfigurationData -> CertificateBaseProfile	Change VNF flavour (see SOL002 and SOL003)
		ChangeExtVnfConnectivityRequest -> CertificateConfigurationData -> CertificateBaseProfile	Change external VNF connectivity (see SOL002 and SOL003)
		ChangeCurrentVnfPkgRequest -> CertificateConfigurationData -> CertificateBaseProfile	Change current VNF package (see SOL002 and SOL003)
NOTE 1: The entry "SOL002" in the "Operation" column refers to ETSI GS NFV-SOL 002 [22], "SOL003" refers to ETSI GS NFV-SOL 003 [25] and "SOL005" refers to ETSI GS NFV-SOL 005 [i.10].			
NOTE 2: This is an empty base type to be extended to a VNF specific type per LCM operation. The extended VNF specific and LCM operation specific type is the one that actually maps to the additionalParams in the API.			
NOTE 3: Only when VduCp is re-exposed as VnfExtCp.			
NOTE 4: Additional mappings corresponding to input parameter sets of VNF LCM operations in UpdateNS requests may be added in future versions of the present document.			
NOTE 5: The present document specifies that these properties may be overridden at runtime. However, ETSI GS NFV-SOL 002 [22] and ETSI GS NFV-SOL 003 [25] do not define standard attributes to provide values to override these.			

D.3 NSD-related constructs

Table D.3-1 provides the mapping between the properties of TOSCA types related to the NSD and API attributes, which include: resource or notification data type (and referenced structured data type when available), attribute name and type in the resource or notification data type, and the interface operation in which the data type is used.

NOTE: In the "Data model" column of the table, an arrow "->" indicates the navigation through the resource, notification and referenced structured data types.

Table D.3-1: Mapping of API attributes and TOSCA constructs

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
tosca.nodes.nfv.NS -> descriptor_id	(Identifier) nsdIds	LccnSubscriptionRequest -> LifecycleChangeNotificationsFilter -> NsInstanceSubscriptionFilter	Subscription to NS LCM notifications (see SOL005)
	(Identifier) nsdId	NsdInfo	Create NSD Update NSD Query/read NSD (see SOL005)
		NsdOnboardingNotification	Notification about NSD management (see SOL005)
		NsdOnboardingFailureNotification	
		NsdChangeNotification	
		NsdDeletionNotification	Subscription about NSD management (see SOL005)
		NsdmSubscriptionRequest -> NsdmNotificationsFilter	
		NsdmSubscription -> NsdmNotificationsFilter	
		CreateNsRequest	Create NS (see SOL005)
		NsInstance	Create NS Query NS Delete NS (see SOL005)
		NsLcmOpOcc -> AffectedNs	Query/read about NS LCM operation occurrence (see SOL005)
		NsLcmOperationOccurrenceNotificatio n -> AffectedNs	Notification about NS LCM (see SOL005)
tosca.nodes.nfv.NS -> name	(String) nsdName	UpdateNsRequest -> AssocNewNsdVersionData	Update NS (see SOL005)
		NsdInfo	Create NSD Update NSD Query/read NSD (see SOL005)
		NsdmSubscriptionRequest -> NsdmNotificationsFilter NsdmSubscription -> NsdmNotificationsFilter	Subscription about NSD management (see SOL005)
tosca.nodes.nfv.NS -> version	(Version) nsdVersion	NsdInfo	Create NSD Update NSD Query/read NSD (see SOL005)
		NsdmSubscriptionRequest -> NsdmNotificationsFilter	Subscription about NSD management (see SOL005)
		NsdmSubscription -> NsdmNotificationsFilter	
tosca.nodes.nfv.NS -> designer	(String) nsdDesigner	NsdInfo	Create NSD Update NSD Query/read NSD (see SOL005)
		NsdmSubscriptionRequest -> NsdmNotificationsFilter	Subscription about NSD management (see SOL005)
		NsdmSubscription -> NsdmNotificationsFilter	
tosca.nodes.nfv.NS -> invariant_id	(Identifier) nsdInvariantId	NsdInfo	Create NSD Update NSD Query/read NSD (see SOL005)

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
		NsdmSubscriptionRequest -> NsdmNotificationsFilter	Subscription about NSD management (see SOL005)
		NsdmSubscription -> NsdmNotificationsFilter	
tosca.nodes.nfv.NS -> flavour_id	(IdentifierInNsd) flavourId	NsInstance	Create NS Query NS Delete NS (see SOL005)
	(IdentifierInNsd) nsFlavourId	InstantiateNsRequest	Instantiate NS (see SOL005)
	(IdentifierInNsd) newNsFlavourId	UpdateNsRequest -> ChangeNsFlavourData	Update NS (see SOL005)
Node template name of type tosca.nodes.nfv.NsVirtual Link	(IdentifierInNsd) nsVirtualLinkDescId	NsLcmOperationOccurrenceNotificatio n -> AffectedVirtualLink	Notification about NS LCM (see SOL005)
		NsInstance -> NsVirtualLinkInfo	Create NS Query NS Delete NS (see SOL005)
(No related property in type "tosca.datatype.nfv.NsVIP rofile") See note 2	(IdentifierInNsd) nsVirtualLinkProfileId	NsInstance -> NsVirtualLinkInfo	Create NS Query NS Delete NS (see SOL005)
	(IdentifierInNsd) vIProfileId	NsLcmOperationOccurrenceNotificatio n -> AffectedVirtualLink	Notification about NS LCM (see SOL005)
See note 2	(IdentifierInNsd) nsInstantiationLevelI d	InstantiateNsRequest	Instantiate NS (see SOL005)
	(IdentifierInNsd) instantiationLevelId	UpdateNsRequest -> ChangeNsFlavourData	Update NS (see SOL005)
	(IdentifierInNsd) nsInstantiationLevel	ScaleNsRequest -> ScaleNsData -> ScaleNsToLevelData	Scale NS (see SOL005)
Node template name in NSD of type derived from tosca.nodes.nfv.NS	(IdentifierInNsd) nsProfileId	UpdateNsRequest -> NestedNsInstanceData	Update NS (see SOL005)
		InstantiateNsRequest -> NestedNsInstanceData	Instantiate NS (see SOL005)
		InstantiateNsRequest -> ParamsForNestedNs	
Node template name in NSD of type derived from tosca.nodes.nfv.VNF	(IdentifierInNsd) vnfProfileId	NsLcmOpOcc -> AffectedVnf	Query/read information about NS LCM (see SOL005)
		NsLcmOperationOccurrenceNotificatio n -> AffectedVnf	Notification about NS LCM (see SOL005)
		NsInstance -> NsAffinityOrAntiAffinityRule	Create NS Query NS Delete NS (see SOL005)
		InstantiateNsRequest -> NsAffinityOrAntiAffinityRule	Instantiate NS (see SOL005)
		InstantiateNsRequest -> ParamsForVnf	
		InstantiateNsRequest -> VnfLocationConstraint	
		InstantiateNsRequest -> VnfInstanceData	
		ScaleNsRequest -> ScaleNsData -> VnfLocationConstraint	Scale NS (see SOL005)

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
tosca.nodes.nfv.PNF -> descriptor_id		ScaleNsRequest -> ScaleNsData -> VnfInstanceData	
		UpdateNsRequest -> VnfInstanceData	Update NS (see SOL005)
	(Identifier) pnfdIds	LccnSubscriptionRequest -> LifecycleChangeNotificationsFilter -> NsInstanceSubscriptionFilter	Subscription to NS LCM (see SOL005)
	(Identifier) pnfdId	PnfdOnboardingNotification	Notification about NSD management (see SOL005)
		PnfdOnboardingFailureNotification	
		PnfdDeletionNotification	
		NsLcmOpOcc -> AffectedPnf	Query/read information about NS LCM (see SOL005)
		NsLcmOperationOccurrenceNotification -> AffectedPnf	Notification about NS LCM (see SOL005)
		NsInstance -> PnfInfo	Create NS Query NS Delete NS (see SOL005)
		InstantiateNsRequest -> AddPnfData	Instantiate NS (see SOL005)
		UpdateNsRequest -> AddPnfData	Update NS (see SOL005)
tosca.nodes.nfv.PNF -> name	(String) pnfdName	NsInstance -> PnfInfo	Create NS Query NS Delete NS (see SOL005)
tosca.nodes.nfv.PNF -> version	(Version) pnfdVersion	NsInstance -> PnfInfo	Create NS Query NS Delete NS (see SOL005)
tosca.nodes.nfv.PNF -> provider	(String) pnfdProvider	NsInstance -> PnfInfo	Create NS Query NS Delete NS (see SOL005)
tosca.nodes.nfv.PNF -> descriptor_invariant_id	(Identifier) pnfdInvariantId	NsInstance -> PnfInfo	Create NS Query NS Delete NS (see SOL005)
Node template name in NSD of type derived from tosca.nodes.nfv.PNF	(IdentifierInNsd) pnfdProfileId	NsLcmOpOcc -> AffectedPnf	Query/read information about NS LCM Fail NS LCM (see SOL005)
		NsLcmOperationOccurrenceNotification -> AffectedPnf	Notification about NS LCM (see SOL005)
		NsInstance -> PnfInfo	Create NS Query NS Delete NS (see SOL005)
		InstantiateNsRequest -> AddPnfData	Instantiate NS (see SOL005)
		UpdateNsRequest -> AddPnfData	Update NS (see SOL005)
Node template name of type toscanodes.nfv.Cp	(IdentifierInNsd) cpdId	InstantiateNsRequest -> AddPnfData -> PnfExtCpData	Instantiate NS (see SOL005)
		UpdateNsRequest -> AddPnfData -> PnfExtCpData	Update NS (see SOL005)

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
		UpdateNsRequest -> ModifyPnfData -> PnfExtCpData	
		NsInstance -> PnfInfo -> PnfExtCpInfo	Create NS Query NS Delete NS (see SOL005)
Group template name of type tosca.groups.nfv.VNFFG	(IdentifierInNsd) vnffgId	NsLcmOpOcc -> AffectedVnffg	Query/read information about NS LCM Fail NS LCM (see SOL005)
		NsLcmOperationOccurrenceNotificatio n -> AffectedVnffg	Notification about NS LCM (see SOL005)
		UpdateNsRequest -> AddVnffgData	Update NS (see SOL005)
		UpdateNsRequest -> UpdateVnffgData -> VnffgInfo	
See note 2	(IdentifierInNsd) aspectId	ScaleNsRequest -> ScaleNsData -> ScaleNsByStepsData	Scale NS (see SOL005)
	(IdentifierInNsd) nsScalingAspectId	ScaleNsRequest -> ScaleNsData -> ScaleNsToLevelData -> NsScaleInfo	
Name of the policy of type tosca.policies.nfv.NsMonitoring	(IdentifierInNsd) id	NsInstance -> NsMonitoringParameter	Create NS Query NS Delete NS (see SOL005)
tosca.policies.nfv.NsMonitoring -> ns_monitoring_parameters -> name	(String) name	NsInstance -> NsMonitoringParameter	Create NS Query NS Delete NS (see SOL005)
See note 2	(Identifier) vnfPkgIds	NsdInfo	Create NSD Update NSD Query/read NSD (see SOL005)
		NsdmSubscriptionRequest -> NsdmNotificationsFilter	Subscription about NSD management (see SOL005)
		NsdmSubscription -> NsdmNotificationsFilter	
See note 2	(IdentifierInNsd) healScript	HealNsRequest -> HealNsData	Heal NS (see SOL005)
See note 2	(IdentifierInNsd) aspectId	ScaleNsRequest -> ScaleNsData -> ScaleNsByStepsData	Scale NS (see SOL005)
	(IdentifierInNsd) nsScalingAspectId	ScaleNsRequest -> ScaleNsData -> ScaleNsToLevelData -> NsScaleInfo	
Node template name of type tosa.nodes.nfv.Sap	(IdentifierInNsd) sapId	NsInstance -> SapInfo	Query NS (see SOL005)
		InstantiateNsRequest -> SapData	Instantiate NS (see SOL005)
		UpdateNsRequest -> SapData	Update NS (see SOL005)
		NsLcmOpOcc -> AffectedSap	Query/read information about NS LCM operation occurrence (see SOL005)

ETSI GS NFV-SOL 001 (the present document)	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note 1)
		NsLcmOperationOccurrenceNotification -> AffectedSap	Notification about NS LCM (see SOL005)
tosca.policies.nfv.DataFlowInfo -> data_flow_info_id	(IdentifierInNsd) dataFlowInfoId	InstantiateNsRequest -> DataFlowMirroringData -> DataFlowData	Instantiate NS (see SOL005)
		UpdateNsRequest -> CreateDataFlowMirroring -> DataFlowMirroringData -> DataFlowData	Update NS (see SOL005)
		NsInstance -> MirroringInfo -> DataFlowData	Query NS (see SOL005)
Node template name of type tosca.nodes.nfv.NsPaasServiceRequest	(IdentifierInNsd) nsPaasServiceRequestId	NsInstance -> PaasServiceInfo	Query NS (see SOL005)
NOTE 1: The entry "SOL005" refers to ETSI GS NFV-SOL 005 [i.10]. NOTE 2: The corresponding TOSCA construct is not included in the present document, the mapping may be updated in future versions of the present document.			

Annex E (informative): TOSCA Imperative workflows

E.1 Purpose

This annex specifies TOSCA Imperative workflows for the NSD and the VNFD by using the grammar defined in TOSCA Simple Profile-YAML-v1.3 [20].

E.2 TOSCA Imperative workflows for the NSD

E.2.1 Introduction

TOSCA Imperative workflows based on TOSCA-Simple-Profile-YAML-v1.3 [20] may be used by the NFVO to fulfil the NS LCM operations described in ETSI GS NFV-IFA 013 [i.8]. TOSCA Imperative workflows provide an additional method for implementation of LCM operations in the Nslcm interface defined in clause 7.7.1.1 of the present document.

NOTE: Even if TOSCA Imperative workflows is described in the NSD, the NFVO will still process the NSD with Nslcm operations as defined in clause 7.7.1.1. Since this is an additional method for implementation of NS LCM operations, execution of workflows instead of NS LCM operations is optional and up to the NFVO implementation.

TOSCA Imperative workflows defined in the NSD describe procedures for the NFVO to manage the lifecycle of network services.

Workflows are comprised of steps associated with the NS LCM operations and additional steps that are preamble and postamble to the execution of the former steps. The name of the preamble and postamble steps is constructed according to the following pattern:

- `<NS_LCM_base_operation_workflow_name>_start_<step_name>` for preamble steps
- `<NS_LCM_base_operation_workflow_name>_end_<step_name>` for postamble steps

Preamble steps are specified before the execution of workflow steps. Postamble steps are specified after the execution of workflow steps.

External and internal stimuli described in clause 7.7.1.4 of the present document, are mapped to workflows as below:

- External stimuli are mapped to TOSCA Imperative workflows, i.e. `<NS_LCM_base_operation_workflow_name>`
- Internal stimuli are mapped to preamble and postamble steps of the workflow

E.2.2 Definition of an NS workflow

The syntax of TOSCA Imperative workflows for LCM operations on the NS has the following definition:

```
workflows:
  description: TOSCA Imperative workflows corresponding to NS LCM operations
  defined in ETSI GS NFV-IFA 013.
  instantiate:
    description: This workflow is invoked upon receipt of an Instantiate NS
    request
    # inputs:
    steps:
      instantiate_start_<step_name>: # Invoked before steps for instantiate LCM
    operation
```

```

        # steps for instantiate workflow
        instantiate_end_<step_name> : # Invoked after steps for instantiate LCM
operation

    terminate:
        description: This workflow is invoked upon receipt of Terminate NS request
        # inputs:
        steps:
            terminate_start_<step_name> : # Invoked before steps for terminate LCM
operation
        # steps for terminate workflow
        terminate_end_<step_name>: # Invoked after steps for terminate LCM operation

```

E.2.3 Examples

The following example template fragment, based on clause A.8 of the present document, illustrates the use of TOSCA Imperative workflows for NS LCM operations.

When the NFVO executes TOSCA Imperative workflows in the NSD, it uses standard APIs for LCM operations defined in the Or-Vnfm interface and delegates the task to VNFM; The VNFM in turn executes corresponding TOSCA Operations on the VNF, as explained in clause 6.7.1 of the present document.

NOTE 1: The NSD consumer makes available all parameters from the message invoking the NS base LCM operation as inputs to the corresponding TOSCA workflows. The additional parameters for NS base LCM operations are defined as workflow inputs.

NOTE 2: It is out of scope of the present document to specify mapping of SOL003/SOL005 API execution results with the success and failure of workflows.

```

tosca_definitions_version: tosca_simple_yaml_1_3

description: NS TOSCA Imperative Workflows

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of NSD related TOSCA types as defined in ETSI
GS NFV-SOL 001
  - example_vnf1.yaml
  - example_vnf2.yaml

data_types:
  MyCompany.datatypes.nfv.NsInstantiateAdditionalParameters:
    derived_from: tosca.datatypes.nfv.NsOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
        required: true
        default: value_1
      parameter_2:
        type: string
        required: true
        default: value_2

node_types:
  tosca.example_NS:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]

```

```

    default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
  designer:
    type: string
    constraints: [ equal: MyCompany ]
    default: MyCompany
  name:
    type: string
    constraints: [ equal: ExampleService ]
    default: ExampleService
  version:
    type: string
    constraints: [ equal: '1.0' ]
    default: '1.0'
  invariant_id:
    type: string
    constraints: [ equal: 1111-2222-aaaa-bbbb ]
    default: 1111-2222-aaaa-bbbb
  flavour_id:
    type: string
    constraints: [ equal: simple ]
    default: simple

topology_template:
  substitution_mappings:
    node_type: tosca.example_NS
  requirements:
    virtual_link: [ VNF_2, virtual_link_2 ] # the External connection point of
                                           # VNF_2 is exposed as the Sap

  node_templates:
# This abstract node template enables the NSD author to use Nslcm scripts if he does
#not use workflows.
    my_service:
      type: tosca.example_NS
      properties:
      interfaces:
        Nslcm:
          operations:
            instantiate:
              implementation: instantiate.workflow.yaml
            terminate:
              implementation: terminate.workflow.yaml

    VNF_1:
      type: tosca.nodes.nfv.example_VNF1
      properties:
        # no property assignments needed for required properties that have a default
#value assigned in the node type definition, e.g. descriptor_id
        vnf_profile:
          instantiation_level: level_1
          min_number_of_instances: 2
          max_number_of_instances: 6
      requirements:
        - virtual_link: NsVirtualLink_1
# Additional parameters input to be defined in the VNFD of VNF_1.
#   interfaces:
#     Vnflcm:
#       operations:
#         instantiate: . . .
#         terminate: . . .

```

```

VNF_2:
  type: toska.nodes.nfv.example_VNF2
  properties:
    vnf_profile:
      instantiation_level: level_1
      min_number_of_instances: 1
      max_number_of_instances: 3
  requirements:
    - virtual_link_1: NsVirtualLink_1
#    - virtual_link_2: # map to virtual_link requirement of the NS node
    - dependency: VNF_1

# Additional parameters input to be defined in the VNFD of VNF_2.
#   interfaces:
#     Vnflcm:
#       operations:
#         instantiate: . . .
#         terminate: . . .

NsVirtualLink_1:
  type: toska.nodes.nfv.NsVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ipv4]
      flow_pattern: mesh
    vl_profile:
      max_bitrate_requirements:
        root: 1000
      min_bitrate_requirements:
        root: 1000

workflows:
  instantiate: #instantiate workflow
  inputs:
    additional_parameters:
      type: MyCompany.datatypes.nfv.NsInstantiateAdditionalParameters
      required: false
  steps:
    # preamble steps for instantiate operation. These correspond to preparatory
#steps internal to the NFVO before instantiate operation

    instantiate_start_step_1:
      #. . .
      target: my_service
      activities: []
      on_success:
        - create_VNF_1

    # steps for instantiate workflow
    create_VNF_1: # Step: Instantiate VNF_1
      target: VNF_1
      activities:
        - call_operation: Vnflcm.instantiate
# invoking Vnflcm.instantiate operation enables NFVO to use internal implementation of
#Vnflcm.instantiate operation which results in an ETSI GS NFV-SOL 003 API call towards
#the VNFM to call VnfInstantiate operation. This enables VNFM to execute LCM operations
#to deploy VNF_1
      on_success:
        - create_VNF_2

#   . . .

```

```

    create_VNF_2:  # Step: Instantiate VNF_2
      target: VNF_2
      activities:
        - call_operation: Vnflcm.instantiate
      # invoking Vnflcm.instantiate operation enables NFVO to use internal implementation of
      #Vnflcm.instantiate operation which results in an ETSI GS NFV-SOL 003 API call towards
      #the VNFM to call VnfInstantiate operation. This enables VNFM to execute LCM operations
      #to deploy VNF_2
      on_success:
        - instantiate_end_step_1
        #. . .
      # postamble steps for instantiate operation. These correspond to closing steps
      #internal to the NFVO after instantiate operation.

    instantiate_end_step_1:
      #. . .
      target: my_service
      activities: []

    terminate: #terminate workflow
    steps:
      # preamble steps for terminate operation. These correspond to preparatory
      #steps internal to the NFVO before terminate operation.
      terminate_start_step_1:
        #. . .
        target: my_service
        activities: []
        on_success:
          - terminate_VNF_2

      # steps for terminate workflow
      terminate_VNF_1:  # Step: Terminate VNF_1
        target: VNF_1
        activities:
          - call_operation: Vnflcm.terminate
      # invoking Vnflcm.terminate operation enables NFVO to use internal implementation of
      #Vnflcm.terminate operation which results in an ETSI GS NFV-SOL 003 API call towards
      #the VNFM to call VnfTerminate operation. This enables VNFM to execute LCM operations
      #to terminate VNF_1
      on_success:
        - terminate_VNF_1
      #. . .

      terminate_VNF_2:  # Step: Terminate VNF_2
        target: VNF_2
        activities:
          - call_operation: Vnflcm.terminate
      # invoking Vnflcm.terminate operation enables NFVO to use internal implementation of
      #Vnflcm.terminate operation which results in an ETSI GS NFV-SOL 003 API call towards
      #the VNFM to call VnfTerminate operation. This enables VNFM to execute LCM operations
      #to terminate VNF_2
      on_success:
        - terminate_end_step_1
        #. . .
      # post amble steps for terminate operation. These correspond to closing steps
      #internal to the NFVO after terminate operation.

      terminate_end_step_1:
        target: my_service
        activities: []

```


#. . .

NOTE 3: As the `on_success` keyword is not used between steps inside the workflow for NS LCM base operation, the order of execution is decided by the NFVO.

NOTE 4: As the `on_failure` keyword is not present inside the workflow for NS LCM base operation, the error handling is decided by the NFVO.

Annex F (informative): Non-Backward Compatible Changes in the GS

F.1 Introduction

This annex provides the list of non-backward compatible changes during the development of the present document.

A change introduced in version n of the present document is non-backward compatible if a service template written according to a previous version n-1 of the present document, i.e. a service template that has not been updated according to this change, is invalid with respect to this change for a NFVO/VNFM compliant to version n of the present document.

This annex focuses on compatibility from a descriptor view point. It does not evaluate whether a change made to the present document leads to non-backward compatible changes on the APIs referenced in Annex D of the present document.

F.2 Non-Backward Compatible changes between version 2.6.1 and 2.7.1

Table F.2-1 provides a list of non-backward compatible changes between version 2.6.1 [i.22] and version 2.7.1 [i.19] of the present document.

Table F.2-1: Non-backward compatible changes

No.	Description	Clause
1	nfvi_constraints type changed from list of string to map of string.	6.8.3
2	The definition of SecurityGroupRule is changed in version 2.7.1, a new policy type toasca.policies.nfv.Abstract.SecurityGroupRule is introduced in the definition, which SecurityGroupRule policy is derived from.	6.10.13
3	PlacementGroup used to be applied for both VNFD and NSD in version 2.6.1, but in version 2.7.1, it is only applied to VNFD.	6.9
4	AffinityRule, AntiAffinityRule used to be applied for both VNFD, NSD in version 2.6.1, but in version 2.7.1, it is only applied to VNFD.	6.10.10
5	SecurityGroupRule policy type used to be applied for VNFD, PNFD and NSD in version 2.6.1, but in version 2.7.1, it is only applied to VNFD.	6.10.13
6	boot_order type changed from list of string to Boolean.	6.8.3
7	boot_data type changed from string to BootData data type.	6.8.3

F.3 Non-Backward Compatible changes between version 2.8.1 and 3.3.1

Table F.3-1 provides a list of non-backward compatible changes between version 2.8.1 [i.20] and version 3.3.1 [i.19] of the present document.

Table F.3-1: Non-backward compatible changes

No.	Description	Clause
1	Multiple VNF deployment flavour design changed from using TOSCA v 1.2 grammar to TOSCA v 1.3 grammar: - in the top level service templates, removing the imports for lower level service templates. See note; - in the low level service templates, use of substitution filter instead of property mapping.	6.11.2
2	Multiple NS deployment flavour design changed from using TOSCA v 1.2 grammar to TOSCA v 1.3 grammar: - in the top level service templates, removing the imports for lower level service templates. See note; - in the low level service templates, use of substitution filter instead of property mapping.	7.11.2
NOTE: Instead the low level service templates are declared in the Other-Definitions of the TOSCA.meta file as specified in TOSCA-Simple-Profile-YAML-v1.3 [20].		

F.4 Non-Backward Compatible changes between version 3.3.1 and 3.5.1

Table F.4-1 provides a list of non-backward compatible changes between version 3.3.1 [i.19] and version 3.5.1 [i.21] of the present document.

Table F.4-1: Non-backward compatible changes

No.	Description	Clause
1	In VNF, Vdu.Compute and VnfVirtualLink node type definition, the type of <u>monitoring_parameters</u> changed from list to map.	6.8.1, 6.8.3, 6.8.9
2	In NsMonitoring policy definition, the type of <u>ns_monitoring_parameters</u> changed from list to map.	7.10.4
3	In VnfMonitoring policy definition, the type of <u>vnf_monitoring_parameters</u> changed from list to map.	7.10.5

F.5 Non-Backward Compatible changes between version 3.5.1 and 4.2.1

None.

F.6 Non-Backward Compatible changes between version 4.2.1 and 4.3.1

Table F.6-1 provides a list of non-backward compatible changes between version 4.2.1 [i.23] and version 4.3.1 [i.25] of the present document.

Table F.6-1: Non-backward compatible changes

No.	Description	Clause
1	In the VirtualFileStorageData data type, constraints have been added on the file_system_protocol property values.	6.2.41
2	In VnfConfigurableProperties data type definition, the type of vnfm_interface_info changed to a list.	6.2.31
3	In VirtualNetworkInterfaceRequirements data type definition, support_mandatory property is removed.	6.2.4

F.7 Non-Backward Compatible changes between version 4.4.1 and 4.3.1

Table F.7-1 provides a list of non-backward compatible changes between version 4.3.1 [i.25] and version 4.4.1 of the present document.

Table F.7-1: Non-backward compatible changes

No.	Description	Clause
1	In the AdditionalServiceData data type, the type of serviceData changes from string to toasca.datatypes.nfv.ServiceData.	6.2.66

Annex G (informative): Change History

Date	Version	Information about changes
2016.05	0.0.1	Implemented NFVSOL(16)000005r1 GS NFV SOL001 ToC
2016.07	0.02	Implemented NFVSOL(16)000026r1, NFVSOL(16)000027r1, NFVSOL(16)000028r2
2017.09	0.1.0	Implemented NFVSOL(17)000539r3, NFVSOL(17)000540r3, NFVSOL(17)000542r2, NFVSOL(17)000544r1
2017.10	0.2.0	Implemented NFVSOL(17)000543r3, NFVSOL(17)000566, NFVSOL(17)000567r1, NFVSOL(17)000568r1, NFVSOL(17)000569 Editorial modification for clause numbering and format
2017.11	0.3.0	Clean-up done by <i>editHelp</i> Implemented NFVSOL(17)000545r4, NFVSOL(17)000559r2, NFVSOL(17)000560r2, NFVSOL(17)000570r1, NFVSOL(17)000575r1, NFVSOL(17)000616r1, NFVSOL(17)000641
2017.12	0.4.0	Implemented NFVSOL(17)000642r1, NFVSOL(17)000675r1, NFVSOL(17)000756
2018.01	0.5.0	Implemented NFVSOL(17)000621r6, NFVSOL(17)000676r3, NFVSOL(17)000677r2, NFVSOL(17)000736r1, NFVSOL(18)000004r2
2018.03	0.6.0	Implemented NFVSOL(18)000049, NFVSOL(18)000025r1, NFVSOL(18)00048R1, NFVSOL(18)00040, NFVSOL(18)00023r2, NFVSOL(18)00024r2, NFVSOL(18)00029r1, NFVSOL(18)00063r5, NFVSOL(18)000077, NFVSOL(18)000046r2, NFVSOL(18)000074r2, NFVSOL(17)0000611r5, NFVSOL(18)000041r1, NFVSOL(18)000038r3, NFVSOL(18)000028, NFVSOL(18)000044r2, NFVSOL(18)000055r2, NFVSOL(18)000045r1
2018.03	0.6.1	Implemented NFVSOL(18)000052r1, NFVSOL(18)000094r5, NFVSOL(18)000043r3, NFVSOL(18)0000112r1, NFVSOL(18)0000117r2, NFVSOL(18)0000124r1, NFVSOL(18)0000129r1, NFVSOL(18)000042r2, NFVSOL(18)0000115r2
2018.04	0.6.2	Implemented NFVSOL(18)000113r2, NFVSOL(18)000119r2, NFVSOL(18)000121, NFVSOL(18)000135, NFVSOL(18)000116r1, NFVSOL(18)000157, NFVSOL(18)000158r1
2018.05	0.6.3	Implemented NFVSOL(18)000168, NFVSOL(18)000169r2, NFVSOL(18)000173, NFVSOL(18)000174r2
2018.05	0.7.0	Implemented NFVSOL(18)000156r7, NFVSOL(18)000142r3, NFVSOL(18)000147, NFVSOL(18)000193r2, NFVSOL(18)000201, NFVSOL(18)000202, NFVSOL(18)000203r2, NFVSOL(18)000205r1, NFVSOL(18)000160r1, NFVSOL(18)000199r2, NFVSOL(18)000200r3, NFVSOL(18)000192r1, NFVSOL(18)000194r2, NFVSOL(18)000231r1, NFVSOL(18)000223r1, NFVSOL(18)000183r2
2018.06	0.8.0	Implemented NFVSOL(18)000287, NFVSOL(18)00012r11, NFVSOL(18)000197r2, NFVSOL(18)000198r3, NFVSOL(18)000286, NFVSOL(18)000292, NFVSOL(18)000294r1, NFVSOL(18)000295r1, NFVSOL(18)000301r2, NFVSOL(18)000302r2, NFVSOL(18)000240r1, NFVSOL(18)000253r3, NFVSOL(18)000254r2, NFVSOL(18)000256r2
2018.06	0.9.0	Implemented NFVSOL(18)000288r2 Adding etsi_nfv_sol001_vnfd_0_9_0_type.yaml and SOL001 Graphics v0_9_0.pptx in the draft GS zip package Editorial changes for all the TOSCA type definitions
2018.08	0.10.0	Implemented NFVSOL(18)000331r4, NFVSOL(18)000335r2, NFVSOL(18)000351r1, NFVSOL(18)000374r1, NFVSOL(18)000376, NFVSOL(18)000382, NFVSOL(18)000395r1, NFVSOL(18)000402r2, NFVSOL(18)000404, NFVSOL(18)000405r2, NFVSOL(18)000406r1, NFVSOL(18)000408r2, NFVSOL(18)000409r1, NFVSOL(18)000411r1, NFVSOL(18)000413, NFVSOL(18)000416r2, NFVSOL(18)000422r4, NFVSOL(18)000423r4, NFVSOL(18)000424r2, NFVSOL(18)000425r2, NFVSOL(18)000430r1, NFVSOL(18)000336r4, NFVSOL(18)000380, NFVSOL(18)000387r2, NFVSOL(18)000394r1, NFVSOL(18)000410r2, NFVSOL(18)000447r3, NFVSOL(18)000427, NFVSOL(18)000403r3, NFVSOL(18)000420r4, NFVSOL(18)000393r1, NFVSOL(18)000415, NFVSOL(18)000398r1, NFVSOL(18)000399r3, NFVSOL(18)000414r2, NFVSOL(18)000379r3, NFVSOL(18)000428r5, NFVSOL(18)000429r4, NFVSOL(18)000465r1, NFVSOL(18)000479, NFVSOL(18)000375r4, NFVSOL(18)000488r2, NFVSOL(18)000492r1

Date	Version	Information about changes
2018.09	0.11.0	Implemented NFVSOL(18)000486r7, NFVSOL(18)000495r1, NFVSOL(18)000497r3, NFVSOL(18)000498, NFVSOL(18)000500r3, NFVSOL(18)000503r1, NFVSOL(18)000504r1, NFVSOL(18)000505r1, NFVSOL(18)000508, NFVSOL(18)000514r2, NFVSOL(18)000515r1, NFVSOL(18)000516r1, NFVSOL(18)000524r1, NFVSOL(18)000529r2, NFVSOL(18)000530r1, NFVSOL(18)000536r1, NFVSOL(18)000538r1, NFVSOL(18)000541r3, NFVSOL(18)000544r1, NFVSOL(18)000545r1
2018.10	0.12.0	Implemented NFVSOL(18)000507r3, NFVSOL(18)000621r4, NFVSOL(18)000547r1, NFVSOL(18)000567r1, NFVSOL(18)000574, NFVSOL(18)000579, NFVSOL(18)000590r1
2018.11	0.13.0	Implemented NFVSOL(18)000563r5, NFVSOL(18)000575r5, NFVSOL(18)000586r2, NFVSOL(18)000587r1, NFVSOL(18)000589r2, NFVSOL(18)000591, NFVSOL(18)000592, NFVSOL(18)000604r2, NFVSOL(18)000606r1, NFVSOL(18)000607r5, NFVSOL(18)000608, NFVSOL(18)000609, NFVSOL(18)000612, NFVSOL(18)000614r1, NFVSOL(18)000615, NFVSOL(18)000616r2, NFVSOL(18)000617r3, NFVSOL(18)000619r3, NFVSOL(18)000620r5, NFVSOL(18)000628r2, NFVSOL(18)000629r1, NFVSOL(18)000630, NFVSOL(18)000631r1, NFVSOL(18)000632, NFVSOL(18)000634, NFVSOL(18)000635, NFVSOL(18)000636r3, NFVSOL(18)000637, NFVSOL(18)000655r2, NFVSOL(18)000658r3, NFVSOL(18)000659, NFVSOL(18)000660, NFVSOL(18)000665r1, NFVSOL(18)000666, NFVSOL(18)000678r1, NFVSOL(18)000679, NFVSOL(18)000682, NFVSOL(18)000684r1, NFVSOL(18)000687
2019.03	2.5.2	Implemented NFVSOL(19)000063r4, NFVSOL(19)000068r1, NFVSOL(19)000069r2, NFVSOL(19)000070r1, NFVSOL(19)000080, NFVSOL(19)000107r1, NFVSOL(19)000120r1, NFVSOL(19)000121, NFVSOL(19)00039r5, NFVSOL(19)00087r2, NFVSOL(19)00067r5, NFVSOL(19)00077, NFVSOL(19)00082r3, NFVSOL(19)00085, NFVSOL(19)00086, NFVSOL(19)000106r1
2019.03	2.5.3	Implemented NFVSOL(19)000084r4, NFVSOL(19)0000101r2, NFVSOL(19)0000165, NFVSOL(19)0000166, NFVSOL(19)0000167r1, NFVSOL(19)0000170r4, NFVSOL(19)0000173, NFVSOL(19)0000163, NFVSOL(19)0000119
2019.03.22	2.5.4	Editorial modification made by ETSI Secretariat allowing to structure the SOL repository on ETSI Forge in a future proof and maintainable way: • Forge structure updated: "v2.6.1" tag created • Yaml filenames updated: version numbers removed from filenames (still included in file header) • Import statements updated: version number removed from imported filenames. • Draft updated: – Updated all references to yaml files – Updated the forge URLs – In B.1: removed the sentence explaining the meaning of x_y_z_ in the filename structure
2019.04.23	2.5.5	2 comments were raised during the Remote Consensus approval: both requesting to implement the WG SOL approved Change Request in NFVSOL(19)000229r3 onto the final SOL001 draft (see these 2 comments in the RC report). The present version implements NFVSOL(19)000229r3: adding machine readable meta info inside the yaml file indicating the SOL001 release version to which they apply + other editorial changes
2019.05	2.6.2	Implemented NFVSOL(19)000162r1, NFVSOL(19)000194r2, NFVSOL(19)000222r2, NFVSOL(19)000241r2, NFVSOL(19)000242r2
2019.06	2.6.3	Implemented NFVSOL(19)000160r9, NFVSOL(19)000239r3, NFVSOL(19)000248, NFVSOL(19)000262r2, NFVSOL(19)000263r3, NFVSOL(19)000268r3, NFVSOL(19)000269, NFVSOL(19)000270r3, NFVSOL(19)000303r1, NFVSOL(19)000307, NFVSOL(19)000279, NFVSOL(19)000280, NFVSOL(19)000296r3, NFVSOL(19)000301r1, NFVSOL(19)000305r1, NFVSOL(19)000338, NFVSOL(19)000340, NFVSOL(19)000342r1, NFVSOL(19)000344r2, NFVSOL(19)000345r4, NFVSOL(19)000266r7
2019.08	2.6.4	Implemented NFVSOL(19)000325r8, NFVSOL(19)000346r6, NFVSOL(19)000347r2, NFVSOL(19)000380r1, NFVSOL(19)000383r1, NFVSOL(19)000384r1, NFVSOL(19)000385, NFVSOL(19)000389r1, NFVSOL(19)000428r2, NFVSOL(19)000449
2019.09	2.6.5	Implemented NFVSOL(19)000386r8, NFVSOL(19)000408r4, NFVSOL(19)000451r8, NFVSOL(19)000559

Date	Version	Information about changes
2019.11	2.6.6	Implemented NFVSOL(19)000727r1_SOL001Ed271_Miscellaneous_corrections, NFVSOL(19)000642r1_SOL001ed271_Annex_Mapping_table_for_SOL_API_NSD_related_cons, NFVSOL(19)000645r3_SOL001ed271_NBWC_issue_list_annex, NFVSOL(19)000700r1_SOL001ed271_updating_mapping_table_in_A_9, NFVSOL(19)000577r2_SOL001Ed271_-_Standards_Configurable_Properties, NFVSOL(19)000593_SOL001ed271_adding_introduction_in_clasue_9_1, NFVSOL(19)000595_SOL001ed271_resolving_requirement_occurrenace_issue, NFVSOL(19)000597r3_SOL001ed271_deployment_flavour_related_CSAR_design_for_NSD, NFVSOL(19)000598r2_SOL001ed271_TOSCA_Imperative_workflows_NSD_Editor_s_Notes, NFVSOL(19)000599r2_SOL001ed271_TOSCA_Imperative_workflows_Example_Editors_Notes, NFVSOL(19)000607r2_SOL001ed271_VNFFG_clause_6_8_2_6_Editor_s_note_handling, NFVSOL(19)000608r2_SOL001ed271_VNFFG_clause_7_8_2_6_Editor_s_note_handling, NFVSOL(19)000609r2_SOL001ed271_VNFFG_clause_7_8_5_1_Editor_s_notes_handling, NFVSOL(19)000610_SOL001ed271_VNFFG_clause_7_8_6_1_Editor_s_note_handling, NFVSOL(19)000611r2_SOL001ed271_adding_TOSCA-Simple-Profile-yaml-v1_3_reference, NFVSOL(19)000616_SOL001ed271_ConnectivityType, NFVSOL(19)000617r3_SOL001ed271_nfviConstraint, NFVSOL(19)000633_SOL001ed271_VnfIndicators_editor_s_notes_resolution, NFVSOL(19)000636r2_SOL001ed271_NS_workflow_defintion_update, NFVSOL(19)000702_SOL001ed271_adding_reference_for_SwImageData, NFVSOL(19)000703r1_SOL001ed271_forwarding_behaviour_input_parameters, NFVSOL(19)000704r1_SOL001ed271_correction_on_NfpPositionElement_definition, NFVSOL(19)000705r1_SOL001ed271_correction_on_descriptor_id, NFVSOL(19)000741r1_SOL001ed271_NfpPositionElement_node_type_improvement, NFVSOL(19)000596r3_SOL001ed271_deployment_flavour_related_CSAR_design_for_VNF D, NFVSOL(19)000725r2_SOL001ed271_vnfd_common_yaml_file_for_v1_3, NFVSOL(19)000744r2_SOL001ed271_Clause_7_1_Introduction_improvement, NFVSOL(19)000566r9_SOL001ed271_adding_boot_data_type, NFVSOL(19)000674r1_SOL001Ed271_Monitoring_Parameters_-_Alignment_with_IFA027
2019.11	2.6.7	Implemented NFVSOL(19)000780r3_SOL001ed271_VnfIndicator_id_Issue_1, NFVSOL(19)000784r1_SOL001ed271_solving_remaining_editor_s_notes, NFVSOL(19)000785r1_SOL001ed271_solving_remaining_monitoring_issues, NFVSOL(19)000789_SOL001ed271_adding_boot_data_in_NBWC_list_table, NFVSOL(19)000719r12_SOL001ed271_forwarding_capability_for_VNF_node_type
2020.02	3.0.1	Implemented NFVSOL(19)000710r3_SOL001ed331_deployment_flavour_related_CSAR_design_for_VNF D, NFVSOL(19)000792r1_SOL001ed331_FEAT10_Add_specification_for_Multi-Site_Connectivity_Services, NFVSOL(19)000799r2_SOL001ed331_FEAT05_Adding_priority_to_NSD
2020.02	3.0.2	Implemented: NFVSOL(19)000714r4_SOL001ed271_support_of_auto-scaling_with_use_of_VNF_indicator, NFVSOL(20)000018r2_SOL001ed331_deployment_flavour_related_CSAR_design_for_NSD, NFVSOL(20)000067r1_SOL001ed331_FEAT16_Adding_SAL_to_NSD
2020.04	3.0.3	Implemented: NFVSOL(20)000016r6_SOL001ed331_FEAT02_VnfPackageChange, NFVSOL(20)000040r6_SOL001ed331_Criteria_for_backward_compatibility_of_changes, NFVSOL(20)000117_SOL001ed331_Vnflcm_update_TOSCA_1_3_grammar, NFVSOL(20)000172r1_SOL001ed331_VNF_specific_datatypes_naming_rules, NFVSOL(20)000239r2_SOL001ed331_release_3_mirror_adding_NS_DF_design_principle_in_annex, NFVSOL(20)000242_SOL001ed331_updating_annex_C, NFVSOL(20)000285_SOL001ed331_VNF_specific_types_naming_rules rapporteur changes: editorial, changing "will" to "with" in some places in annex C.2 and C.3 editorial, removing extra space in some of the TOSCA type definitions

Date	Version	Information about changes
2020.05	3.0.4	Implemented: NFVSOL(20)000295r3_SOL001ed331_FEAT02_ChangeCurrentVnfPackage_interface, NFVSOL(20)000320r6_SOL001ed331_FEAT15_VNF_Snapshot, NFVSOL(20)000354r1_SOL001ed331_FEAT02_ChangeCurrentVnfPackage_AnnexA_D_ma pping, NFVSOL(20)000327r1_SOL001ed331_rel3_mirror_corrections_of_specific_node_tpye, NFVSOL(20)000353_SOL001ed331_Vnflcm_interface_EN_cleanup, NFVSOL(20)000376r2_SOL001ed331_mirror_of_375_PNFD_geographic_coordinates_supp or, NFVSOL(20)000378r2_SOL001ed331_rel3_mirror_add_VNF_related_type_names_in_claus e, NFVSOL(20)000380r1_SOL001ed331_rel3_mirror_add_NS_related_type_names_in_clause _7, NFVSOL(20)000382r1_SOL001ed331_rel3_mirror_clarification_of_using_VnfMonitoring, NFVSOL(20)000384_SOL001ed331_rel3_mirror_clarification_on_ip_address_type, NFVSOL(20)000386_SOL001ed331_rel3_mirror_clarification_on_VirtualLinkable_cap, NFVSOL(20)000194r3_SOL001ed331_release_3_mirror_example_corrections
2020.06	3.0.5	Implemented: NFVSOL(20)000267r3_SOL001ed331_VNF_node_type_definitions, NFVSOL(20)000355r2_SOL001ed331_examples_cleanup_tosca1_3_interfaces, NFVSOL(20)000356_SOL001ed331_example_cleanup_to_tosca_1_3, NFVSOL(20)000357_SOL001ed331_all_type_definitions_metadata_cleanup, NFVSOL(20)000394r2_SOL001ed331_clause6_11_cleanup_tosca1_3, NFVSOL(20)000395r4_SOL001ed331_sw_image_data_align_with_TOSCA_1_3, NFVSOL(20)000396r5_SOL001ed331_removing_reference_of_TOSCA_1_1_and_1_2, NFVSOL(20)000397r1_SOL001ed331_clause_7_11_align_tosca_1_3, NFVSOL(20)000399r1_SOL001ed331_clause_8_11_align_tosca_1_3, NFVSOL(20)000517r1_SOL001ed331_Rel-3_mirror-fixing_optional_properties, NFVSOL(20)000527r4_SOL001ed331_update_annex_F, NFVSOL(20)000597_SOL001ed331_resolve_remaining_issues_for_TOSCA_reference, NFVSOL(20)000598r1_SOL001ed331_resolve_editor_note_for_package_change. Undo the implementation of NFVSOL(19)000799r2, supporting for FEAT5 will be removed from this version
2020.09	3.3.2	Implemented: NFVSOL(20)000387r3_SOL001ed341_adding_virtualLinkProtocolData_for_NsVirtualLink, NFVSOL(20)000617r7_SOL001ed341_Use_of_Credentials_datatype, NFVSOL(20)000618_SOL001ed341_VnfPackageChange_corrections, NFVSOL(20)000674_SOL001ed341_modifiable_attributes_example_correction, NFVSOL(20)000676r1_SOL001ed341_update_extension_rule
2020.11	3.3.3	Implemented: NFVSOL(20)000723r4_SOL001Ed341_Fixing_examples, NFVSOL(20)000359r5_SOL001ed331_support_of_trunk_port_topology, NFVSOL(20)000679r2_SOL001ed341_monitoring_parameter_identifier, NFVSOL(20)000716r2_SOL001Ed341_Clarification_on_node_type_definitions_in_an_NSD, NFVSOL(20)000737r1_SOL001Ed341_Ambiguous_use_of_may_not_, NFVSOL(20)000744r2_SOL001ed341_uniform_delta_correction, NFVSOL(20)000747r1_SOL001ed351_fix_TOSCA_YAML_version_reference, NFVSOL(20)000749r1_SOL001ed341_Adding_NS_scaling_aspects_instantiation_levels, NFVSOL(20)000752r6_SOL001ed351_support_using_VnfConfigurableProperties_for_boot_ data, NFVSOL(20)000767_SOL001ed351_Removal_of_constraint_in_VnfIndicator_attribute
2020.12	3.3.4	Implemented: NFVSOL(20)000774r1_SOL001ed351_add_Dependencies_in_VNFD, NFVSOL(20)000775_SOL001ed351_adding_example_in_NSD_for_VL_protocol_data, NFVSOL(20)000776_SOL001ed351_fix_issues_for_VnfcConfigurableProperties, NFVSOL(20)000777_SOL001_Correct_typos_in_NsAffinityRules_etc_, NFVSOL(20)000798_SOL001ed351_Corrections_in_policies_definitions
2021.02	3.3.5	Implemented: NFVSOL(21)0000001_SOL001ed351_-_Annex_A_-_Fixing_errors_in_YAML_examples_A1- A6, NFVSOL(21)000003r1_SOL001ed351_-_Annex_A_-_Fixing_errors_in_YAML_examples_A7- A17, NFVSOL(21)000004r1_SOL001ed351_-_Use_of_TOSCA_functions_-_specification, NFVSOL(21)000007_SOL001ed351_Scale_Inputs, NFVSOL(21)000008_SOL001ed351_Wrong_indentation_of_entry_schema, NFVSOL(21)000030_SOL001ed351_vducp_occurrences_correction, NFVSOL(21)000089_SOL001ed351_virtual_binding_capability
2021.02	4.0.1	Created based on SOL001 V3.3.5

Date	Version	Information about changes
2021.03	4.0.2	Implemented: NFVSOL(21)000011r8_SOL001ed421_Extensions_to_VNFD_Data_Model_in_Support_of_C ontainer, NFVSOL(21)000018r6_SOL001ed421_OsContainer_and_OsContainerGroup_node_definitio n, NFVSOL(21)000020r2_SOL001ed421_Changes_to_tosca_relationships_nfv_AttachesTo_de scription in support of OsContainers, NFVSOL(21)000064r5_SOL001ed421_Mciop_Profile_modelling, NFVSOL(21)000118_SOL001ed421_Mirror_SOL118_Adding_VnfLcmCoordination_to_interf aceName, NFVSOL(21)000019r4_SOL001ed421_Enhancements_to_VduCp_in_support_of_containers
2021.04	4.0.3	Implemented: NFVSOL(21)000021r4_SOL001ed421_New_Nodes__Capabilities__Relationships_and_Data types_to_model_VirtualCp, NFVSOL(21)000171_SOL001ed421_release_4_mirror_adding_VnfIndicator_support_in_NS D, NFVSOL(21)000172_SOL001ed421_rel_4_mirror_add_new_NsAutoScale_policy, NFVSOL(21)000173_SOL001ed421_rel_4_mirror_new_NsVnfindicator_interface, NFVSOL(21)000174_SOL001ed421_rel_4_mirror_NS_node_attribute_for_VnfIndicator, NFVSOL(21)000193_SOL001ed421_Add_per_vnfc_instance_property_to_Storage_Nodes_, NFVSOL(21)000213_SOL001ed421_rel4_mirror_add_new_input_in_Nslcm_operation_to_su pport_autosclae, NFVSOL(21)000219_SOL001ed421_Rel-4_mirror_of_131__VipCp_node_property, NFVSOL(21)000220_SOL001ed421_Rel- 4_mirror_of_159_FEAT05_Adding_priority_to_NSD, NFVSOL(21)000221_SOL001Ed421_- _Corrections_to_Annex_A_examples_mirror_of_120, NFVSOL(21)000223_SOL001ed421_Change_name_of_container_deployable_unit_requirem ent,
2021.04	4.0.4	Implemented: NFVSOL(21)000211r3_SOL001ed421_VNFD_TOSCA_model_update, NFVSOL(21)000214_SOL001ed421_remove_sw_image_data, NFVSOL(21)000227_SOL001d421_-VIP_CP_requirements_Mirror_of_122r1_, NFVSOL(21)000228_SOL001ed421_-Support_of_additional_TOSCA_Functions_- _Mirror_of_121r4, NFVSOL(21)000229_SOL001ed421_Rel-4_mirror_of_212r1_correct_valid_values_forma, NFVSOL(21)000238r2_SOL001ed421_Additional_VduCp_vnic_types_in_support_of_contain ers, NFVSOL(21)000246r1_SOL001ed421_Correct_some_minor_errors_in_the_definitions_of_M ciop_and_OsContainerDeployableUnit, NFVSOL(21)000254_SOL001ed421_rel4_mirror_correct_NSD_VNFFG_example, NFVSOL(21)000256_SOL001ed421_-Support_of_additional_TOSCA_Intrinsic_Functions, NFVSOL(21)000259_SOL001ed421_rel4_mirror_update_table_in_annex_A_9, NFVSOL(21)000260r1_SOL001ed421_rel4_mirror_multiple_errors_correction, NFVSOL(21)000262_SOL001ed421_Rel-4_mirror_of_208_Resolution_of_editors_notes, NFVSOL(21)000263_SOL001ed421_Rel-4_mirror_of_234_Resolution_of_editor_s_note, NFVSOL(21)000266_SOL001ed421_rel4_mirror_clarification_on_virtualLinkProtocol, NFVSOL(21)000267_SOL001ed421_mirror_of_060r7_VnfLcmOperationCoordination, NFVSOL(21)000268_SOL001ed421_mirror_of_061_VnfLcmOperationCoordination_extens, NFVSOL(21)000269_SOL001ed421_mirror_of_062_VnfLcmOperationCoordination_extens, NFVSOL(21)000298_SOL001ed421_- _Inconsistencies_in_VNF_and_NS_Node_templates_-, NFVSOL(21)000300_SOL001ed421_-properties_and_requirements_in_top-level_template, NFVSOL(21)000302_SOL001ed421_-Use_of_the_substitution_directive_in_top-level
2021.06	4.0.5	Implemented: NFVSOL(21)000152r5_SOL001ed421_Helm_chart_artifact_type, NFVSOL(21)000311_SOL001Ed421_Small_fixes_in_Annex_E, NFVSOL(21)000312r2_SOL001Ed421_Small_fixes_in_Annex_A, NFVSOL(21)000313r1_SOL001Ed421_Small_fixes_in_Yaml_definitions, NFVSOL(21)000336_SOL001Ed351_Clarification_on_the_contents_of_service_template
2021.07	4.0.6	Implemented: NFVSOL(21)000364r3_SOL001Ed421_Fixing_issues_in_Annex_A, NFVSOL(21)000370_SOL001Ed421_Fixing_issues_in_Annex_E, NFVSOL(21)000373r1_SOL001ed421_update_SwImage_artifact_definition, NFVSOL(21)000374_SOL001ed421_adding_affinityOrAntiAffinity_scope_for_CIS_node, NFVSOL(21)000377_SOL001ed421_New_scope_value_for_Ns_affinity_policies, NFVSOL(21)000395_SOL001ed421_Per_instance_storage, NFVSOL(21)000409_SOL001ed421_Mirror_408_VnfPackageChange_add_targets.

Date	Version	Information about changes
2021.09	4.0.7	Implemented: NFVSOL(21)000252r3_SOL001ed421_Container_requests_of_extended_resources, NFVSOL(21)000369r3_SOL001Ed421_Fixing_issues_in_Annex_A_14_VNFFG__, NFVSOL(21)000375r3_SOL001ed421_input_example_for_VnfcConfigurableProperties, NFVSOL(21)000407_SOL001ed421_Mirror_of_406_Bugfix_with_adding_fixedIpAddress_, NFVSOL(21)000435r1_SOL001ed421_swImage_for_virtualBlockStorage, NFVSOL(21)000436r2_SOL001ed421_updating_placementGroup_for_container, NFVSOL(21)000450_SOL001ed421_Missing_note_in_SwImage_artifact, NFVSOL(21)000451r1_SOL001ed421_mcio_constraint_parameters, NFVSOL(21)000437r2_SOL001ed421_VnfExtCp_updating, NFVSOL(21)000480_SOL001ed421_mciop_artifact_clarification, NFVSOL(21)000481r1_SOL001ed421_SecurityGroupRule_for_container, NFVSOL(21)000484_SOL001ed421_instantiation_levels_and_scaling_aspects, NFVSOL(21)000487_SOL001ed421_resolution_of_Helm_charts_related_editor_s_notes, NFVSOL(21)000491r1_SOL001ed421_resolution_of_CRD_related_editor_s_notes, NFVSOL(21)000495r2_SOL001Ed421_Minor_YAML_improvements
2021.10	4.0.8	Implemented: NFVSOL(21)000097r9_Example_of_VNFD_in_support_of_OsContainers, NFVSOL(21)000440r2_SOL001ed421_editor_notes_handling_for_OsContainer, NFVSOL(21)000496r2_SOL001ed421_add_support_of_requirement_for_hugepages__, NFVSOL(21)000506_SOL001ed421_Removal_of_editor_s_note_on_bootdata, NFVSOL(21)000509_SOL001Ed421_update_figure_6_1_2, NFVSOL(21)000510_SOL001Ed421_mapping_of_configurable_properties_in_Vdu_Compute
2021.10	4.0.9	Implemented: NFVSOL(21)000439_OL001ed421_release_4_mirror_clarify_support_of_autoScale, NFVSOL(21)000448r4_SOL001ed421_Support_of_multiple_selectors_in_VnfPackageChang , NFVSOL(21)000525_SOL001Ed421_Additional_YAML_fixes, NFVSOL(21)000528_SOL001Ed421_Harmonisation_of_policy_target_specifications, NFVSOL(21)000529r1_SOL001Ed421_Additional_Requirements_violation_in_Annex_A
2021.10	4.0.10	Implemented: NFVSOL(21)000542r1_SOL001ed421_Rel_4_Mirror_of_475_and_541_Bugfix_with_adding min_number_of_preserved_instances_to_NfviMaintenanceInfo
2021.11	4.0.11	Implemented: NFVSOL(21)000617r1_SOL001ed421_mciopProfile_note, NFVSOL(21)000618_SOL001ed421_Missing_references_to_note, NFVSOL(21)000619_SOL001ed421_Annex_A_18_corrections, NFVSOL(21)000621_SOL001ed421_note_reference_missing_for_affinityRule, NFVSOL(21)000622r1_SOL001ed421_monitoring_parameter_not_support_for_containers NFVSOL(21)000620r1_SOL001ed421_Sw_image_properties, NFVSOL(21)000635_SOL001ed421_wrong_reference
2022.02	4.2.2	Created by the published V4.2.1 document with the following implemented CRs: NFVSOL(21)000630_SOL001ed431_Rel_4_mirror_of_629r2_follow-up_of_DP_519r1, NFVSOL(21)000649_SOL001ed431_Rel_4_mirror_of_536r3_Adding_externallyManaged_to _VnfVirtualLinkDescriptor, NFVSOL(22)000008r1_SOL001ed431_enh02_03_adding_policy_for_dataFlowMirroring, NFVSOL(22)000010r3_SOL001ed431_Forwarding_node_extension, NFVSOL(22)000011r3_SOL001ed431_enh02_03_example_for_dataFlowMirroring, NFVSOL(22)000017r1_ENH02_04_SOL001ed431_Add_ExtInvariantId_to_VNF__NS_and_P NF, NFVSOL(22)000018_SOL001ed431_Removal_of_flavour_id_in_NsProfile, NFVSOL(22)000028r2_SOL001ed431_Rel_4_mirror_of_NFVSOL_21_000642r1_fixing_defini tion_of_ipAddressAssignment_attribute, NFVSOL(22)000030r1_SOL001ed431_Rel_4_mirror_of__NFVSOL_22_000029_fixing_defini tion_of_macAddressAssignment_attribute, NFVSOL(22)000032r1_SOL001ed431_Rel- 4_mirror_of_031_Add_flavour_id_to_node_template, NFVSOL(22)000045r1_SOL001ed431_fixing_schema_error_introduced_by_NFVSOL_21_00 0649

Date	Version	Information about changes
2022.03	4.2.3	Implemented: NFVSOL(22)000057_SOL001Ed431_Enhancements_and_small_fixes_to_Annex_A_18_, NFVSOL(22)000059_SOL001Ed431_Fixing_vnfm_info_constraints, NFVSOL(22)000072_SOL001Ed431_Missing_units_for_Vdu_OSContainer_properties, NFVSOL(22)000074r4_SOL001ed431_enh02_01_adding_L2_Network_scope_in_NsAffinityRule_NsAntiAffinityRule, NFVSOL(22)000075r1_SOL001ed431_update_mapping_table_in_A_19, NFVSOL(22)000076r2_SOL001ed431_clarify_node_type_for_profile_element_in_NfpPositionElement, NFVSOL(22)000085r1_SOL001Ed431_underspecified_file_system_in_VirtualFileStorage, NFVSOL(22)000086_SOL001ed431_fix_example_in_annex_A_, NFVSOL(22)000088_SOL001ed431_update_description_for_additionalServiceData_, NFVSOL(22)000096r1_SOL001ed431_mcio_identification_data, NFVSOL(22)000097_SOL001ed431_SwImage_properties
2022.04	4.2.4	Implemented: NFVSOL(22)000104r1_SOL001Ed431_VNF_Scaling_Policies_for_initial_deltas, NFVSOL(22)000105_SOL001Ed431_Adding_missing_constraints_to_properties_of_type, NFVSOL(22)000112r1_SOL001Ed431_A_18_-_Missing_mcio_identification_data, NFVSOL(22)000116_SOL001ed431_mirror_of_638_update_vnfm_interface_info_type, NFVSOL(22)000117_SOL001ed431_mirror_of_639_removal_of_deprecated_element_in_vnf_d, NFVSOL(22)000134_SOL001Ed431_Fixing_figures_in_clause_6_1, NFVSOL(22)000160r1_SOL001Ed431_Uncommenting_modifiable_attributes_and_similar_properties, NFVSOL(22)000162_SOL001ed431_Annex_D_mapping_table_updates, NFVSOL(22)000189_FEAT21_SOL001ed431_implementation NFVSOL(22)000164_SOL001Ed431_VNF_Scaling_Policies_for_initial_deltas
2022.06	4.2.5	Implemented: NFVSOL(22)000141_SOL001Ed431_Affinity_rules_between_instances_of_the_same_NestedNS, NFVSOL(22)000161r6_SOL001Ed431_Simplification_of_VNF_PNF_NS-specific_node_type_definitions, NFVSOL(22)000213r2_SOL001ed431_update_VNFD_figure_in_6_1, NFVSOL(22)000216_SOL001ed431_adding_vipCpDelta_policy, NFVSOL(22)000217_SOL001ed431_correct_VirtualFileStorageData_description, NFVSOL(22)000225_FEAT21_SOL001ed431_correction, NFVSOL(22)000228r1_SOL001Ed431_Huge_Pages_for_VM-based_VNFs, NFVSOL(22)000231_SOL001Ed431_CPU_Pinning_for_Containerized_VNFs, NFVSOL(22)000263_SOL001Ed431_Affinity_rules_between_instances_of_the_same_NestedNS, NFVSOL(22)000270r1_SOL001ed431_multiple_corrections, NFVSOL(22)000271_SOL001ed431_update_annexD_for_dataFlowMirror, NFVSOL(22)000272_SOL001ed431_correct_associatedVdu_requirement_name, NFVSOL(22)000215r3_SOL001ed431_VirtualCp_node_type_usage_clarification, NFVSOL(22)000289r2_ENH02_05_SOL001ed431_implementation
2022.10	4.3.3	Implemented: NFVSOL(22)000380_SOL001Ed441_Minor_editorial_changes_to_Annex_A_examples_mirror_of_348, NFVSOL(22)000381_SOL001Ed441_Fix_CPU_pinning_policy_description, NFVSOL(22)000382r1_SOL001ed441_MetalLB_address_pool_name_parameterization, NFVSOL(22)000386_SOL001Ed441_Missing_node_types_in_Clause_6_1_figures, NFVSOL(22)000403_SOL001Ed441_Graphical_Conventions_for_TOSCA_requirements_mirror_of_402 NFVSOL(22)000456_SOL001ed441_ENH02_05_NsScaleInfo_in_NsProfile
2022.12	4.3.4	Implemented: NFVSOL(22)000448r1_SOL001ed441_Add_BootDataVimSpecificProperties_EN, NFVSOL(22)000454r1_SOL001Ed441_Simplification_of_Annex_A_examples, NFVSOL(22)000470_SOL001ed441_annex_B_update_for_yaml_files_mirror_of_327, NFVSOL(22)000483r1_SOL001ed441_Support_for_dual_stack_, NFVSOL(22)000449_SOL001ed441_multiple_corrections, NFVSOL(22)000452_SOL001ed441_mirror_of_451_corrections_for_SwImageDescId
2022.12	4.3.5	Implemented: NFVSOL(22)000505_SOL001ed441_Handling_of_obsolescence_of_IETF_RFC_7230, NFVSOL(22)000517r2_SOL001ed441_update_serviceData_to_support_ingress, NFVSOL(22)000518r1_SOL001ed441_bug_fix_in_A_18

Date	Version	Information about changes
2023.01	4.3.6	Implemented: NFVSOL(23)000005r1_SOL001ed441_update_IETF_reference, NFVSOL(23)000010r1_SOL001ed441_Address_pool_name_clarification_, NFVSOL(23)000017_Feat_17_SOL001ed441_support_of_floating_IP_address_for_containerized_VNFs
2023.06	4.4.3	Implemented: NFVSOL(23)000083r1_SOL001ed451_Guidelines_for_the_import_of_interface_type_definition, NFVSOL(23)000134_SOL001ed451_Align_occurrences_of_virtual_link_requirements_in_cp_nodes, NFVSOL(23)000166r1_SOL001ed451_Add_support_for_DaemonSets
2023.07	4.4.4	Implemented: NFVSOL(23)000176_FEAT31_SOL001ed451_Introduction_of_deployable_modules_in_the_VNFD, NFVSOL(23)000177_FEAT31_SOL001ed451_Deployable_modules_relation_to_scaling_aspect, NFVSOL(23)000179_FEAT31_SOL001ed451_Adding_deployable_modules_to_the_component_mapping, NFVSOL(23)000180_FEAT31_SOL001ed451_Deployable_modules_relation_to_MCIOPs, NFVSOL(23)000181_FEAT31_SOL001ed451_Adding_deployable_modules_in_VNF_and_NS_profiles, NFVSOL(23)000208_SOL001ed451_Bug008207_Add_the_missed_mapping_information_between_node_template_name_of_OsContainerDeployableUnit_and_vduld in Annex D.2
2023.08	4.4.5	Implemented: NFVSOL(23)000190r1_ENH02_06_SOL001ed451_Introduction_of_parameter_mapping_artifacts, NFVSOL(23)000240_SOL001ed451_correct_ABNF_Grammar_, NFVSOL(23)000256_FEAT31_SOL001ed451_Supported_operations_for_change_of_deployable_modules, NFVSOL(23)000257_FEAT31_SOL001ed451_SelectVnfDeployableModulesOperationConfiguration, NFVSOL(23)000258_SOL001ed451_VnfInstantiateOperationConfiguration
2023.09	4.4.6	Implemented: NFVSOL(23)000287r1_FEAT31_SOL001ed451_Error_corrections NFVSOL(23)000288_FEAT31_SOL001ed451_Example_of_a_VNF_with_deployable_modules, NFVSOL(23)000291r1_FEAT31_SOL001ed451_SAP_to_deployable_modules_mapping_information_in_the_NSD, NFVSOL(23)000299r1_FEAT31_SOL001ed451_Examples_of_SAP_to_deployable_modules_mapping, NFVSOL(23)000310_FEAT31_SOL001ed451_Add_deployableModule_to_mapping_tables, NFVSOL(23)000311_FEAT31_SOL001ed451_Error_corrections_2
2023.10	4.4.7	Implemented: NFVSOL(23)000309r5_SOL001ed451_FEAT21_VNFD_PaaS_Service_modeling_refinements, NFVSOL(23)000341_SOL001ed451_FEAT21_PaaS_Annex_D_mapping, NFVSOL(23)000217r5_Enh01_01_SOL001ed451_Add_CertificateDesc, NFVSOL(23)000289_Enh01_01_SOL001ed451_update_mapping_table_and_add_CertSubjectData, NFVSOL(23)000290r5_Enh01_01_SOL001ed451_resolve_EN_related_to_Certificate, NFVSOL(23)000308r3_SOL001ed451_FEAT21_NSD_modelling_to_use_PaaS_Services
2023.11	4.4.8	Implemented: NFVSOL(23)000366r1_SOL001ed451_Review_comments_for_ENH01_01
2024.01	5.0.2	Implemented: NFVSOL(23)000376r2_FEAT31_SOL001ed511_Adding_sibling_properties_for_VDU_capacity_related_properties, NFVSOL(23)000378_FEAT31_SOL001ed511_new_property_in_VduProfile, NFVSOL(23)000397_SOL001ed511_FEAT21_NSD_modelling_to_use_PaaS_Services, NFVSOL(23)000400r1_FEAT31_SOL001ed511_new_property_in_VnfProfile, NFVSOL(23)000403_FEAT31_SOL001ed511_new_value_of_supported_operations, NFVSOL(24)000001r1_FEAT31_SOL001ed511_Scale_VNF_operation_stimuli, NFVSOL(24)000002r1_FEAT31_SOL001ed511_Text_about_new_property_in_VnfProfile

Date	Version	Information about changes
2024.04	5.0.3	Implemented: NFVSOL(23)000313r2_SOL001ed511_FEAT21_NSD_Enhancing_VL_with_additional_capability, NFVSOL(24)000046_SOL001ed511_ENH01_01_corrections, NFVSOL(24)000056_FEAT35_SOL001ed511_clarification_on_osContainer_and_Mciop_usage, NFVSOL(24)000081_FEAT35_SOL001ed511_an_example_of_VNFD_with_simple_design, NFVSOL(24)000084_SOL001ed511_Monitoring_parameters_Alignment_with_IFA027ed511, NFVSOL(24)000085r1_FEAT30_SOL001ed511_Enhancements_for_configuration, NFVSOL(24)000086_FEAT21_ENH01_01_SOL001ed511_Update_mapping_tables_in_annex_A_9, NFVSOL(24)000090r1_FEAT35_SOL001ed511_upgrade_type
2024.04	5.0.3	Implemented: NFVSOL(24)000148r2_SOL001ed511_FEAT35_External_access_for_PM
2024.07	5.1.1	Published
2024.10	5.1.3	Implemented: NFVSOL(23)000255r3_SOL001ed451_FEAT21_NFV_support_for_5G_Additional_network_properties_for_CPs, NFVSOL(24)000211_SOL001ed521_correction_on_VNFD_example_description_in_A18, NFVSOL(24)000213_FEAT29_SOL001ed521_Monitoring_parameters_Alignment_with_IFA027ed521, NFVSOL(24)000218_SOL001ed521_correction_on_defined_of_max_number_of_instance, NFVSOL(24)000229_FEAT19b_SOL001ed521_VduSubCp_enhancements, NFVSOL(24)000288_FEAT30_SOL001ed521_Indication_of_supported_configuration_protocols, NFVSOL(24)000304_FEAT30_SOL001ed521_an_example_of_VNFD_with_VNF-specific_metadata NFVSOL(24)000243r1_FEAT29_SOL001ed521_NS_Monitoring_parameters_Alignment_with_IFA 027ed521 clause 7_3
2024.10	5.1.4	Implemented: NFVSOL(24)000336_SOL001ed521_update_on_annex_B
2024.12	5.2.1	Published
2025.06	5.2.2	Implemented: NFVSOL(25)000121_FEAT29_SOL001ed531_Addition_of_CpuPowerState_data_type, NFVSOL(25)000222r2_FEAT29_SOL001ed531_Addition_of_PowerProfile_data_type

History

Version	Date	Status
V5.1.1	July 2024	Publication
V5.2.1	December 2024	Publication
V5.3.1	August 2025	Publication