ETSI GR NFV-TST 006 V1.1.1 (2020-01)

Network Functions Virtualisation (NFV);
Testing;
Report on CICD and Devops

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
DGR/NFV-TST006

Keywords
CI/CD, DevOps, NFV, NFVI, SDN

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 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is 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 Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, 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.

Foreword

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

Modal verbs terminology

In the present document "**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 provides guidance and recommendations on how to leverage DevOps and CI/CD techniques across the boundary from SW provider to service provider, or any combination of developer, installation and operational entities. It explores the implications of the processes with regard to the impact of the SW package handoff between SW provider and service provider, the required functionality in the NFV system, the different deployment and operational options by:

- Exploring use cases.
- Defining the steps in the process.
- Defining the metrics and measurements.
- Defining the test procedures.
- Defining post handoffs tests.
- Providing recommendations on VNF Package.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

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-REL 006: "Network Functions Virtualisation (NFV); Reliability; Maintaining Service Availability and Continuity Upon Software Modification".

[i.2] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for main concepts in NFV".

[i.3] ETSI GS NFV-IFA 011 (V3.2.1) (2019-04): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; VNF Descriptor and Packaging Specification".

[i.4] ETSI GS NFV-TST 001 (V1.1.1) (2016-04): "Network Functions Virtualisation (NFV); Pre-deployment Testing; Report on Validation of NFV Environments and Services".

[i.5] ETSI GR NFV-TST 004 (V1.1.2) (2017-07): "Network Functions Virtualisation (NFV); Testing; Guidelines for Test Plan on Path Implementation through NFVI".

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

VNF operator: person or company that operates the VNF

VNF validator: person or company that validates the functionality of a VNF

3.2 Symbols

Void.

3.3 Abbreviations

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

CI	Continuous Integration
DA	Disciplined Agile
ITSM	IT Service Management
JAD	Joint Agile Delivery
SAFe	Scaled Agile Framework
SDLC	Software Development Life Cycle
SRE	Site Reliability Engineering

4 Background on Devops and CI/CD

4.1 Introduction

The basic intention in DevOps/Joint Agile Delivery is to incorporate operational and other related aspects of the entire software lifecycle as early in the development process as practical, and to ensure smooth delivery jointly across organizations. This requires that many steps in the process are more continuous, incremental, and iterative than in traditional waterfall models, and higher levels of automation are applied to the delivery pipeline, including testing.

To that end, there are a number of processes and concepts involved; these are described in the following clauses of the present document.

The context of the present document is the testing elements of DevOps/CI/CD; the present document does not seek to provide a comprehensive overview of these topics beyond that scope.

4.2 General Backgrounder

DevOps is a combination of practices that embody a culture of end-to-end ownership and shared responsibility across teams from Development, QA, Security, Operations, and other areas as a single cohesive, service focused team. DevOps takes much of its heritage from the areas of lean manufacturing and operational management theory. This is about applying industrial design techniques to the software industry.

Joint Agile Delivery (JAD) is a series of multi-party interlocks to support agile software development and delivery in a complex carrier environment. It does this through the expected liaison at key points along the way, such as around requirements, testing, and acceptance. It leverages CI/CD tools and techniques to ensure robust, repeatable, rapid delivery of each code iteration. It is compatible with agile software delivery methodologies such as the Scaled Agile Framework (SAFe), and Disciplined Agile (DA).

As hardware becomes commodity, a new technology base of cloud and virtualisation, microservices, and APIs has emerged. These technologies have provided an inflection point - an opportunity to re-examine the way in which ICT is performed and managed; to distil key processes down to their essence, and re-apply those principles in a way that leverages these technologies to provide a better way to manage quality and change. This new approach has dramatically altered the risk landscape, and this has an impact on the operating model, if an enterprise is to recognize the benefits of this new approach.

Some of the most important aspects of this change include:

- **Cloud versus bespoke infrastructure solutions**. Traditional applications required bespoke infrastructure solutions. Often the production environment was, due to cost, significantly different from other environments. This had implications in how it was maintained: managed in-situ across an extended lifespan; all-or-nothing changes executed in ‘quiet' times, often involving service disruption; configuration divergence between environments from development to test to production having implications on testing and quality assurance. Cloud, by contrast, has the potential to eliminate all of these concerns.

- **Infrastructure as code**. The nature of cloud environments is that infrastructure can be created, modified, and deleted using APIs. This means that it can be created under programmatic control. This implies: repeatability and consistency, eliminating human error, and elimination of the entropy associated with having to maintain bespoke, manually configured infrastructure in-situ across an extended life span.

- **All environments virtualised**. The virtualisation of all environments provides an opportunity to ensure consistency across environments never previously available (or cost prohibitive). This significantly reduces the risk associated with change, both due to consistency of the environments, and consistency of the change process used to promote code between environments.

- **Microservices; elimination of heavy integration overheads**. A significant source of enterprise risk is the integration of a large number of changes rolled up into major releases. This leads to extended integration timeframes, and a lack of direct traceability to the source of errors found during this integration cycle. The industry trend towards discrete microservices directly addresses this risk, increasing both velocity and quality simultaneously. The limited scope of each service makes testing more predictable and simpler. The use of APIs as the entry point protects consumers from internal changes, further simplifying integration testing. Each service can evolve at its own speed, no longer enforcing large, complex integration cycles.

Against this technology backdrop, DevOps institutes a set of cohesive software delivery practices:

- **Continuous Integration**. The defining characteristic of DevOps projects is their creation and ubiquitous use of a delivery pipeline which controls the advancement of release artefacts from idea through to production. This is not a static pipeline, nor is there only one. The pipeline more closely resembles a factory production line with many conveyor belts and components being assembled until ultimately brought together to make a new car. Each of these components have their own pipeline and operate under their own rules and speed.

- **Small, discrete changes**. A significant source of work and error is the delayed integration of new features and code changes into the "baseline". **Continuous Integration** espouses a view that work is continuously checked into the baseline, to catch any integration related issues early, and ensure that the product is always stable and passes regression tests. This is then coupled with **Continuous Delivery** to ensure that a new, clean version of the product is always shippable. This combination means that there is a confidence in the build and release process that supports a significantly faster end-to-end release process, further creating a virtuous cycle of quality and velocity.

- **Automated testing**. Supporting continuous integration and the rapid promotion of changes through the delivery pipeline, automation of testing is held out as an essential aspect of DevOps. Not all testing can be automated, but automation should always be sought to the extent possible.

- **High degree of automation**. Beyond testing, automation is applied to all steps in the delivery pipeline, including generation of intermediate artefacts, packaging, environment creation, delivery, deployment and event response. This significantly increases both the power and authority of the development team.

- **Monitoring, metrics, and real world feedback**. DevOps is about rapid and continuous feedback. A key focus is on instrumentation and operational telemetry, and using real world feedback rather than relying on requirements and test environments.

- **Cohesive, cross-functional teams; shared responsibilities**. One of the most talked about elements of DevOps is the integration of previously siloed expertise into a single, cohesive team with shared incentives and responsibilities.

Beyond these base practices, a number of "industry best" practices are also often cited:

- **Version Control and the "Left Edge"**. Critical to successful DevOps projects is the use of version control over all **source artefacts** (not just source code). This applies to less obvious source artefacts, such as visual workflows and related design artefacts. Developers only have write access to the "left edge" - the source artefacts. Everything generated from those artefacts is designated an **intermediate artefact**, that cannot be directly altered, and is re-generated by committing new source changes. This ensures that version control is enforced, and that every version of the system, and every element of the system is reproducible.

- **Continuous Integration Triggers and Automated Testing**. Another aspect of a mature CI delivery pipeline is the implementation of triggers - that the pipeline CI controller can detect events and automatically take action, such as continue to promote the object along the delivery pipeline, without human intervention. The most obvious example of this is where a code check-in causes the automated test suite to be run, in order to determine if the build is still clean after the most recent code change. CI triggers include source code repository triggers (code commit), test triggers (success/failure), and human approval triggers (approval to promote).

- **Use of "ephemeral" infrastructure**. A major historical source of error was the entropy introduced by the in-situ maintenance of infrastructure over an extended period. A cloud best practice is to generate new infrastructure for each new release, build and release the new version of the service onto that infrastructure, then to cutover/migrate the workloads to this new version in a controlled manner, and eventually destroy the virtual infrastructure of the previous version. This means that the current and previous version of the service remain operating in parallel until the migration is complete, and this provides additional mechanisms for managing risk.

- **No rollback, only fail forward**. The in-situ nature of traditional large systems meant that change was "all or nothing", and if a change failed, it would be "rolled back" and the system returned to its previous (pre-change) state. This technique was always fraught, as any transactions applied from the time of the change would also need to be rewound and then reapplied, something that was not always possible. It would also often mean restoring from backups - a procedure often untested except in high risk circumstances such as restoration during a major event; complications would abound. In mature DevOps environments, a bug is addressed at the source (left edge) and the tests that allowed the bug to slip through are also addressed. The velocity of the end-to-end process making it faster and more reliable to push a new version quickly. Where necessary, functionality might be immediately disabled without any new code change or rollback (see next).

- **Decoupling of Deployment from Activation**. Advanced DevOps practices include the use of "feature flags" to decouple deployment from activation. This addresses the most significant risk associated with change - the all-or-nothing nature of a release. This technique has service code consult a dynamic configuration service to determine whether a feature should remain dormant, be activated for a particular set of users (e.g. specific users, specific locations, specific volumes, etc.), or be fully activated for all, and this becomes the basic risk control mechanism to control the potential impact of a new feature or release.

- **Continuous Deployment**. With deployment and activation decoupled, delivery pipelines can extend all the way into production. Continuous Deployment, sometimes called "push on green" is instituted where sufficient confidence exists in the delivery pipeline, including the automated testing steps.

- **Continuous feedback, A/B testing**. All of the above features provide an environment of continuous feedback, where real production data is used to guide development and operations practices and priorities. Feature flags can be used to present alternate versions of a feature to different users, providing a way to compare their usefulness and value (A/B testing). This moves software development from the guessing associated with traditional requirements to the real world feedback of how the system is used.

- **Site Reliability Engineering (SRE)**. A sister movement to DevOps, SRE is the application of IT Service Management (ITSM) at cloud scale. Pioneered by Google™, Amazon™, Netflix™ and other Silicon Valley™ cloud companies, SRE is the application of automation to ICT operations, and the evolution of the IT operations function.

4.3 DevOps in an NFV Context

In the context of Network Function Virtualisation, all of the above benefits can be brought into play. Smaller, incremental releases of VNFs and VNFCs can be rapidly promoted through a continuous integration/delivery pipeline as described in the present document. The use of discrete functions and APIs help to promote rapid evolution and delivery of services in a complex, multi-vendor environment.

To recognize the full benefits of DevOps, significant process retooling may be required. The techniques described in the present document can help form a basis for planning such activities.

4.4 DevOps in a Multi-Party Setting (Joint Agile Delivery)

A further complication for NFV operators is the coordination and alignment of efforts across organizational boundaries. The operator may make use of functions from one or more suppliers. Each of these organizations may have their own DevOps Delivery Pipelines, and effort is needed to coordinate and synchronize across the lifecycle to ensure the value from DevOps practices is realized. "Joint Agile Delivery", as described in the present document, provides a framework for this coordination between parties, ensuring the benefits of DevOps in the complexities of an operator environment.

4.5 Delivery Pipeline

The "delivery pipeline" describes the end-to-end process, from idea to production. This pipeline is modelled on the traditional Software Development Life Cycle (SDLC) as a baseline, including SDLC activities such as requirements capture, design, coding, test, and release, but applied at a more fine-grained resolution, following an iterative software development methodology such as agile software development. Further, DevOps views this pipeline as a production line of activities that should be automated to the extent practical.

As software components, VNFs and VNFCs are developed, unit tested, and run in one or more test, staging or pre-production environments before they are deployed into a production environment. This approach allows development, operations, security, and other teams - in both the supplier and operator organizations - to build confidence in the release through testing, and to support a joint agile VNF development and delivery process all the way through to production deployment and operation.

The pipeline steps can be all within a single organization, or split across organizations. There may be different additional steps involved when a cross-organizational model is applied. For example, the testing environment might be split into test of the total VNF, a single VNFC, or a network service. Some operators might share environments with suppliers to assist with integration testing, others may force a hand-off between environments for this purpose.

Naturally, this introduces its own set of challenges, since the process should be reliable. For telecommunication environments, several check-points are likely to exist where a software component may not progress further down the line.

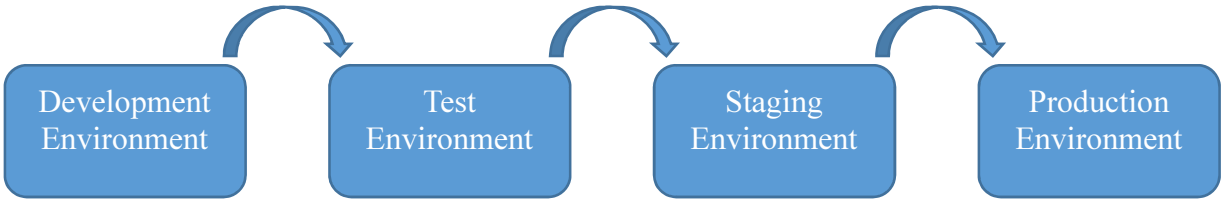

Figure 4.1: Example Delivery Pipeline (Baseline version)

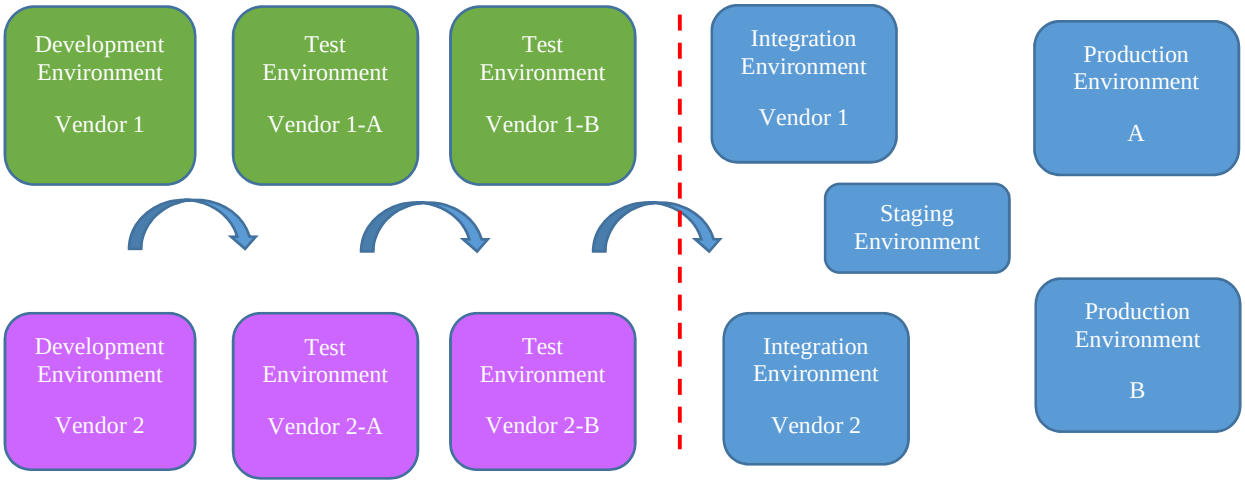

Figure 4.2: Example Cross-Organizational (Joint Agile) Delivery Pipeline/Alternate Production Environments

4.6 Continuous Integration

Continuous Integration refers to a core DevOps practice of automated promotion of code artefacts along the delivery pipeline and the automated commencement of the next activity where applicable. A typical delivery pipeline commences with creation or editing of a source artefact and the check-in of that artefact into a code repository. The Continuous Integration (CI) pipeline manager would trigger the next activity based on this check-in, such as execution of relevant unit tests. Should those tests exit cleanly, the next activity would be triggered. Activities along the pipeline might include creation of intermediate artefacts (such as binary objects) and further tests (e.g. integration tests, load tests, etc.). Each step along the pipeline takes the unit of delivery closer to production.

Continuous Integration is of critical importance in an NFV setting, where the units of functional delivery are smaller and more frequent.

In an NFV setting, continuous integration includes software integration of different components into a bigger entity on the fly or in very short cycles. Since there are potentially many dependencies, more frequent integration enables identification and resolution of problems at the interfaces between the components earlier in the process.

Using continuous integration, development of VNFCs will regularly be integrated with other VNFCs for a whole VNF. The same applies for several VNFs integrated together for a network service.

4.7 Joint Solution Verification Environment

Joint Solution Verification Environment refers to a shared verification environment (between suppliers and the operator), as a way to make the integration environment as similar as possible to the operator's target production environment to aid testing and integration across organizations. This Joint Agile Delivery approach is necessary in order for a supplier to deliver high quality components to an operator's complex existing environment. This will further reduce the time spent on solving environment related problems and make continuous integration activities run more smoothly.

4.8 Continuous and Automated Testing

The goal for testing in DevOps is that development and quality assurance should be able to test in production-like environments for early detection of problems. For various levels of testing, this can be done in a highly automated manner, if the test cases are defined as part of the software development activities. The bigger the system under test and the nearer to production, the more complex the tests are going to be; this is where automation of testing is going to be of most benefit.

4.9 Joint Test Design & Execution

Along the delivery pipeline, each organization may have different requirements regarding testing. Joint test design and execution enables organizations that are involved in testing and staging to start working on designing test specifications and requirements earlier, to have a common understanding on what will be tested and where the tests will be executed.For example, two organizations share the testing plan and test cases so that all the tests can be executed earlier, often in the development phase. By having comprehensive transparent joint test design and execution, duplication of effort on the testing in different steps along the pipeline is eliminated, resulting in lower TTM and lower operation cost.

The multi-organizational nature of most operator NFV environments puts additional emphasis on the need for clear agreement on requirements, testing, and acceptance. The use of APIs can help focus functional testing around API conformance; other operational requirement testing (e.g. performance, interoperability) will need the shared focus of all involved parties.

In addition to sharing test design and execution, the co-embedding of subject matter experts from the supplier and operator into a single joint delivery team ensures the highest levels of collaboration and communication, leading to significantly reduced friction associated with development, testing and release.

4.10 Continuous and Automated Delivery

As already mentioned above DevOps is about delivering pieces of software from development to production along a delivery pipeline with the different steps involved to get to a high quality production outcome. Continuous delivery means that the software components are delivered down the pipeline frequently, all the way to the "production boundary". Normal production change management controls continue to be applied from that point (but can be tuned based on demonstrated risk profiles). Therefore, more automation is required at all steps to automate that process as much as possible, including the generation of intermediate artefacts, and the packaging of release artefacts. Also the necessary tooling including environment management, release management, and provisioning tools are required.

4.11 Continuous Monitoring

Monitoring was always a strength of telecommunication systems. The change with DevOps is that monitoring is now required along the whole path from development to production and that other metrics might start playing a role.

Where previous environments often relied on manual (human) response to alert events, the scale and complexity of NFV infrastructure requires deeper monitoring, and autonomous response to most alarm conditions. This requires significant change in how monitoring, alarming, and event response are designed and performed.

4.12 Processes and Tooling

4.12.1 Process

In figure 4.3 there is an example process (single organization, single component), which many DevOps systems follow to a certain degree and some examples on the tooling used in such an environment.

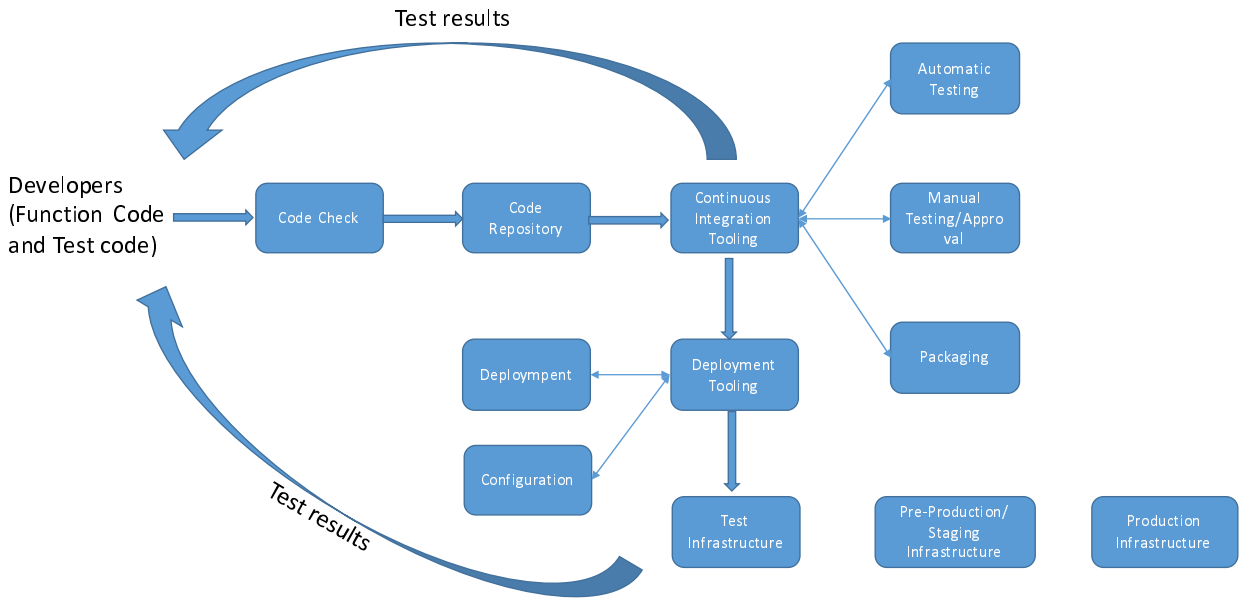

Figure 4.3: Example process in DevOps

Starting from development, the developer is writing some code for a function and with it also write test code for this function. Best current practice in DevOps is even that test code is written first to understand what the function should do, but that is a bit a matter of taste.

If the code is finished, there first syntax and code quality checks, before it will be added to the code repository on a development branch.

For new code in the repository the continuous integration tools take the code automatically out of the repository and run a whole set of automatic tests. Those test depend really on the function and many times on the programming language used, because there are at least some automatic language specific test frameworks available.

Depending on the scenario, there are also some manual tests required (seldom, because that process should run automatic) or a super user (human) approval of the code is required. The latter happens many time in open source projects such that there is some governance and control what code makes it to trunk (the main branch) of the code.

Next is packaging the code into a shippable form. Again that depends largely on the system used. The package format might be different, but also the granularity of a package might be different. For example, container-based micro-services based packaging is a relatively small unit of functionality packaged. For some NFV use cases the unit of the package might be a full VNF containing several VM images. It is important to know that the package should contain as much as possible all the code required to run, basically it should be as self-contained as possible. Still the package requires to have if not contained already a description of all the dependencies of that package.

In whatever form, the code is then deployed into a test infrastructure and it requires to get configured with the infrastructure specific configuration and the dependencies need to be resolved and bound to the dependent components available (late binding of dependencies).

On the test infrastructure, again a set of test are performed showing that the code is running on that particular infrastructure.

Finally, the further steps to pre-production/staging environment and eventually into production platform can be done the same way as with deploying it on the test infrastructure, and it can be done completely automatic, if enough test cases are defined, but many times there is much more human involvement and testing done before code makes it into the next steps.

In case of any test failing, the developer and configuration designers will be notified on what failed and the code change is rejected to be included into the main branch.

4.12.2 Tooling

In the following, the tools used are described in a generic way and abstract way.

Code repository: the main purpose of a code repository is the software code management including versioning, conflict detection/prevention, and as it says as a central repository for code in various maturity level of development and testing.

Code checking: the code checking is largely meant to gate keep the repository from syntactically incorrect code and it might be used to have certain best practices in code structuring and code writing checked.

Continuous Integration Tool: that tools basically do various things including the automatic testing of a component and the packaging of that component. For testing there are programming language specific tools available which do some basic and many times generic test of code. Additionally, there are function specific tests, typically written as test code to be run against the component. Then for packaging a set of tools are available and basically largely depend on the packaging format. For VNFs, the VNF descriptors are part of the packaging.

Deployment and Configuration: deployment tools can be as easy as copy/ship a package up to more sophisticated deployments tools with the proper integration into the environment. Most times there is a step of configuring some part of the software with environment specific configuration parameters including dependencies with services and other components.

5 Use cases

5.1 Introduction

This collection of use cases is created to collect and define requirements to the interface between the VNF Provider and VNF Operators. Where VNF Providers are organizations creating and selling VNF-s as products while VNF Operators are organizations using, operating and providing services based on these products. The internal processes of VNF Providers and VNF Operators are not discussed in these clauses until these processes have no effect to the interface between the VNF Providers and the VNF Operators.

The internal DevOps processes of the VNF Providers are generalized as "DevOps pipeline, VNF Provider part" while the DevOps processes of the VNF Operators are generalized as "DevOps pipeline, VNF Operator part". The process is described in detail in clause 4.12.1.

The use cases are defined in two levels. There are epics describing a high level use cases that are independent from the cardinality of the different organizations, number of VNF-s the phases of the process or from the place of the border in the process between the VNF Providers and the VNF Operators. Use cases are defined for the different specialized variants of the epics.

In the DevOps process there can be different number of organizations depending on the environment where the process is executed. Even with the VNF Provider and VNF Operator organizations the following variations are possible:

- The VNF Provider and the VNF Operator are the same organization.
- There is one VNF Provider and one VNF Operator.
- There are several VNF Providers and there is one VNF Operator.
- There is one VNF Provider and there are several VNF Operators.
- There are several VNF Providers and there are several VNF Operators.

VNF Providers and VNF Operators can run more than one DevOps process with different configurations according to their needs.

There are several optional phases in the DevOps process. It is always the subject of agreement between a VNF Provider and a VNF Operator which phases to include into their DevOps process.

Different VNF Providers and VNF Operator can execute different parts of the DevOps process and it is always a subject of agreement between the VNF Providers and VNF Operators where to define the boundary between the VNF Provider and VNF Operator in the DevOps process.

The "actors and processes" in figures 5.1 to 5.5 are to describe the context of the discussed use cases. In figures 5.1 to 5.5 the vertical arrows show the responsibility domain of the different organizations while the rounded rectangles represent different environments where the actual phases of the process are executed.

5.2 Roles

This clause contains the description of the roles or actors of the use cases. In the use case description only the cardinality of the roles are described.

Table 5.1

Role/Actor	Description
VNF Provider	Makes a virtualised Network Function (VNF) available for another role to use. The commercial terms of use are out of scope. As examples, the VNF could be a proprietary commercial product, the output of an open source community, the output of an internal development organization. A VNF Provider is typically the developer of the VNF.
VNF Operator	The entity that operates a VNF on an NFVI.
VNF Validator	Is an entity with the responsibility of: • Testing of the VNF according to some specification. • Validation of certain characteristics of a VNF, which might include conformance to certain standards, certain performance characteristics or other KPIs.

5.3 Initiation of joint pipeline

5.3.1 Concepts

A joint pipeline can be built up between one or more VNF Providers, optionally one or more VNF Validators and one or more VNF Operators.

The target of the joint pipeline is to open up a bidirectional communication channel between a VNF Provider and a VNF Operator. A VNF Validator can be added to the pipeline as an optional actor if some of the validation tasks are done by a different organization then the VNF Provider or the VNF Operator.

There are cases when a section of the pipeline is set up from a VNF Provider to a VNF Validator or a VNF Operator and cases when a section of the pipeline is set up from a VNF Validator to a VNF Operator.

The preconditions, components interactions and post conditions are the same in cases when a VNF Validator is in one side of the initiation procedure and when a VNF Validator is not involved. In the descriptions of clause 5.3 a VNF Provider can mean either a VNF Provider or a VNF Validator while a VNF Operator can mean either a VNF Operator or a VNF Validator.

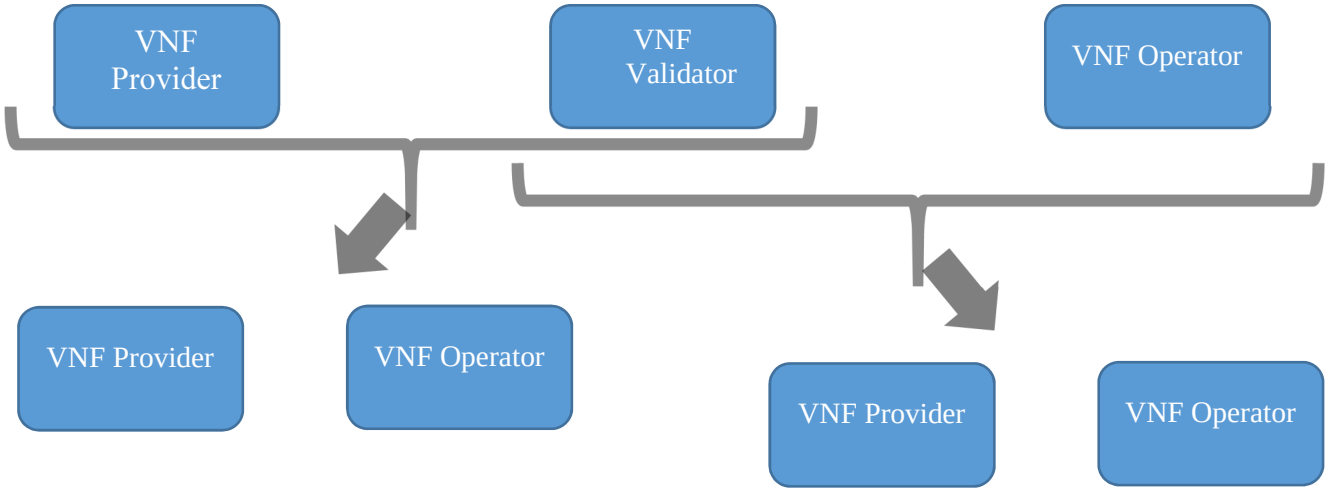

Figure 5.1: Usage of VNF Provider and VNF Operator based on the context of VNF Validator

5.3.2 Components

5.3.2.1 DevOps server

A component is needed between the responsibility domains of the different actors what is trusted by the actors. This DevOps server is responsible for the following tasks:

- Stage the VNF Packages and upload them to the next stage.
- DevOps workflow execution including the pre-checks of the NFVI, trigger the different testing phases, evaluate the testing phases, post event health checks of the VNF(s), sending feedback to the VNF Provider.

During these tasks the DevOps server interacts with the VNF Providers systems, VIM, NFVO, VNFM, VNF, EM and test automation systems using the interfaces defined in the ETSI MANO architecture. It is for further study if these interfaces need changes.

It is required to state, that based on the agreement between the actors a VNF Operator can have the following number of DevOps server instances:

- One DevOps server instance per VNF Provider.
- One DevOps server instance for the whole VNF Operator.
- One DevOps server instance per network domain of the VNF Operator.

In the last two case it is recommended to set a requirement for the DevOps server to support multitenancy, as several different VNF Providers artefacts, data and connections are handled in a single instance.

5.3.2.2 Data handling component

A component is needed to handle the data from the feedback channel from the VNF Operator to the VNF Provider. It is recommended that the following tasks are required from the data handling component:

- Enforce policies for the datastream from the VNF Operator to the VNF Provider.
- Filter sensitive data from the datastream from the VNF Operator to the VNF Provider.

During these tasks the data handling component interacts with the VNF Providers systems, VIM, NFVO, VNFM, VNF, EM and test automation systems using the interfaces defined in the ETSI MANO architecture. It is for further study if these interfaces need changes.

It is required to set a requirement to always host the handling component always hosted in the VNF Operators network and never in the premises of VNF Validator or VNF Provider.

It is recommended to state, that based on the agreement between the actors a VNF Operator can have the following number of data handling instances:

- One data handling component instance per VNF Provider.
- One data handling component instance for the whole VNF Operator.
- One data handling component instance per network domain of the VNF Operator.

In the last two cases it is recommended to set a requirement for the data handling component to support multitenancy, as several different VNF Providers data and connections are handled in a single instance.

It is recommended to state, that optionally the data handling component can be integrated to the DevOps server.

5.3.3 Preconditions

- A contractual relationship between the actors including an agreement on initiating a DevOps pipeline, accessing the staging environment and automatically installing upgrades to the staging environment.
- An agreement from the actors, that they can build up network connections between each other.
- A schematic blueprint of the pipeline including the number of handoff points between the actors.

5.3.4 Interactions

The DevOps server is deployed and configured to handle the new VNF Provider.

The endpoint of the feedback channel required in clauses 5.7.3 and 5.7.4 are configured in the DevOps server.

The policies and data masking rules of the feedback channel are configured in the DevOps server.

It is recommended that a requirement to be specified that if the VNF Operator user a test automation framework it should be connected to the DevOps server.

It is recommended that a requirement to be specified that optional connections to the VNF Providers digital marketplace, dashboard of new features, dashboard of faults overview should be configured to the DevOps server.

It is recommended that a requirement to be specified that optionally a direct communication channel between the personnel of VNF Operator and VNF Provider can be established. This direct communication channel can be used to discuss issues related of the VNF-s provided by the VNF Provider and operated by VNF Operator and can be for example a chat channel.

It is recommended that a requirement to be specified that workflow definitions for the different VNFs from the VNF Provider should be uploaded and configured to the DevOps server. It is recommended that a requirement to be specified that the description of the workflows should be capable to describe the different components of the pipeline, actions of the components, a relation between the components and criteria to move the pipeline from one component to the next. It is recommended that a requirement to be specified that the DevOps server is capable to execute the workflow definitions.

It is recommended that a requirement to be specified that the initiation of the pipeline should be tested.

5.3.5 Post conditions

DevOps pipeline is operational, capable to deliver artefacts from the VNF Provider to the VNF Operator and capable to provide feedback data from the VNF Operator to the VNF Provider.

5.4 Delivery

5.4.1 Single VNF Provider to single VNF Operator

5.4.1.1 Roles

One VNF Provider and one VNF Operator.

5.4.1.2 Preconditions

- A contractual relationship between the actors including an agreement on accessing the staging environment and automatically installing upgrades to the staging environment.
- Joint pipeline is initiated between the VNF Providers and the VNF Operator.

5.4.1.3 Interactions

The use case is limited to the case of VNFs being installed in the operator environment and with it excluding anything on the network service composition, testing, and operation. The tested VNF by the VNF Provider is automatically deployed into the staging environment of the operator. Continuous monitoring allows the VNF Provider to get data about the VNFs functional and non-functional behaviour in the operator environment.

Variants: An operator might have a two stage staging environment, one where VNF Providers can access and another one where there is already limited production traffic/trial traffic running. In this case the monitoring in stage 1 might be open, and the results of the monitoring in the second stage might be provided to the VNF Provider anonymized/changed/process.

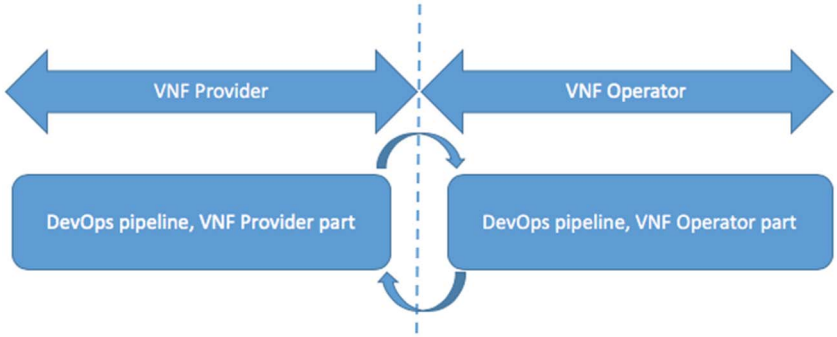

Figure 5.2: Illustration of the actors and processes

5.4.1.4 Post conditions

- The VNF runs in production and is updated regularly with new releases and bug fixes.

5.4.2 Multiple VNF Providers to single VNF Operator

5.4.2.1 Roles

Several VNP Providers and one VNF Operator.

5.4.2.2 Preconditions

- Joint pipeline is initiated between the VNF Providers and the VNF Operator.
- Each VNF Provider has his development and testing environment, meaning he has a test infrastructure he deploys his software on for testing purposes.

- On the VNF Operator side, since the discussion is now about components from different suppliers, another testing infrastructure is required and the test code is not for the components but for the combined integrated VNF or NS.

5.4.2.3 Interactions

- Option 1: the components from a supplier can delivered to the operator on a more traditional release by release schedule. The operator deliberately decides when he upgrades to a new release of the component.
- Option 2: more interesting for agile development and quick addition of new features is that the component is quite often shipped/pushed to the operator test infrastructure. That has the benefit, that the developer gets quite quick feedback of problems of his code in an operator environment, which supposedly looks very similar to the production environment.

Depending on the degree of change the level of testing might be different, however, since testing is assumed to be automatic, the tests can be run all the time as long as the change frequency is smaller than the time for the test is required.

Pre-conditions are a contractual relationship between the actors including an agreement on the frequency of deploying new functions and whether it is a pull or push action.

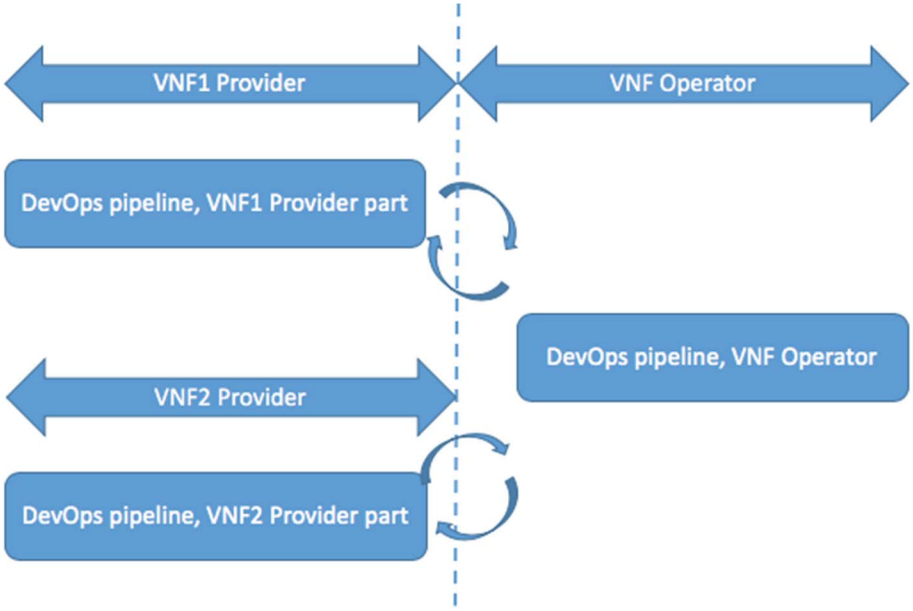

Figure 5.3: Illustration of the actors and processes

The test environment might be a joint test environment between operators and suppliers or it might be different instances of test environments. For example, a test setup at a supplier has X number of nodes, but at the operator it is Y number of nodes, which might make a difference from a performance and resilience perspective.

5.4.2.4 Post conditions

The VNF or NS runs in production and is updated regularly with new releases and bug fixes.

5.4.3 Single VNF Provider, single VNF Validator to single VNF Operator

5.4.3.0 Introduction

In other use cases the role for certifying a VNF is attached to the VNF Provider or the operator takes those risks himself. Here the assumption is that the role of a VNF Validator is introduced, which is different from VNF Provider. The benefits of using VNF Validator may be many, including time saving and knowledge sharing of testing and certifying.

5.4.3.1 Roles

One VNF Provider, one VNF Operator and one VNF Validator.

5.4.3.2 Preconditions

- A contractual relationship between the actors including an agreement on accessing the staging environment by the VNF Validator and automatically installing upgrades to the staging environment, per receiving them from VNF Provider and certifying them.
- Joint pipeline is initiated between the VNF Provider and the VNF Validator and between the VNF Validator and the VNF Operator.
- VNF package ready for onboarding.
- VNF Validator has the details of the Operator onboarding interface.

5.4.3.3 Interactions

The use case is limited to the case of VNFs being tested and certified prior to installation in the operator environment and with it excluding anything on the network service composition, testing, and operation. The tested VNF by the VNF Provider is provided to the 3rd party VNF Validator as VNF package ready for onboarding. The VNF Validator performs its own acceptance and validation testing including, VNF package integrity and completeness validation, onboarding, security, orchestration ability and manageability and functional testing. Then, the certified VNF is automatically deployed into the staging environment of the operator. Continuous monitoring allows the VNF Provider to get data about the VNFs functional and non-functional behaviour in the operator environment.

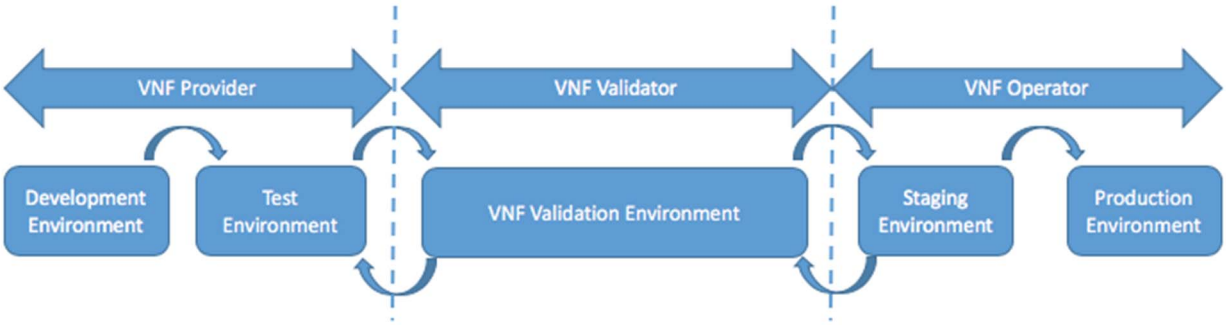

Figure 5.4: Illustration of the actors and processes

5.4.3.4 Post conditions

- The VNF package is validated and marked as certified.
- The VNF runs in production and is updated regularly with new releases and bug fixes.

5.4.4 Multiple VNF Providers, single VNF Validator to single VNF Operator

5.4.4.1 Roles

Multiple **VNF Providers**, one VNF Operator and one VNF Validator. The VNF Validator might also assume the larger role of hosting a distribution outlet for VNFs (similar to the "app-stores" that serve popular computing and communication platforms today), in which case there would be more than one VNF Operator.

5.4.4.2 Preconditions

- A contractual relationship between the actors (VNF Providers, VNF Validator and VNF Operator) including an agreement on the frequency of deploying new functions and whether it is a pull or push action.
- Joint pipeline is initiated between the VNF Providers and the VNF Validator and between the VNF Validator and the VNF Operator.
- VNF packages from VNF Providers ready for onboarding.

5.4.4.3 Interactions

Each VNF Provider has their own development and testing environment, meaning he has a test infrastructure he deploys his software on this for software testing purposes. The tested VNF from the VNF Provider is provided to the 3rd party VNF Validator, which performs their own testing consisting of e.g. automatic onboarding, VNF package completeness, security, orchestrateability and manageability, functional testing and interoperability testing. This testing is performed on an NFVI platform of similar configuration, design, and function to those used by both the VNF Provider for development and the VNF Operator for production network deployment. The interoperability test code is not for the components but for the combined integrated VNF or NS (not for the VNF components).

There are two options for VNF updates following the initial Production deployment:

- Option 1: the components from a supplier can be delivered to the VNF Validator and consequently to the operator on a more traditional release by release schedule. The operator deliberately decides when to upgrade to a new release of the component.
- Option 2: more interesting for agile development and quick addition of new features is that the component is quite often shipped/pushed to the VNF Validator test infrastructure. That has the benefit, that the VNF Provider's developers get rapid feedback of problems identified by the VNF Validator environment, which should reflect the production environment.

Depending on the degree of change the level of testing might be proportional in coverage. However, since testing is assumed to be automatic, the tests can be run continuously as long as the change frequency is smaller than the time for the test is required.

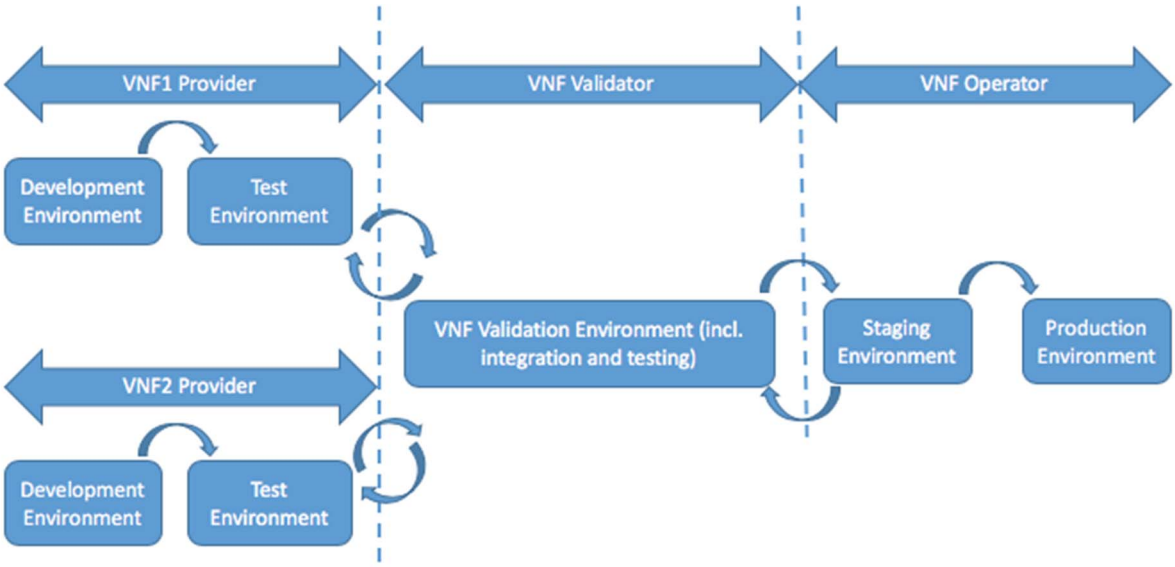

Figure 5.5: Illustration of the actors and processes

The test environment might be a joint test environment between VNF Validator and suppliers or it might be different instances of test environments. For example, a test setup at a supplier has X number of nodes, but at the VNF Validator it is Y number of nodes, which might make a difference from a performance and resilience perspective.

There may be a different models of relations supported between an operator, a VNF Validator and a VNF Providers (s) e.g.:

- One VNF Provider, One VNF Validator, One VNF Operator (described by the clause 5.1).
- Several VNF Providers (two in the example given in the present document, but may be extended to n), one VNF Validator, one VNF Operator.
- One or several VNF Providers, one VNF Validator, several VNF Operators having the same Platform requirements. In this case the process will be the same as described by the clauses 5.4.1 and 5.4.2, but provided for several operators.
- In the cases with multiple VNF Providers and the VNF Validator also performs the distribution task, the VNF Operator may select a specific VNF from those available that perform the desired function.

5.4.4.4 Post conditions

- The VNF package is validated and marked as certified.
- Continuous monitoring allows any of the three actors to obtain data about the VNF's functional and non-functional behaviour in the operator environment (with the operator's permission).
- The VNF or NS runs in production and is updated regularly with new releases and bug fixes.

5.5 Error in production

First of all, the topic of errors in production is a critical issue and difficult to handle since there are many challenges including the following:

- What is an error? And what severity level does an error have? Or when does an error report being sent to the developer makes sense?
- How much context from around the VNF is needed that an error report is beneficial for the developer? The environment in production is typically complex and hard to describe, so it is difficult for the developer to have a similar environment to reproduce the error.
- The complete real traffic causing the error might not be captured and it might be difficult to be sent as well, so that might make the debugging difficult since the error might not be reproduceable in another environment.
- Production data is very sensitive and therefore extra care should be taken to not leak those.
- VNFs in production are typically running as part of a service, so the service context might be important, but difficult to expose to the developer.
- If a VNF is running as part of a service together with VNFs from other vendors it might be forbidden to expose with what VNFs the interaction is happening and how, since that is part of the service description and not viewable in the VNF itself.

Despite many of those challenges it makes sense to have a feedback channel for production VNF error reports being automatically or with human intervention being sent to the developer.

5.6 Tearing down of joint pipeline

5.6.1 Preconditions

- Joint pipeline is initiated between the VNF Providers and the VNF Operator.
- An agreement to terminate the DevOps pipeline.

5.6.2 Interactions

It is recommended that a requirement is set that the feedback channel from the VNF Operator to VNF Provider should be closed down.

It is recommended that a requirement is set that the optional connections to the VNF Providers digital marketplace, dashboard of new features, dashboard of faults overview should be closed down.

It is recommended that a requirement is set that if there is one DevOps server per VNF Provider in the VNF Operators network or this is the last VNF Provider the DevOps server serves both the data of VNF Provider and VNF Operator should be exported from the DevOps server and the DevOps server should be undeployed.

It is recommended that a requirement is set that if there are more VNF Providers served by the DevOps server the data of VNF Provider should be exported and deleted together with all related configuration in the DevOps server.

5.6.3 Post conditions

DevOps pipeline teared down. Connections between the VNF Provider and VNF Operator are closed.

5.7 Common functionalities

5.7.1 Pull of VNF Package

5.7.1.0 Introduction

The VNF Operator pulls the most recent version of the VNF Package from the VNF Provider.

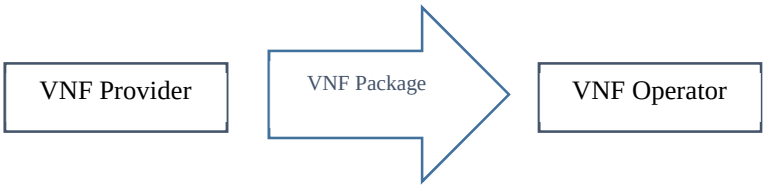

Figure 5.6: VNF Operator pulls the VNF Package from the VNF Provider

5.7.1.1 Roles

A VNF Provider and one VNF Operator.

5.7.1.2 Preconditions

The Provider has at least one version of VNF Package available.

The VNF Operator knows some identifier so that it can select the correct VNF Package - e.g. the VNF Package for the Firewall vs the VNF Package for the virtualised IMS.

The VNF Operator knows where the VNF Package is located e.g. URL, file system directory, etc.

The VNF Operator is authorized to acquire the VNF Package. The authorization may be assumed to occur through some out of scope commercial transaction and licencing agreements.

5.7.1.3 Interactions

1) VNF Operator requests VNF Package from VNF Provider.

NOTE: An optimization might be to request only those elements of the VNF package that have changed from a prior version of the VNF package.

2) VNF Provider validates authentication:

   a) If authentication invalid return error.

   b) If authentication valid deliver VNF Package with relevant test case set, test execution framework (test architecture, test tools, etc.) and test results to VNF Operator.

3) After delivery, VNF Operator validates VNF Package:

   a) VNF Operator validates integrity of VNF Package (e.g. checksum).

   b) VNF Operator validates completeness of VNF Package (e.g. all the expected metadata and executables present).

   c) VNF Operator validates provenance of VNF Package.

   d) VNF Operator stores executables, test case set and test execution framework (test architecture, test tools, etc.) and metadata ready for further testing.

5.7.1.4 Post conditions

VNF Operator has VNFC executables in image storage, have the test execution framework (test architecture, test tools, etc.) and the test case set ready for testing:

- Error if not all the expected VNFCs executable images are available.

The VNF Operator has parsed the metadata in the VNF Package metadata has been parsed.

The VNF Operator has the information required to successfully deploy the VNF:

- Error if VNF metadata missing, incomprehensible or inconsistent.

VNF Operator ready to perform additional testing on this version of the VNF to confirm viability for service operation.

The VNF Provider does not need to stop the development of the VNF Package in a new version.

5.7.2 VNF Package operability validation

5.7.2.0 Introduction

The VNF Operator performs various tests to validate that the current version of the VNF Package will be operable when deployed in service. The specific set of tests required may vary due to business requirements, the results of tests on previous versions, the scale of the feature changes from previous versions, etc. These tests could include:

- Portability to the Operator's NFVI - e.g. BIOS variants, acceleration hardware, SRIOV, etc.
- Performance benchmarking - e.g. does the VNF provide the expected performance in the operator's NFVI.
- Resilience testing - e.g. can the VNF continue normal operation despite failures in the NFVI.
- Isolation - e.g. does the VNF behaviour impact other NFVI operations.
- Stability - e.g. does the VNF operate normally under load for an extended period of time.
- Security - e.g. virus scanning.

5.7.2.1 Roles

One VNF Operator.

5.7.2.2 Preconditions

The VNF Operator has the VNF ready for deployment on the NFVI.

The VNF Operator has the test configurations ready for deployment. The test all have explicit success criteria defined.

5.7.2.3 Interactions

The specific test selection and sequence are beyond the scope of the present document.

The VNF Operator deploys the VNF.

The VNF Operator performs the tests and records the results.

5.7.2.4 Post Conditions

The VNF Operator has a record of which tests passed and which tests failed.

5.7.3 Provide test result

5.7.3.0 Introduction

The VNF Operator provides information on executed tests to the VNF Provider. The test can be made at different stages on the VNF Operator infrastructure. The tests can be executed in the testing, pre-production, or production environment.

The set of test cases can be standardized tests, test cases from a 3rd party test provider, or operator internal tests. Depending on the tests executed, there might be another mechanism, a different filter, or a different place for transferring the VNF Test report to.

Figure 5.7: The VNF Operator delivers the test results to the VNF Provider

5.7.3.1 Roles

One VNF Provider and one VNF Operator.

5.7.3.2 Preconditions

VNF Operator has VNF Test Reports to share with VNF Provider.

5.7.3.3 Interactions

1) VNF Operator identifies the VNF Provider to which the VNF Test Report should be delivered.

2) VNF Operator prepares the VNF Test Report for that VNF Provider.

NOTE: The VNF Test Report preparation may need some redaction and anonymization e.g. to respect various privacy regulations.

3) VNF Operator delivers the VNF Test Report to the VNF Provider.

5.7.3.4 Post Conditions

The VNF Provider received the VNF Test reports for the VNFs that it has developed. The details of the VNF Provider's development process are beyond the scope of the present document. The life cycle of VNF Test Reports is beyond the scope of the present document, but it may be anticipated that some VNF Test Reports may be associated with future versions of the VNF Package to as an indication of issue closure.

5.7.4 Operating dashboard

5.7.4.0 Introduction

The VNF Operator provides continuous information on operation results of the VNF-s to the VNF Provider. Since this case is about operational results being sent, while the VNF is running in a pre-production or production environment. Therefore, the operational data needs to be handled with more care compared to results from test environments.

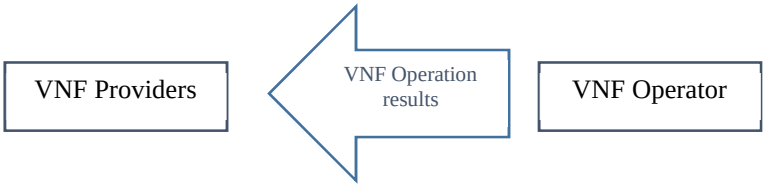

Figure 5.8: The VNF Operator provides the operations results to the VNF Provider

5.7.4.1 Roles

Many VNF Provider and one VNF Operator.

5.7.4.2 Preconditions

VNF Operator has VNF operation results to share with VNF Provider.

5.7.4.3 Interactions

1) VNF Operator identifies the VNF Provider to which the VNF operation results should be delivered.
2) VNF Operator starts streaming the operation results for that VNF Provider.

5.7.4.4 Post Conditions

The VNF Provider receives the continuous VNF operation results for the VNFs that it has developed. The life cycle of the delivered VNF operation results is beyond the scope of the present document.

6 Test steps and approach

6.1 Step 1: Test Definition

In DevOps models, it is best current practice that tests are written such that the functionality can be checked and tested whether it does what it is supposed to do. In the case of cross-organizational DevOps, that also means there needs to be an agreement between the different roles about what the functionality of a VNF is and how that is tested.

Since there are several levels of granularity possible, for each level there are test definitions needed. For example, for a Network Service (NS) the test definitions specify the behaviour of the network service.

There are several options on how automated tests can be defined:

1) The tests can be directly coded as separate software.

2) They can be specified with a test specification language and run using the appropriate interpretation/compilation engine.

The tests can be defined by the provider and given to the VNF operator, or it can be commonly defined by VNF provider and VNF operator, or the tests are defined by the VNF operator and communicated to the VNF provider. The latter is specifically important in the case that the tests are classified as acceptance tests of the VNF.

6.2 Step 2: Code/VNF Package Shipment

The traditional model is that a supplier is packaging some software and ships it to the customer (using a DVD, FTP site to download, etc.). See also ETSI GS NFV-REL 006 [i.1] for further information on that.

Since in DevOps the basic storage system is a repository (software code or VNF package) and additions to the repository are continuous, there are several options on how this can be handled based on preferences and business arrangements:

1) Operator-triggered: If a new VNF package is found in the supplier repository, the VNF is automatically downloaded by the operator. The frequency of the VNF package query is an operator controlled interval. The VNF package gets pulled from the supplier and gets integrated into the CI/CD pipeline at the operator. In an ideal case, the tests are automatically started, however, that might not be desirable in cases of very long query intervals. It is recommended to deploy every new delivery to make the integration more easy and to provide fast feedback to the vendors.

2) Supplier-triggered: A notification of new VNF package version is sent from the supplier to the operator. This event triggers the pull of the new VNF version into the CI/CD pipeline of the operator, which then initiates the automated testing. The operator can reserve the right to approve the inclusion of the new VNF package into the pipeline, however, this would introduce a manual step into the automated process.

Also the VNF package as such does not need to be only binary/disk image/VM image/etc. but can also be directly source code, depending on the contractual agreements in place. This is specifically true when open source code of certain licences are involved. VNF package should contain a test set, a test execution framework (test architecture, test tools, etc.) and test reports executed by the VNF vendor.

Shipment of new Code/VNF Package should trigger a new DevOps cycle that includes all necessary testing and deployment steps agreed in the contract between the VNF Provider and the VNF Operator.

The frequency of packaging and shipping new versions needs to be configured for supplier and operator triggered options and depends on the level of novelty an operator wants to see in his test and production environment.

6.3 Step 3: Automated Test Execution

The tests are performed on a test infrastructure. So the deployment target is a test infrastructure internally or external to the operator. Those tests might include an entire network service, the isolated VNF, all of which are running on the operators' NFVI and MANO configuration. Test results are fed back to the provider of the function or the developer of the network service.

The tests might contain recommended test procedures (as outlined in documents like ETSI GR NFV-TST 004 [i.5] and ETSI GS NFV-TST 001 [i.4]) or operator specific tests, environment specific tests, or implementer specific test depending on the agreements and contractual setting and the level of testing foreseen by the operator.

For a certain set of tests there is a need for having emulated end-systems such as emulated mobile phones connecting to a core network. So in the test environment a set of emulated end-systems need to be started and run against the VNF or NS to be tested. Those emulated end-systems might need to be configured to fit the environment and the test setup. Also for scalability testing the number of end-systems might need to be changed during the runtime of the test. For system or integration tests, emulated end-systems should be used only in the case when testing with real devices is not possible.

The frequency of test executions depends on the frequency of changes happening in the VNF code. The tests can be executing in a regular time interval, some open source project run the test and with it the packaging daily.

6.4 Step 4: Moving to Production

The steps involved from testing to production are assumed to be very operator specific and therefore difficult to specify in detail. Also many service providers applying DevOps today do not have a fully automated pipeline from testing to production. It involves some human decision points and eventual manual tests and checks.

One best current practice with moving from testing to the staging environment is that in staging there is a small part of traffic/user load hitting the functions to test under more realistic contexts in a realistic environment. So the main difference is moving from emulated test traffic to real traffic but in small scale and with a small impact in case of failures. For example, traffic of a friendly user group could be used to run through those functions.

VNF Operators request an accepted documentation about the intended changes in their labs before the changes can be applied. As in a DevOps environment these changes are triggered and done automatically these workflows should be checked and approved before the DevOps pipeline is initiated between the parties.

Then moving from staging to the production environment is basically scaling it up for large traffic volumes of real traffic. And also in this step manual checks and tests might be used before the full load is switched over. High volume of emulated traffic can be used first on the production platform before real traffic is moved to the installed functions.

From the staging as well as from the production environment it is assumed that logs and errors are feedback to the developer for improving the code in the next stages.

6.5 Step 5: Collecting operational data

There should be an interface where the VNF Operator can provide operational data to the VNF Provider about the VNF. This data can include files, like (rotated) logfiles, post mortem analysis data, configuration files specific to the deployment, or streamed data like log, tracing or monitoring streams. The data provided from the VNF Operator to the VNF Provider should be obfuscated, so no sensitive data is handed over.

6.6 Test Function Packaging and Shipment

This clause handles the packaging and shipment of test functions (ETSI GS NFV-IFA 011 [i.3]). Various options can be foreseen:

1) The Test Function can be packaged together with the VNF Under Test (A single VNF package including test functions and VNF under test).

2) The Test Function can be a separate VNF with its sole purpose to test a VNF or NS.

The sourcing of the Test Function can also differ. In case of the Test Function being packaged together with the VNF under test there is not much of a choice. However, in all other cases the Test Function can be sourced from third parties.

6.7 Installing and Running Test Functions

Installing Test VNF means to instantiate the Test VNF and configure it such that it can use the VNF under Test appropriately (VNF dependent) in a particular environment. In addition, the VNF under test most likely should to get a test configuration as well. Finally, MANO or a test controller can instantiate Test VNF together with the VNFs like a special purpose Network Service with the right networking and other resource configurations.

From an Orchestration perspective, it might need to be clear that this is not productive workload and it might be scheduled accordingly.

6.8 Keeping Track of already done tests

As shown in the use cases there are several constellations of roles in a full chain. Therefore, it might make sense to have a track record of what tests have been made successfully from one role to the next. For example, a VNF provider naturally does its own tests anyway and could add that information into the VNFD including the platform description this has been run on. However, since it is likely that the platforms differ from installation to installation it might be wise to just redo the testing.

In addition, in case a certain test has guarantees associated with like "conforms to" or "works guaranteed on a certain setup" that information is added and should not be changeable.

Having test results being part of the VNF package would be an optimization not to redo them again and again.

6.9 Format of Results in Report

As described above there is a need to report back test results in order to improve the code of the VNF under test.

The issue with defining reports is that there are that many diverging requirements on what needs to be reported. On the other hand, there is certain information in the report of importance. Naturally, the VNF under test with its configuration and the environments it was running on. Also, the list of test cases successfully run or failed. Any other logging or VNF specific information for debugging or helping to improve the VNF.

7 Recommendations

7.0 recommendation

The following features or requirements for features are recommended to add to future feature sets of NFV-MANO.

7.1 Structure of a VNF Package

7.1.1 VNF Package

It is recommended that a requirement be specified for the VNF package to contain references to dependent functions or services, which are machine readable and can be bound automatically to the particular VNF at deployment time into the test, staging, production environment. (Late Binding of Dependencies.)

It is recommended that a requirement be specified for the VNF package to contain a testing section with various information concerning testing and DevOps (see below for more details). Structurally, it might be that it is a full information element or that it is a reference to testing related information or to a separate VNF implemented for testing.

7.2 Description of VNF Package content

7.2.1 VNF Package

It is recommended that a requirement be specified for the VNF Package that it should contain acceptance test (see ETSI GS NFV-IFA 011 [i.3]) descriptions. It might be possible that several test case descriptions of the acceptance tests are received from various VNF Operators as part of a Test Driven Development process.

It is recommended that a requirement be specified for the VNF Package to contain acceptance test code that can be executed automatically after deployment into the various environments. Typically, it is environment specific what tests are made.

It is recommended that a requirement be specified for the VNF Package that should contain information about a test execution framework what is capable to execute the test code included in the VNF Package. The test execution framework specific information can be any of the following four:

- A reference to test execution framework what can be installed from online, instructions to install the test execution framework, resource needs of the test execution framework and instructions to run the acceptance test code with the test execution framework. The information in the VNFD should contain everything to ensure that both the download and installation of the test execution framework and the execution of tests are fully automatic.
- A reference to the VNFD of a test VNF which automatically starts and executes the acceptance test code on deployment.

- A test VNFC which automatically starts and executes the acceptance test code on deployment.
- A trigger what should be executed on the deployed VNF, so the VNF runs the acceptance test code itself.

It is recommended that a requirement be specified for the VNF Package that it should contain a reference to the acceptance test results of the VNF Provider. These results should indicate if the acceptance tests were executed successfully or not.

It is recommended that a requirement be specified for the VNF Package that it should contain information about the operator environment, in which it makes sense to test and where it is low risk to test.

NOTE: For example, some acceptance tests are executable and make sense to only execute in pre-production environment, while others are executable in a production environment in case the risk of failure is assessed to be lower.

The impact of testing in live networks is not part of the present document, it will be handled in future versions.

7.3 VNF Identification

7.3.1 VNF Package

Since versions of software will be a very high number, the identification scheme recommended to be defined in a way, that it allows for large numbers of versions.

7.4 Test Execution of Test Functions

7.4.1 Modelling testing code as Test VNFs

7.4.1.1 Description

The assumption in this option is that the VNF under test is provided as a production VNF package.

In the option of modelling testing code as separate Test VNFs, implemented in whatever way, the test execution is sort of a special version of installation and configuration of one or more VNF in a special context intended for testing only.

This can be modelled as a network service for testing purposes. So the VNFs under test are installed as is, but the network service consists of the VNF under test and one or more test VNFs being part of the setup.

The Test VNF needs to be configured such that they are associated with the VNFs under test and might need to get environment specific configurations.

The VNF under test might need special configurations for testing purposes. For example, it might be that the Virtual Links are changed to a test VNF instead of the next hop VNF. Also for testing a VNF, the surrounding systems might need to be emulated and therefore needs a special configuration.

For certain tests the Test VNF might need a re-configuration in order to have different test parametrizations.

7.4.1.2 Test Network Service

For the option of combining VNFs under test with Test VNFs running as a network service, the specification of NSD needs to be enhanced with additional test related information. At least some information that the NSD is only for testing purposes such that the resource management functions in MANO can de-prioritize testing over other work load in the infrastructure.

NOTE: Test Network Service is for further study.

7.4.2 Modelling test as being part of a VNF package

7.4.2.1 Description

In the option of modelling tests as part of the VNF package with information contained in the VNFD, the execution of the test needs to be triggered through special call to the part of the VNF running the test. Common practice should be that the MANO stack is triggering testing after installation. There might be configuration or state associated with the VNFD that only a limited number of tests are done in case the update of the VNF software is marginal.

7.4.2.2 VNF Package

It is recommended that a requirement be specified for the VNF Package to be able to store the description of the VNF in normal and in test mode including an optional set of test VNFCs and test configurations.

The additional set of test VNFCs can be organized into an additional VNFD in the VNF Package or to a test specific Deployment Flavour in the VNFD. It is recommended to consider these alternatives during the specification work following the requirement specification.

7.4.2.3 VNF

It is recommended that a requirement be specified for the VNF to be able to differentiate between its deployment in normal and test mode. This is needed when the VNF is running its own tests as it is described in the last item of the bullet list in clause 7.2.1. The behaviour of the VNF in test mode is VNF specific.

7.5 Acceptance test feedback to VNF provider/developer

7.5.1 Description

It is recommended to specify an interface between the VNF Operator/VNF Validator and the VNF provider for communicating test results and test feedback in one or several reports.

Depending on the contractual relationship various degrees of details in the content of the reports sent to the VNF Provider can be foreseen.

After test execution, there needs to be a filter for sensitive data in the test report. It is recommended to specify requirements about the capability to filter and the capability for configuration of the filter.

It is recommended to specify requirement about the mechanism for policies of the VNF Operator for decisions on sending the report automatically or after a human assessment. Therefore, it is not the Test VNF itself sending back reports, but this should be under control of the MANO stack. It might be architecturally easier that the test process is also steered through the MANO process.

The format of the reports about test results might need to be standardized such that it can be commonly used across the various implementations and across the various internal processes of each partner in the specified role.

7.5.2 VNF

It is recommended that a requirement be specified for the VNF to be capable to provide the information about itself what is considered useful by the VNF Provider. This feedback data can be any test results, logs, etc., from a test, staging, or production environment. The feedback data should be sent to the VNF Provider either automatically or after a human assessment based on operators policies. The feedback data should contain notifications on whether tests have passed of failed. In case of failure ideally the logs or other information is essential for developers to fix the problems. Also a description of the environment used helps developers to figure out what the problems are.

The location where the feedback data should be forwarded should be read from OSS during the instantiation of the VNF.

7.5.3 OSS and EM

It is recommended that a requirement be specified for the OSS to be capable to receive the feedback from the VNF via the EM, execute the filtering of operator sensitive data based on the policies of the VNF Operator and forward the feedback to the VNF Provider.

7.6 Test Specification Languages

7.6.1 Description

Test specification language are built to write tests in a domain specific language and running that language (interpreted or compiled) on an execution environment.

In the option of modelling testing code as Test VNFs, that Test VNF would contain the test specification in defined format plus the execution environment of that particular language combined in one Test VNF.

In the option of modelling tests as part of the VNF package with information contained in the VNFD, it might be that tests are defined in a standardized test specification language and the compiler or interpreter of that language would need to be part of the NFV platform.

7.7 VNFC Software Component Update and Upgrade

7.7.1 Description

For the case of frequent software updates, it is more optimal to update parts of the VNF, for example on a per VNFC granularity, instead of the whole VNF. For example, to avoid the frequent delivery and onboarding of big VNF Packages, the VNF Package need to provide a possibility to deliver only parts of the software of a VNF or a VNFC.

Whether there is a need to do software updates and upgrades on per sub-VNFC granularity is for further study. However, note that the VNF developer can choose the VNFCs appropriately to have small enough upgradable component.

The consequence of updatability and upgradability of VNFCs is that VNFCs are exposed to the CI/CD pipelines and that versioning of the VNF itself is properly designed.

Specifically for software updates it makes a lot of sense to change only part of the software of a VNF, whereas for software upgrades adding new functionality or changes in the interfaces or APIs a full new version of the VNF could be more beneficial including management of the dependencies might be needed.

This recommendation covers same problem as the VNF software update use case described in clause A.5 of ETSI GS NFV-REL 006 [i.1] except that it is done on a higher frequency and on a VNFC granularity.

7.7.2 VNF Package

It is recommended that a requirement be specified for the VNF Package to provide a possibility to contain only parts (VNFC) of the software of a VNF. These partial VNF Packages should have a reference to VNF Package versions to where they are applicable.

It is recommended that a requirement be specified for the VNF Package to provide a possibility to deliver one or more VNFCs of a VNF separately. These VNFCs are a list of binary blob, metadata and installation script triples.

It is recommended that a requirement be specified for the VNF Package to contain information about the VNFCs contained such that independent VNFC updates and upgrades are enabled.

It is recommended that a requirement be specified for the VNF versioning being adapted to allow for proper mechanisms to reflect an updated or upgraded VNFC also in the VNF version.

7.7.3 NFV-MANO

It is recommended that a requirement be specified for the NFV-MANO to be able to recognize the fact when a partial VNF Package is delivered. In this case NFV-MANO should be able to construct a complete VNF Package from the just delivered partial VNF Package and from the already available baseline VNF Packages. The resulted merged VNF Package could be used as a baseline VNF Package in the future.

It is recommended that a requirement be specified for the NFV-MANO to be able to update the VNFCs of an instantiated VNF delivered as partial VNF package.

It is recommended that a requirement be specified for the NFV-MANO to be able to recognize the fact when a VNF is updated on VNFC software component level. In this case NFV-MANO can execute the update using the provided installation scripts.

It is recommended that the NFV-MANO supports updates and upgrades at the VNFC granularity.

It is also recommended that NFV-MANO allows for VNFs being combined with technology which may support for upgrades and updates at granularity smaller than the VNFC, in which case the methodology of that technology is used without exposure to NFV-MANO and is self-contained in the VNF or VNFC.

History

Document history		
V1.1.1	January 2020	Publication