D5.3 Integrated and qualified public release of SONATA platform

Project Acronym	SONATA
Project Title	Service Programing and Orchestration for Virtualized Software Networks
Project Number	671517 (co-funded by the European Commission through Horizon 2020)
Instrument	Collaborative Innovation Action
Start Date	01/07/2015
Duration	30 months
Thematic Priority	ICT-14-2014 Advanced 5G Network Infrastructure for the Future Internet

Deliverable	D5.3 Integrated and qualified public release of SONATA platform
Workpackage	WP5 Integration, system testing and qualification
Due Date	31/01/2017
Submission Date	6/02/2017
Version	0.1
Status	To be approved by EC
Editor	Dario Valocchi (UCL)
Contributors	Santiago Rodríguez (Optare), Aurora Ramos, Felipe Vicens (ATOS), Michael Bredel (NEC), Shuaib Siddiqui, Daniel Guija (i2CAT), José Bonnet, Alberto Rocha (ALB), Manuel Peuster, Hadi Razzaghi Kouchaksaraei (UPB), Sharon Mendel-Brin (NOKIA), Luís Conceição (UBI), Stuart Clayman, Alex Galis, Elisa Maini, Francesco Tusa, Dario Valocchi, Zichuan Xu (UCL), Geoffroy Chollon (TCS), Stavros Kolometsos, George Xilouris (NCSRD), Steven Van Rossem, Thomas Soenen, Wouter Tavernier (IMEC), Theodore Zahariadis, Panos Trakadas, Panos Karkazis, Sotiris Karachontzitis (SYN)
Reviewer(s)	George Xilouris, Panos Trakadas, Stuart Clayman.

Keywords:

prototype, integration, qualification

Deliverable Type		
R	Document	
DEM	Demonstrator, pilot, prototype	
DEC	Websites, patent filings, videos, etc.	
OTHER		**X**
Dissemination Level		
PU	Public	**X**
CO	Confidential, only for members of the consortium (including the Commission Services)	

Executive Summary:

This Deliverable documents the progress made in the WP5 of the SONATA project. WP5 focuses on defining processes and methodologies for an *agile* integration process of the output of the work done in WP3 and WP4. The presented results are relevant to the second open source release of the SONATA environment.

Deliverables D3.2 [7] and D4.2 [9] were submitted to the Commission in December 2016. These documents contain the implementation details of the modules developed for the SONATA SDK and the SONATA SP respectively, including the internal module architecture, APIs definitions, unit and module tests definitions. This Deliverable can be considered as the document accompanying the implementation effort of SONATA version 2.0 open source release, by verifying the proper integration within the qualification environment of the SONATA system. Given the importance of the integration and qualification aspect, this Deliverable also dedicates its two main sections to the detailed documentation of **Integration tests**, which have been designed and developed to ensure the expected behaviour of new features and new modules, and **Qualification tests**, designed as a mean of qualification and validation of the output of the integration phase. All tests are described in terms of the SONATA components involved, the designed interactions, and the specific requirements in terms of the infrastructure as well as the expected results.

Contents

List of Figures

1 Introduction

This Deliverable covers the second release of the integrated SONATA platform in a qualification environment, which includes not only the components developed in the first year of the project but also some newly added modules that are specially to cater new features in a 5G environment. It will also include the description of the qualification tests that are used to validate the outcome of our development effort against the Use Cases as they have been defined in WP2, in terms of the functional requirements and the key characteristics which have been derived for the use-cases. It will also include a description of new integration tests, which has been introduced for the new modules. In the following of this introduction, we will first briefly summarise the architecture of the SONATA platform for the sake of self-containedness of this Deliverable, and then describe its contributions.

As this Deliverable will include the integration tests, and the new features and software modules of the second release of SONATA, the architecture of the SONATA platform is briefly summarised as follows Figure 1.1. The main components of the SONATA architecture consist of two parts: (1) the SONATA Software Development Kit (SDK): this part offers functionalities and tools for the development and validation of virtual network functions and virtual network services, including an editor for service descriptors, an emulator to locally test developed services, monitoring data analysis tools, service storage, packaging and publishing tools; (2) the SONATA Service Platform (SP), which offers the functionalities to orchestrate and manage network services during their life-cycles with a MANO framework, interact with the underlying virtual infrastructure through Virtual Infrastructure Managers (VIM) and WAN Infrastructure Managers (WIM) to efficiently use a heterogeneous set of virtual resources, store available network services and virtual network functions with dedicated catalogues, represent the status of the deployed network services and network functions, of the virtual infrastructure, and of the virtual resources through a set of Repositories, which represent the information system of the SP, and offer the functionalities of the SP itself to the outside world, with a unique endpoint, called the Gatekeeper.

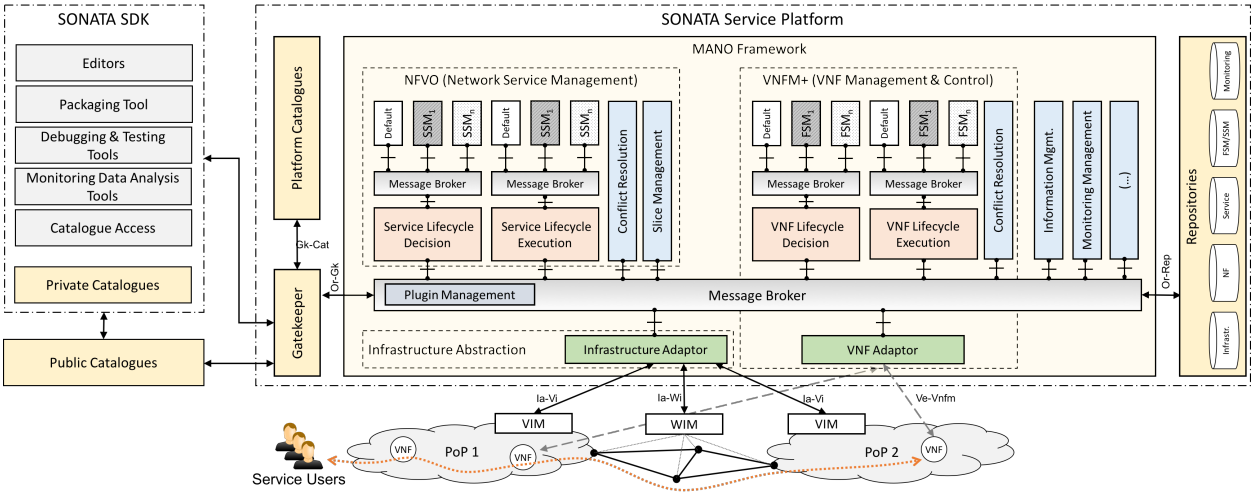

Figure 1.1: Detailed architecture of SONATA service platform

This Deliverable describes new modules as micro-services, keeping the flexibility in terms of scalability we have designed from the start. Each module will provide the most adequate kind of interface, either REST-based or message-based. It must be mentioned that some of these new modules are innovative (e.g., a container-based VIM, working in a MANO context), therefore deserving some more time of consolidation of the proposed solution. Detailed description on the software modules that are added/modified for these newly added features will be described in this Deliverable. New integration and qualification tests are needed to testify the efficiency and effectively of the integration of these new modules into both the SONATA SDK and service platform.

The work related to this Deliverable will also enable service providers to easily and and automatically add new service functions. This is done via designing a set of tests for the integration of the MANO framework, in particular the Service Lifecycle Management (SLM) plugin, the Function Lifecycle Management (FLM), and Infrastructure Adaptor. The purpose of the test is to trigger the SLM to instantiate a new service instance, and within this test various modules, including SLM, FLM, VIM adaptors, message broker, and SONATA SDK will interact in a systematic, efficient, and effective way.

The relation on the SONATA platform and other state-of-the-art platforms, e.g., Open Source Mano, are also briefly introduced in this Deliverable. Further, a list of future work is proposed, which include (1) the agility improvement of the SONATA platform, (2) integration of network slicing into SONATA platform with affected modules and beneficial modules, and (3) the integration with existing state-of-the-art platforms, such as Open Source MANO.

1.1 Deliverable structure

The document is organised as follows.

- Section 2 elaborates on the second software release of SONATA, which includes software modules in the service platform and the SONATA SDK;
- Section 3 illustrates the new integration tests defined for the new modules of SONATA;
- Section 4 describes the qualification tests;
- Section 5 includes a description on the relation between SONATA platform and other state-of-the-art platforms, consideration on the added value of our development approach, and a list of future work;
- Section 6 concludes the Deliverable.

1.2 Clarifications on SONATA qualification phase

The SONATA description of work [3] marks the second release of SONATA as a qualified software release. Software qualification could have a wide spectrum of meaning, depending on the requirements and on the development status of a software product. In the context of this project, we give two meaning to qualification:

1. Verify the outcome of the integration phase in terms of functional behaviour, performance, security and conformance, as described in Deliverable D5.2 [10];
2. Validate the outcome of the development through pilot use-cases.

In this document we are going to tackle the first aspects, which more strictly relates to the topic of software integration and to the scope of WP5 in general At the current status the Qualification Environment has been deployed in the project testbed and functional tests have been designed to validate end-to-end functional behaviour and conformance to the general requirements elicited from the use cases documented in Deliverables D2.2 [4] and D2.3 [5].

2 SONATA release 2.0

In this section we document the new features and functions implemented by the second software release of SONATA. The section illustrates, module by module, a development **delta** with respect to the SONATA 1.0, released in September 2016.

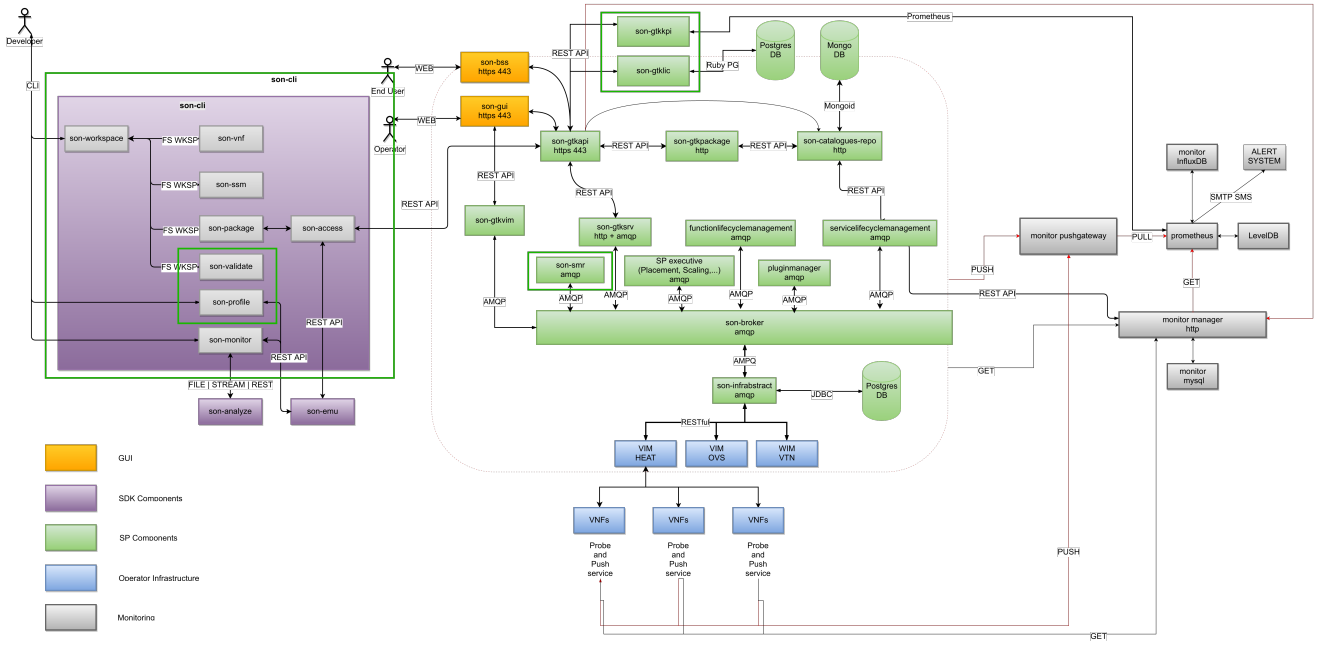

Figure 2.1: Modules and interfaces of SONATA 2.0. New modules marked in green.

Figure 2.1 shows an overview of the current status of the SONATA software. A higher quality version is available in Section Appendix A. The main new introductions have been marked in green rectangles. The SONATA SDK had incurred a major refactor in terms of module packaging. The `son-cli` module has been introduced to include the main sub-module which are accessed by the user through a command line interface. Two new sub-modules, `son-validate` and `son-profile`, have been introduced to allow validation of the packages submitted by developers and profiling of monitoring data coming from the emulator, respectively. Moreover the SDK local repository has been marked as deprecated, so that descriptors packages submitted by the users are always stored and managed remotely through interaction with a SONATA Gatekeeper. On the Gatekeeper side, two new sub-modules have been introduced: the `gtk-kpi`, which is in responsible for collecting metrics and key performance indicator on the Gatekeeper API usage; the `gtk-lic`, which is in charge of managing users accesses to service and function descriptors stored in the catalogue. Moreover security has been introduced to the interfaces of the main Gatekeeper's endpoint, the Gatekeeper GUI and BSS, which are now accessible only through HTTPS. Finally, a Specific Manager Registry has been integrated with the MANO framework, allowing the management of the so called Service Specific Managers (SSMs) and Function Specific Managers (FSMs). In general new features and improvements have been included to all the modules composing SONATA. Further details on the architecture and functionality of each module can also be found in Deliverables D4.1

[8], D3.1 [6], D4.2 [9] and D3.2 [7]. In what follows we give a brief resume on the most important of these functions for all the SONATA modules, to be used as reference documentation to the actual SONATA software release 2.0.

2.1 SONATA service platform modules

2.1.1 Gatekeeper

This section describes the improvement, in terms of WP5, in the Gatekeeper of the SONATA service platform.

We first introduce changes made to the Gatekeeper's **API** (Section 2.1.1.1), and then present two new modules: the **Licence Management** (Section 2.1.1.2) and the **KPI Management** (Section 2.1.1.3). Next, changes to the **BSS** (Section 2.1.1.5) and the **GUI** (Section 2.1.1.4) modules are presented.

For further details about the Gatekeeper please refer to [9].

2.1.1.1 Gatekeeper API

The current section describes the delta, in terms of integration, from the previous SONATA version. Details are available in [9].

HTTPS only access

From this version on, the API will be accessible only through HTTPS. We therefore need to define the needed certificate(s) and test if access through the common HTTP is not possible (or is restricted, for example, to the root of the API.

URL scoping and API versioning

The second version of the SONATA service platform brought URL scoping, i.e., all URLs related to the API will be prefixed with `.../api`

Furthermore, we have implemented a URL versioning schema, which means that the above stated URL will give access to a default version of the API, currently version 2 (`v2`). An explicit version can also be accessed, thus guaranteeing backwards compatibility, like in `.../api/v2`

These different kinds of access have to be tested, specially to guarantee that no un-authorised access is authorised.

New modules available

Version 2 of the service platform makes available two new modules. These are directly or indirectly made available to the remaining SONATA ecosystem. Moreover a third one is still under development. These modules are:

- **Licence Management:** accepted services and functions are either 'Public', which can be re-used by any developer in composing new services and functions and by any customer, or 'Private', for which the user (developer or customer) have to 'buy' a licence in order to use the service or function (see Section 2.1.1.2);
- **KPIs Management:** we are starting to collect a set of indicators that will demonstrate the quality of our solution (see Section 2.1.1.3);

- **User Management:** ince we are 'opening' the platform to external users, we need to know them and give each one of them the right level of access to the platform (module still under development, not present in this release).

Access to these new modules will be done, respectively, with the following endpoints:

- `.../api/v2/users`
- `.../api/v2/licences`
- `.../api/v2/kpis`

Other auxiliary endpoints will also be available, but still a work in progress at the time of this writing.

Integration tests for these endpoints are being designed and implemented into the whole integration flow.

2.1.1.2 Licence management module

Besides authorising users to use a certain service and/or function, we need to distinguish between those services and functions that are made available for everyone to use (we call these **Public** services and functions) and those that require the developer to authorise their usage, either to be part of other service or function, or to be used by a customer (we call these **Private** services and functions).

A service or function which does not specity anything about its licensing scheme is considered to be `public`. When a service or function is marked as `private` in its definition, there has to be a URL, provided by the developer of that service or function, that will validate (or not) its usage, whenever another developer wants to reuse the service or functions or a customer wants to buy the service (we are assuming only services are sold, but not functions).

2.1.1.3 KPIs management module

The KPIs Management is a sub-module of the SONATA Gatekeeper. This component is designed to manage the key metrics and the key performance indicators (KPIs) of the gatekeeper. Examples of these indicator are:

- Total API calls received;
- Total or Active users registered;
- Total or Active customers registered;
- Total Developers registered;
- Total Services instantiated;
- Mean Service Instantiation time.

An extensive list of these indicators is given in Deliverable D4.2 [9]

2.1.1.4 Graphical user interface module

Gatekeeper GUI module is an API management and visualisation tool that enables SONATA developers to manage their services throughout their whole lifecycle and enables the service platform administrator to provision and monitor platform resources easily and securely.

In this perspective, the Gatekeeper GUI has been designed, developed and implemented to cover the needs of the two aforementioned groups, namely service developers and platform administrators, in supporting the process of DevOps in SONATA.

This section describes the Graphical User Interface (GUI) functionality enhancements achieved from the previous open source release (v1). In particular, the following functionality has been added:

- This second release includes the addition of a new VIM. From the GK GUI point of view, this required additions and modifications to existing (release 1) source code to support a specific set of values that would allow a new VIM to be instantiated in the SONATA service platform in an automated way;
- In this second release of SONATA source code, the css files have been polished to extend user interface with a mobile (Android) app.

2.1.1.5 BSS module

This release includes new features and enhancements in the BSS. The enhancements are the use of HTTPS based APIs and header links pagination; the new features includes both an user authentication/authorisation management, and a service license management.

2.1.2 MANO plugins

The core of SONATA's SP is the highly flexible management and orchestration (MANO) framework. The framework consists out of a set of loosely coupled components, called MANO plugins. Each of these plugins implements a limited, well-defined part of the overall management and orchestration functionality.

In our design, all components are connected to an asynchronous message broker used for inter-component communication. This makes the implementation and integration of new orchestration plugins much simpler than it is in architectures with classical plugin APIs. The only requirement a MANO plugin has to fulfil is the ability to communicate with the messaging system, in our case RabbitMQ. We do not enforce a programming language for implementing plugins. Similarly, we do not prescribe how the plugins are executed.

To keep track of the connected MANO plugins, the *plugin manager* component is used, which is a plugin-like component that communicates with other plugins over the broker, offers a management interface, and interacts with the configuration interface of the message broker.

In the following subsections, the main achievements in MANO plugins (Service Lifecycle Manager, Specific Manager Registry, son-mano-base) integrated in the second release of SONATA open source are discussed.

2.1.2.1 Service lifecycle manager

The Service Lifecycle Manager (SLM) is the plugin that handles all decisions concerning the lifecycle of the services. Through the message broker, it receives requests from the Gatekeeper (GK) and it translates this request into instructions for the other MANO framework microservices and the

Infrastructure Adaptor (IA). The following functionality was added to the SLM, since the last release:

- The SLM is converted into a task manager. Each incoming request is translated in a set of tasks that need to be completed to satisfy the request. This will allow Service Specific Managers (SSM) to customize the behavior of the SLM when it handles a request, as SSMs will be allowed to make changes to the list of tasks;
- The owner of the SP can decide to increase the reliability of the SLM. The SLM is constructed in such a way, that when multiple instances of the SLM are running, they are balancing the load in a distributed way and are sharing their state. Based on their own uuid, and the correlation id of the requests, SLMs decide individually whether they should handle an incoming request or not. During the process of handling a request, the SLM that is handling the request shares the status of this process, by informing the other SLMs which tasks are completed and what data is needed to start the next task. This allows another SLM to continue the process of handling the request when the initial SLM fails;
- The MANO framework supports the new "service deployment" workflow, where MANO framework requests the deployment of individual VNFs to the IA and instructs the IA how to chain them together when they are deployed.

2.1.2.2 Son-mano-base

For the python implemented MANO plugins, we provide the module son-mano-base, which abstracts the messaging and the plugin layer, with the following improvements:

- A thread that consumes a topic on which a response is expected is now terminated after the response is received, instead of keeping this thread alive for the remainder of the lifecycle of the MANO framework.

2.1.2.3 Specific manager registry

Specific Manager Registry (SMR) is a MANO framework plugin that is responsible for FSM/SSMs lifecycle management including onboarding, instantiation, registration, updating, and termination. It interacts with other MANO framework plugins through the message broker, e.g., to obtain SSM onboarding request from SLM. The SMR provides the following functions:

- SSM/FSM on-boarding: It downloads SSM/FSM images from the docker registry that stores the SSM/FSM images;
- SSM/FSM instantiation: It starts the SSM/FSM docker containers;
- SSM/FSM registration: It is responsible for SSM/FSMs registration by storing a record of them;
- SSM/FSM updating and termination: This function replaces an SSM/FSM with a new version and terminates the old one.

2.1.3 Catalogues and repositories

This release includes few enhancement features for the SP Catalogues including Meta-data and data levels and SONATA package storage. The addition of meta-data to the stored descriptors in the Catalogues enables efficient handling of security (Signatures, etc.) and licensing (License type, etc.) related information about the descriptor. The SONATA package storage allows easy storage and retrieval of complete SONATA packages, as one entity, and avoid the burden of compiling the package from descriptors as was done in previous release.

2.1.4 Monitoring framework

The SONATA Monitoring Framework is responsible for processing monitoring data collected from the respective Virtual Network Functions comprising Network Services, through monitoring probes. Moreover, the Monitoring Framework provides the ability to the developer to activate predefined metrics (RAM and CPU usage, hard disk usage, etc.) or develop and implement his own monitoring metric in order to capture service-specific behaviour. In order to close the cycle between development and operations (DevOps), the Monitoring Framework is able to send alerts to the developer through subscription to the dedicated SONATA message queue, while it also allows for asynchronous-type notifications, e.g. by email and/or SMS.

This section describes the Monitoring Framework functionality enhancements achieved from the previous open source release (v1). In particular, with respect to the monitoring probes, the following functionality has been added or corrected:

- In the first release of SONATA source code, the data collected from monitoring probes and pushed to the PushGateway were not including a timestamp. Thus, Monitoring Manager (Prometheus server) was gathering the data (through an API GET method) and added a timestamp that could not be the correct one. In this second release, a timestamp has been added on the data collected from the probes and thus it is possible for the Monitoring Manager to identify whether the data are new or stale;
- As it has been highlighted from the beginning of the project, the inclusion of probes related to virtual networks data collection was essential for the project, as SONATA is targeting SDN technology, in particular related to VNF service deployment. In this respect, the second release includes an integrated library allowing for the collection of network data to the SDN controller taking advantage of Openflow standardised protocol;
- Moreover, supporting scalability is one of the cornerstones of the project. To support this requirement on monitoring level, an extra mechanism has been implemented on monitoring probes, called "send on change" that allows for defining a threshold indicating whether new data will be sent to the PushGateway or not, thus reducing the exchange of information in the network and reducing the requirements for storing duplicate data.

Also, the following feature extensions have been achieved during this period with respect to the Monitoring Manager:

- In the second release, MySQL database on Monitoring Manager has been replaced by PostgresDB. This replacement has taken place since other partners have been using PostgreSQL, so a homogenisation has been achieved (in terms of maintenance, etc.);
- Call `/functions/service/{srvID}` has been introduced to return details regarding the list of functions composing a network service. This will simplify the process of managing probes and other monitoring parameters in the Network Services;

- Call `/metrics/function/{funcID}` has been introduced to return the metrics collected per VNF, so it is simpler to modify user settings;
- Call `/users` has been introduced to retrieve the list of authorised users of the SONATA service platform.

2.1.5 Infrastructure abstraction

The Infrastructure Abstraction allows the SONATA MANO framework to interact with the underlying virtual infrastructure in a vendor-agnostic fashion. Through the SONATA service platform Message Bus, it offers a set of abstract API to control the deployment and the life-cycle of VNFs and Services instances. These are internally mapped to vendor-specific procedures using a plug-in system which allows attaching and detaching adaptor (called Wrappers) for different vendors of Virtual Infrastructure Managers. The latest IA model for data, wrappers and VIMs has been described in Deliverable D4.2 [9].

The second release of the Infrastructure Abstraction (IA) adds several crucial functionalities and provides some internal refactoring aimed at optimising performance and increasing reliability:

- First release of the IA leveraged a set of python-based clients to allow its OpenStack VIM wrapper to interact with the OpenStack RESTful servers. This was bug-prone and difficult to debug and extend. The new release features an all-java OpenStack RESTful client;
- the OpenVSwitch VIM Wrapper, which allows installing SFC chains on Neutron-based OpenStack VIMs, and VTN WIM Wrapper, able to configure WIMs between NFVi-PoPs to connect users to service deployment, have occurred a major bug fix, especially to provide an error tracking system when SFC or WIM configuration cannot be enforced;
- Minor bug fix for communication between VIM adaptor and WIM adaptor, to avoid messages repetition and loss due to pub/sub queue re-use;
- First release considers the network service as the basic unit of deployment. As such, all VNFs of network service had to be deployed only on single NFVi-PoP. The new release allows the MANO framework to split the deployment on a per-function routine. Therefore a network service could be deployed on more than one NFVi-PoP, enhancing the control functionalities of the MANO framework, especially in terms of life-cycle management and placement routines;
- The second release of the IA offers a new set of API to distribute and provision VNFs images on the underlying infrastructure.

2.1.6 Son-install

The module 'son-install' is a new addition to the SONATA ecosystem. It is in charge of the automatic deployment of the SONATA service platform. It contains a full set of Ansible playbooks for Create, Manage, Upgrade and Destroy (CMUD) SP resources. Since this module has not been documented in Deliverable D4.2 [9], we provide here a brief description of its structure and features.

The main playbook, 'son-cmud.yml', allows a matrix combination at run time:

- **Multi-environment** - specify for which kind of environment it will be deployed, namely **qual**ification, **demo**nstration or the **sp** as a standalone platform (default: 'qual');
- **Multi-PoP** - specify the point-of-presence (or 'placement') where the resources will be instantiated (default: 'ncsrd');

- **Multi-VIM** - specify the cloud infrastructure technology (default: 'openstack');
- **Operations** - specify what type of operation will be executed, namely **install**ation, **manage**ment, **upgrade**, **destroy**, **test** (default: 'install');
- **Services** - specify over which set of services the operation will accur, namely: **gtk**, **repo**, **mano**, **ifta** (default: 'all');
- **Action** - only used for 'manage' operations over a service, example: **start**, **stop**, **status**;
- **Multi-distro** - specify the operating system for a particular VM - supported OS: **Ubuntu 14.04**, **Ubuntu 16.04**, **CentOS 7**;

Being a command line interface (CLI), the playbooks are invoked from a Linux 'bash' with Ansible installed.

Here is the flow to deploy an environment (eg, a Service Platform, a Qualification platform or Demonstration platform) from the scratch:

The latest version of 'son-install', included in the second release of SONATA, also enables the following new features:

- Deployment of infrastructure resources to build a designated platform - eg, the Qualification environment;
- Test the deployed infrastructure;
- A wider set of Ansible roles, eg, Nginx, PostgreSQL, Jenkins, JDK.

2.2 SONATA SDK modules

2.2.1 Son-cli : SONATA SDK toolset

`son-cli` SDK component is a set of command line tools that are meant to assist the SONATA service developers on their tasks. It includes a series of tools to cover critical points of service development within SONATA SDK. This section focuses on the new features, improvements and implementation details of `son-cli` tools for SONATA SDK v2 release. Overall information on the component can be found in deliverables D3.2 [7] and D2.2 [4].

2.2.1.1 Son-access

The son-access tool in son-cli enables a developer to get authenticated access to SONATA SP for storage and retrieval of service packages. Its main new feature is secure access to SONATA SP Catalogues for storage and retrieval of packages and descriptors.

2.2.1.2 Son-workspace

The `son-workspace` tool handles the creation and management of a development workspace and environment and the creation of projects. A workspace contains a user-specific configuration, such as user credentials and addresses to private catalogues, which can be used for the creation and maintenance of multiple projects. The new functionalities of SDK tools, namely the authentication in the service platform, led to a different workspace configuration schema. As such, the `son-workspace` configuration is not backwards compatible with the previous release.

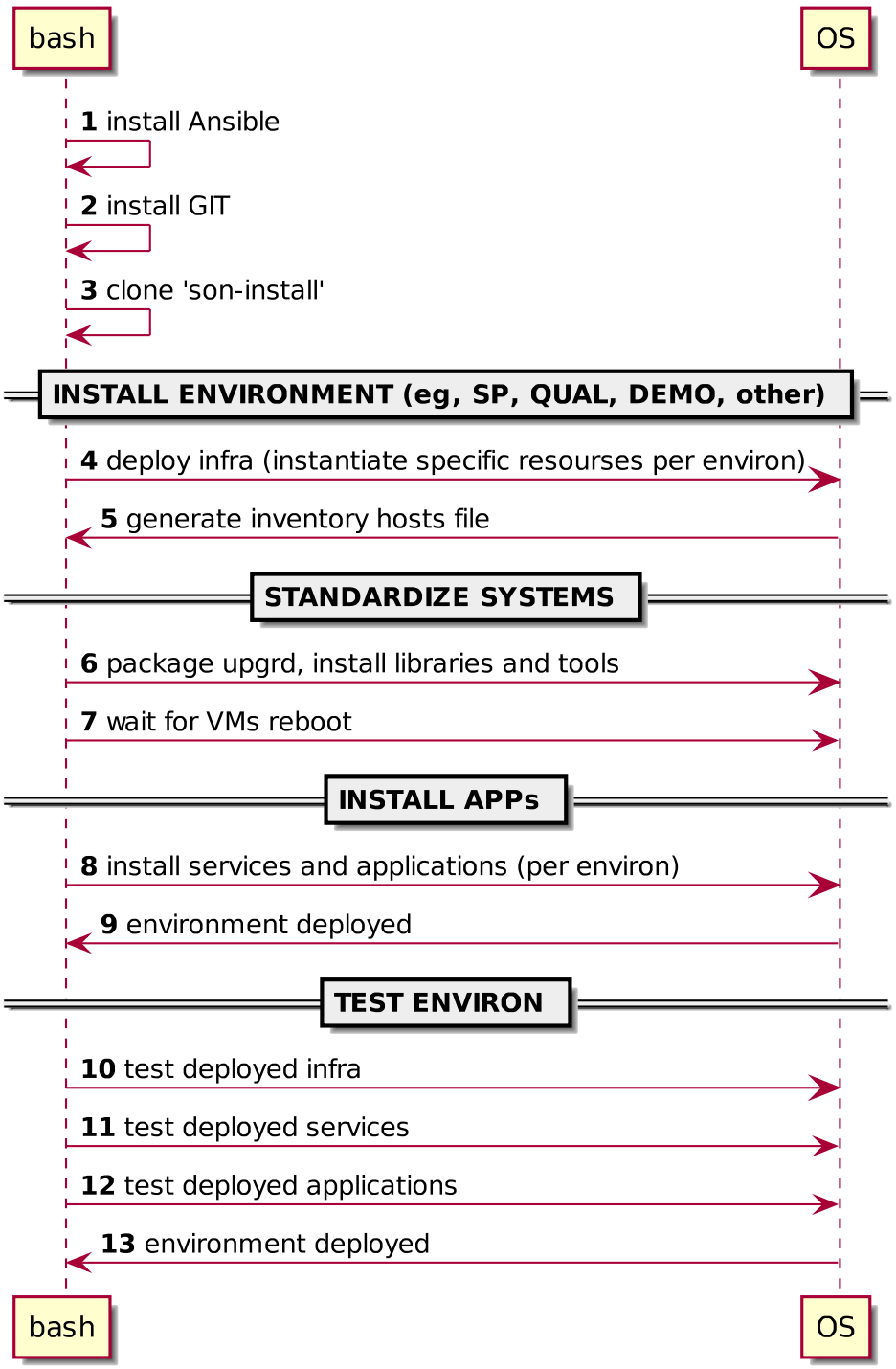

Figure 2.2: Son-install Create, Manage, Upgrade, Destroy and Test SONATA environments

2.2.1.3 Son-package

The `son-package` tool has the main role of packaging a project, making it ready and available for instantiation in the service platform. Previously, the packaging process involved an interaction with the SDK `son-catalogue`, however as this component was removed this interaction will be performed with the service platform, using the `son-access` tool, in order to retrieve external dependencies. Moreover, the validation of project and its components that were integrated inside `son-package` will be performed by the new `son-validate` tool.

The workflow of `son-package` can be as illustrated as in Figure 2.3.

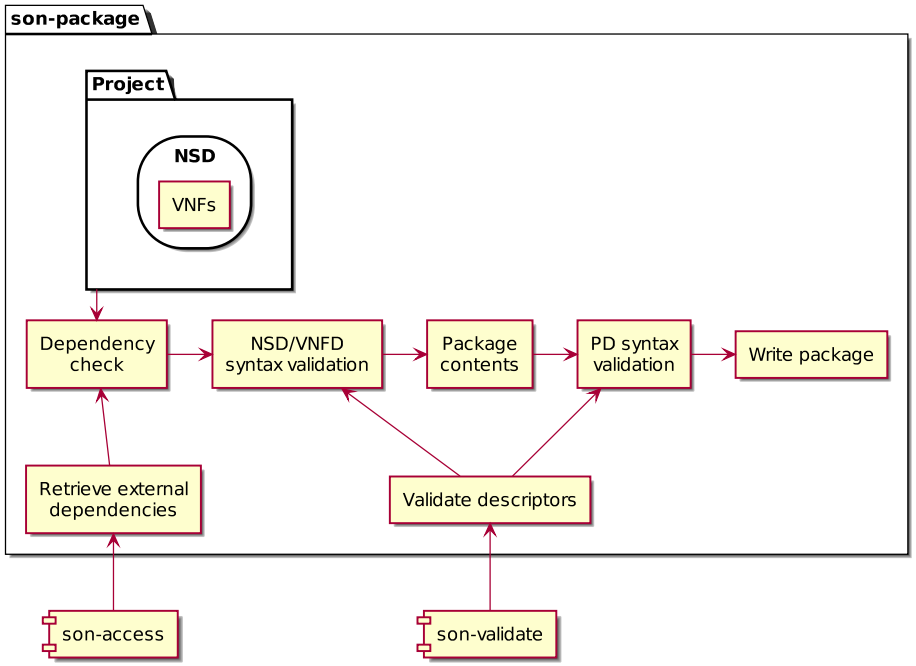

Figure 2.3: Son-package tool

Interaction with `son-access`

To be able to solve external dependencies, `son-package` must interact with `son-access` which will retrieve service and function descriptors from the service platform catalogues. This does not affect the logic and workflow of building a service package, but a modification of the inner `son-package` client to interact with `son-access`. To be noted that a cache system for the external dependencies will still exist and work according to the same logic of the previous implementation.

Interaction with `son-validate`

The validation process of services, functions and the final package itself is also outsourced to the `son-validate` tool. As `son-validate` is being developed specifically for this purpose, but also providing additional validation features other than syntax (service integrity and network topology), it is more proficient to use this tool to perform the validations.

2.2.1.4 Son-validate

The son-validate is a new CLI tool for the SDK with the purpose of aiding the development of services and functions. This tool was mainly developed to support the validation of SDK projects, however it is also designed to be utilised outside the SDK scope. Individual service and function descriptors can be validated independently, without requiring a developer workspace, which makes it adequate to be used by the SONATA's service platform. The son-validate addresses the following validation scopes: Syntax, Integrity and Network Topology.

- **Syntax:** the service descriptor and corresponding function descriptors are syntactically validated against the schema templates, available at the `son-schema` repository.
- **Integrity:** the validation of integrity verifies the overall structure of descriptors by inspecting references and identifiers both within and outside individual descriptors. Because service and function descriptors have a different structure, it is differentiated in two components, the Service Integrity and the Function Integrity.
  - **Service Integrity:** service descriptors typically contain references to multiple VNFs, which are identified by a composition of the vendor, name and version of the VNF. The integrity validation ensures that the references are valid by checking the existence of the VNFs. Integrity validation also verifies the connection points of the service. This comprises the virtual interfaces of the service itself and the interfaces linked to the referenced VNFs. All connection points referenced in the virtual links of the service must be defined, whether in the service descriptor or in the its VNF descriptors.
  - **Function Integrity:** similarly to service descriptors, VNFs may also contain multiple sub-components, namely the Virtual Deployment Units (VDUs). As a result, the integrity validation of a VNF follows a similar procedure of a service integrity validation, with the difference of VDUs being defined inside the VNF descriptor itself. Again, all the connection points used in virtual links must exist and must belong to the VNF or its VDUs.
- **Network topology:** the `son-validate` provides a set of mechanisms to validate and aid the development of the network connectivity logic. Typically, a service contains several inter-connected VNFs and each VNF may also contain several inter-connected VDUs. The connection topology between VNFs and VDUs (within VNFs) must be analysed to ensure a correct connectivity topology.

The `son-validate` tool comprises the following validation mechanisms. Figure 2.4 shows a service example used to better illustrate validation issues.

- **Unlinked VNFs, VDUs and connection points:** unconnected VNFs, VDUs and unreferenced connection points will trigger alerts to inform the developer of an incomplete service definition. For instance, *VNF#5* would trigger a message to inform that it is not being used.
- **Network loops/cycles:** the existence of cycles in the network graph of the service may not be intentional, particularly in the case of self loops. For instance, *VNF#1* contains a self linking loop, which was probably not intended. Another example is the connection between *vdu#1* and *vdu#3* which may not be deliberate. The `son-validate` tool analyses the network graph and returns a list of existing cycles to help the developer in the topology design. In this example, `son-validate` would return the cycles:

– [*VNF#1*, *VNF#1*]
– [*vdu#1*, *vdu#2*, *vdu#3*, *vdu#1*]

- **Node bottlenecks:** warnings about possible network congestions, associated with nodes, are provided. Taking into account the bandwidth specified for the interfaces, weights are assigned to the edges of the network graph in order to assess possible bottlenecks in the path. As specified in the example, the inter-connection between *vdu#2* and *vdu#3* represents a significant bandwidth loss when compared with the remaining links along the path.

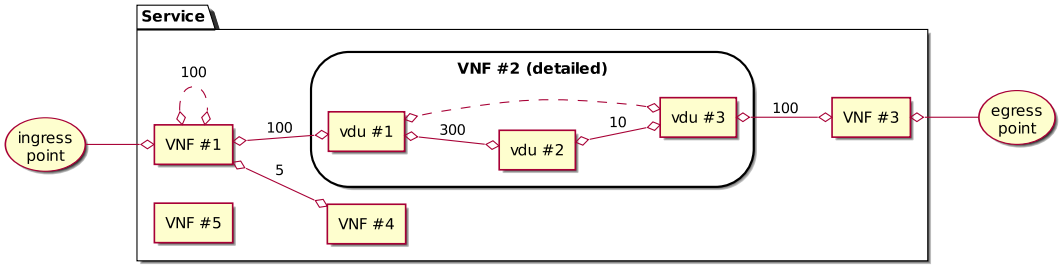

Figure 2.4: Example of a service network topology

Functionalities and requirements

The `son-validate` tool offers different validation levels, the syntax, integrity and topology. However, integrity validation must comprise a syntax validation and in turn topology must comprise an integrity validation.

As previously mentioned, `son-validate` can be used inside the SDK, under the developer environment, and as an external tool. In the first case, the workspace should be specified in order to read the environment configuration. Moreover, if an SDK project should be validated a workspace must be specified. If `son-validate` is being used outside the SDK it is possible to validate a service or individual functions. The validation of a service comprises the validation of the service itself and, if at least integrity is specified, its referenced VNFs. On the other hand, the validation of functions only verifies each function individually.

2.2.1.5 Son-profile

`son-profile` is a new tool in SONATA's SDK to allow network service developers to automatically profile and test network services before they are deployed to production. The basic idea of `son-profile` is to deploy network services on SONATA's emulation platform and do load testing under different resource constraints. During these tests a variety of metrics can be monitored which allows service developers to find bugs, investigate problems or detect bottlenecks in their services. The main purpose of `son-profile` is to automate big parts of this workflow to support network service developers as much as possible. Detailed design and the general concept of `son-profile` was already introduced in Section 3.5 of Deliverable D3.2 [7].

The tool's full functionality is still under development, `son-profile`'s early component design includes the possibility to automate following functions:

- The execution of commands in the Test-VNFs to generate load, as described in a test-specific *Profile Experiment Descriptor* (PED) file;

- Query monitored KPI values during the load test, as described in a test-specific *Monitoring Service Descriptor* (MSD) file.

This requires that the service or VNF is pre-deployed on `son-emu` before the profiling test is started. The output of this profiling test characterizes the performance of the service or VNF under test.

2.2.1.6 Son-monitor

The functionality implemented in `son-monitor` enhances the SONATA SDK to easily export and process monitored metrics from deployed services. The architecture to handle the monitoring in the SDK is illustrated in Figure 2.5. The usage is focused on gathering information from services deployed in the SONATA SDK emulator, 'i.e.' from the `son-emu` module.

Compared to the previous open source release (v1), the modular approach has been further extended:

- **Resource monitoring and updating**: compute, storage and network related metrics gathered by cAdvisor in `son-emu`. In addition, `son-monitor` has the ability to modify the resource allocation (currently only CPU share) of a deployed VNF in the emulator, using the `son-emu` REST API;
- **Network metric exporters in son-emu**: Additional network metrics are gathered and exported by extra monitoring functionality in `son-emu`. These metrics are the packet and byte counters of the (virtual) network interfaces that are installed in the emulated service. Next to the interface counters, also specific flow counters can be installed to allow a more finer-grained network traffic monitoring;
- **VNF resource starvation monitor**: When testing different VNFs in a deployed service, resource depletion can occur on the host where `son-emu` is deployed. A dedicated monitoring tool can be started, that exports a metric that indicates if the VNF resources are sufficiently available or not. This can be used by `son-monitor` to asses if a deployed VNF is suffering from resource starvation or not;
- **Prometheus Gateway and Database**: A Prometheus Gateway is deployed in `son-emu`. This database is started when son-monitor is deployed, and it gathers the metrics exported by `son-emu`. This framework is compatible with the Prometheus instance running in the SP, and also metrics exported by the SP could be gathered;
- **Visualization using Grafana**: To provide a more user-friendly way of monitoring and debugging, monitored metrics can be visualized using Grafana [11]. This visualizes the queried metrics from the Prometheus database through a web-based GUI. A Monitoring Service Descriptor (MSD) file describes which metrics need to be visualized. The `son-monitor` control code parses this MSD file and translates this to a Grafana Dashboard, accessible via a web-GUI;
- **Test-VNF development**: When a service is deployed in the SONATA SDK emulator, several Test-VNFs can be automatically attached to the service's end-points. From these Test-VNFs, different traffic generating and analysis tools can be started (currently iperf is used, additional tools can be easily installed). By deploying them as a separate VNF, their resource usage can be isolated from the actual service under test.

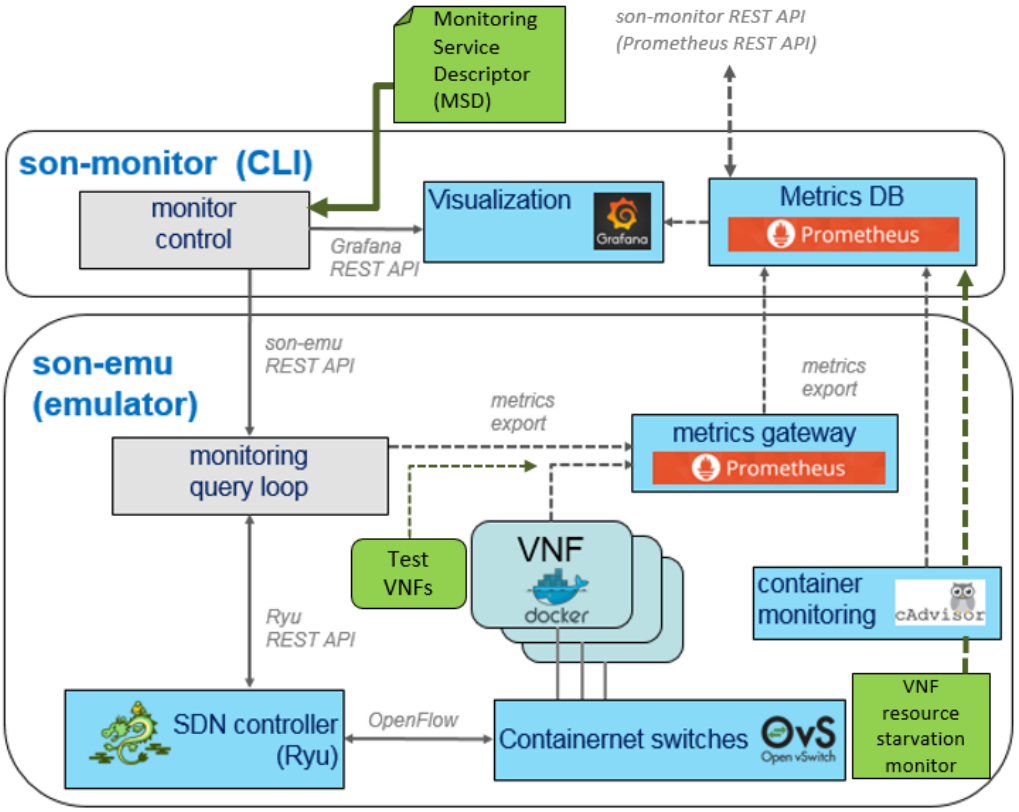

Figure 2.5: Different functional blocks in the SDK monitoring framework

The described functionality will be used by the `son-profile` tool, developed to automate the deployemnt of a VNF or service, put it under a specified load and measure dedicated KPI's. This mechanism can then characterize the performance of a VNF or service in an automated way for varying input load or resource limitations. The `son-monitor` command is situated as a part of `son-cli` repository, and its current release is further detailed in Deliverable D3.2 (section 3.6) [7].

2.2.2 Son-emu : SONATA SDK Emulator

Our emulation platform called `son-emu` allows network service developers to locally prototype and test complete network services in realistic end-to-end multi-PoP scenarios. This allows a developer to fully validate and debug SONATA service packages, before they are are pushed to the the SONATA service platform. The new and planned functionality for the second year of SONATA is described in Section 3.4 of Deliverable D3.2 [7].

As illustrated in Figure 2.6, the emulator deploys a pre-defined topology, a virtual network with multiple PoPs. Secondly, a service consisting of different interconnected VNFs is deployed on these PoPs. A dummy-gatekeeper receives and deploys the SONATA package that describes the service. After the developer has verified the service, the SONATA service package can be sent to the SONATA service platform via the SP GateKeeper.

Regarding the version 2 release of the SDK we can highlight these improvements and new features available for `son-emu`:

- **PoP resource isolation models:** The emulator attaches VNFs to virtual PoPs, organized in a pre-loaded network topology. This mechanism allows developers to emulate resource isolation between PoPs and to simulate cloud-like overbooking of individual PoPs. The VNFs deployed in the emulator receive a share of the CPU as defined by the resource model. There are two resource models available which are described and evaluated in [14];
- **VNFD-based container resource limit support:** Resource configurations, such as number of CPU cores or memory limits that can be defined within a VNFD, are now applied to the VNF containers running in the emulator when a service is deployed using our dummy gatekeeper. This improves the compatibility of our dummy gatekeeper to the SONATA descriptors. This feature is also important for the foreseen integration between `son-emu` and the planned profiling tool, called `son-profile`;
- **Round-robin placement in dummy gatekeeper:** The dummy gatekeeper now distributes the VNFs of a deployed service evenly across all available PoPs. A developer can easily add customized placement algorithms;
- **Isolated E-LAN networks are supported and deployable from the dummy-gatekeeper:** VNFs can now be interconnected using two types of links, as described in the service descriptor: E-Line and E-LAN;
- **Xterm window to access a VNF:** To assist a developer, a terminal window can be opened using a single command that attaches to the VNF and allows direct manipulation inside the VNF.

2.2.2.1 REST API updates

The `son-emu` REST API is the main interface to control the deployed VNFs in the emulator. It can be used to manage/instantiate/shutdown VNFs or services. Additionally, also the chaining of

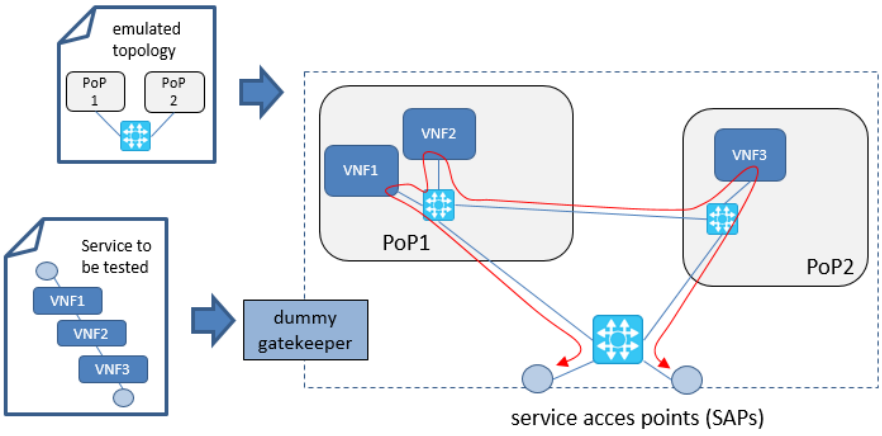

Figure 2.6: SONATA emulation tool general deployment

the VNFs in the service can be installed and specific (compute/network/storage) metrics can be monitored. The updates for the second release include:

- Gracefully shut down services through the SONATA dummy gatekeeper interface;
- REST API extension to allocate/limit VNF CPU resources;
- The son-emu REST API returns the names of connected interfaces of a deployed VNF; This makes it possible to retrieve the correct interface name that should be monitored, with e.g. Wireshark, when trying to monitor the network traffic flowing through a VNF's emulated interface on the host.

3 Integration tests

In Deliverable D5.2 [10] we documented an extensive list of integration tests designed and developed to ensure the stability of our software base throughout the whole Continuous Integration/Continuous Delivery cycle. Since new modules have been introduced in the SONATA ecosystem, new integration tests have to be designed and developed. Such tests will ensure that new contribution and features development for these modules will not break the service platform stability and will respect the designed interfaces and protocols. In this section we document these new integration tests.

3.1 CLI - Development of a service

3.1.1 Test description

This set of integration tests aim to translate a typical SDK workflow during the development of a service, starting by an empty workspace project up to the construction and upload of a service package to the service platform.

The CLI is composed of by multiple individual tools, briefly described as follows:

- `son-workspace` is responsible for the initialisation of the working environment for development of projects. This tool is also used to create and instantiate projects;
- `son-package` gathers all components from a project and packages them in a zipped container in order to be pushed to the service platform. During the packaging process, external project components are retrieved using `son-access` and all components are syntactically validated using `son-validate`;
- `son-validate` is a descriptor validation tools, designed to handle the validation of SDK projects, packages, service descriptors and function descriptors;
- `son-access` provides the interaction with the service platform. It handles user authentication and enables the exchange of data between the SDK and the SP's Gatekeeper through pull and push requests.

To be able to test the integration between CLI tools, a test story was defined following two major sets of tests:

- Test CLI tools and test the integration with the Emulator Gatekeeper (`son-emu`);
- Test CLI tools and test the integration with the service platform Gatekeeper (`son-gkeeper`).

In each set of tests, the following procedure is followed:

3.1.1.1 Workspace creation and project creation and development

1. Create a new workspace: a workspace environment is created to support the development of projects;
2. Configure workspace: the workspace is configured with the developer credentials, son-schema repository address, son-catalogue repository address and several customized parameters, such as level of debugging, name of workspace, etc.;
3. Create a new project: a project is created. A service descriptor (NSD) is added to the project, as well as all its required function descriptors (VNFDs). This results in a self-contained project, i.e. it does not depend on external components.

The sequence diagram for the above steps is shown in Figure 3.1

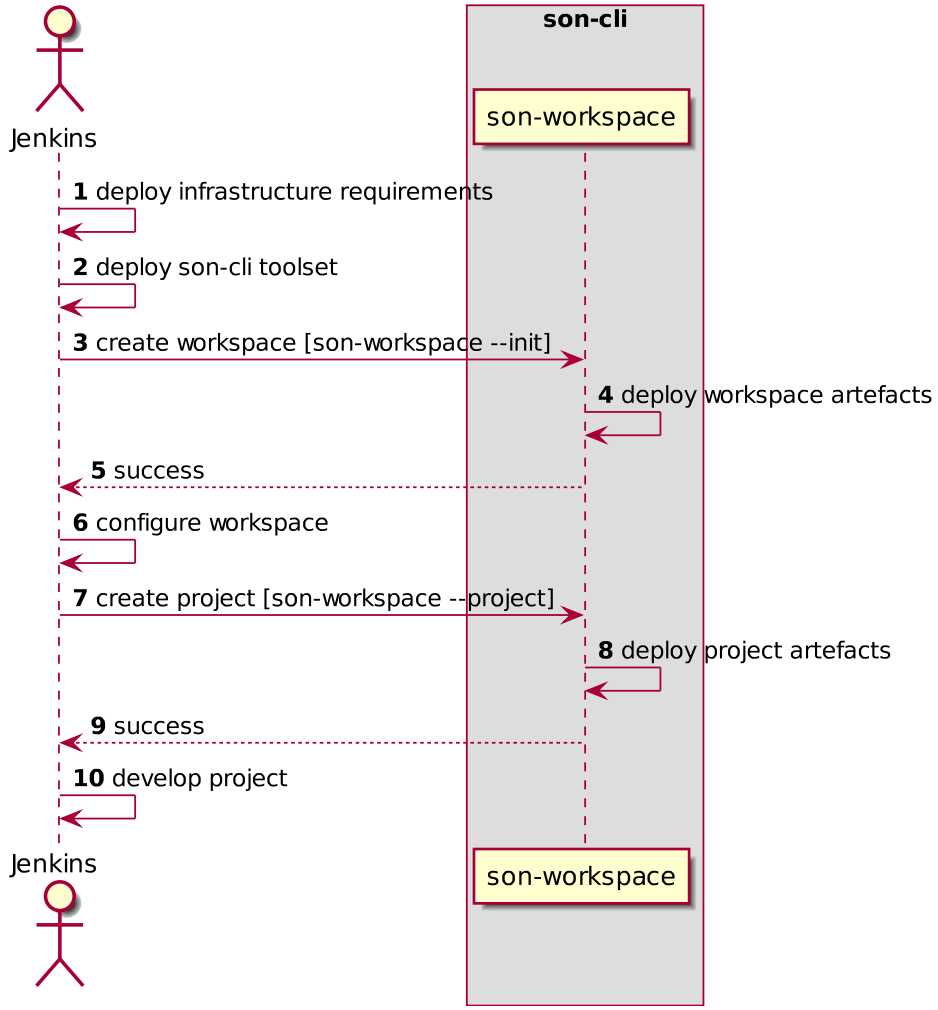

Figure 3.1: Integration test - Workspace configuration and project creation

3.1.1.2 Project packaging, validation and upload to SP

1. Package the developed project using `son-package`. `son-package` will inherently invoke `son-access` to solve dependencies and `son-validate` to validate the syntax of descriptors of the project;

2. Perform a full package validation using `son-validate`. The validation procedure will trigger the validation of syntax, integrity and topology of the entry service descriptor and all its referenced function descriptors;

3. Upload the generated package using `son-access`.

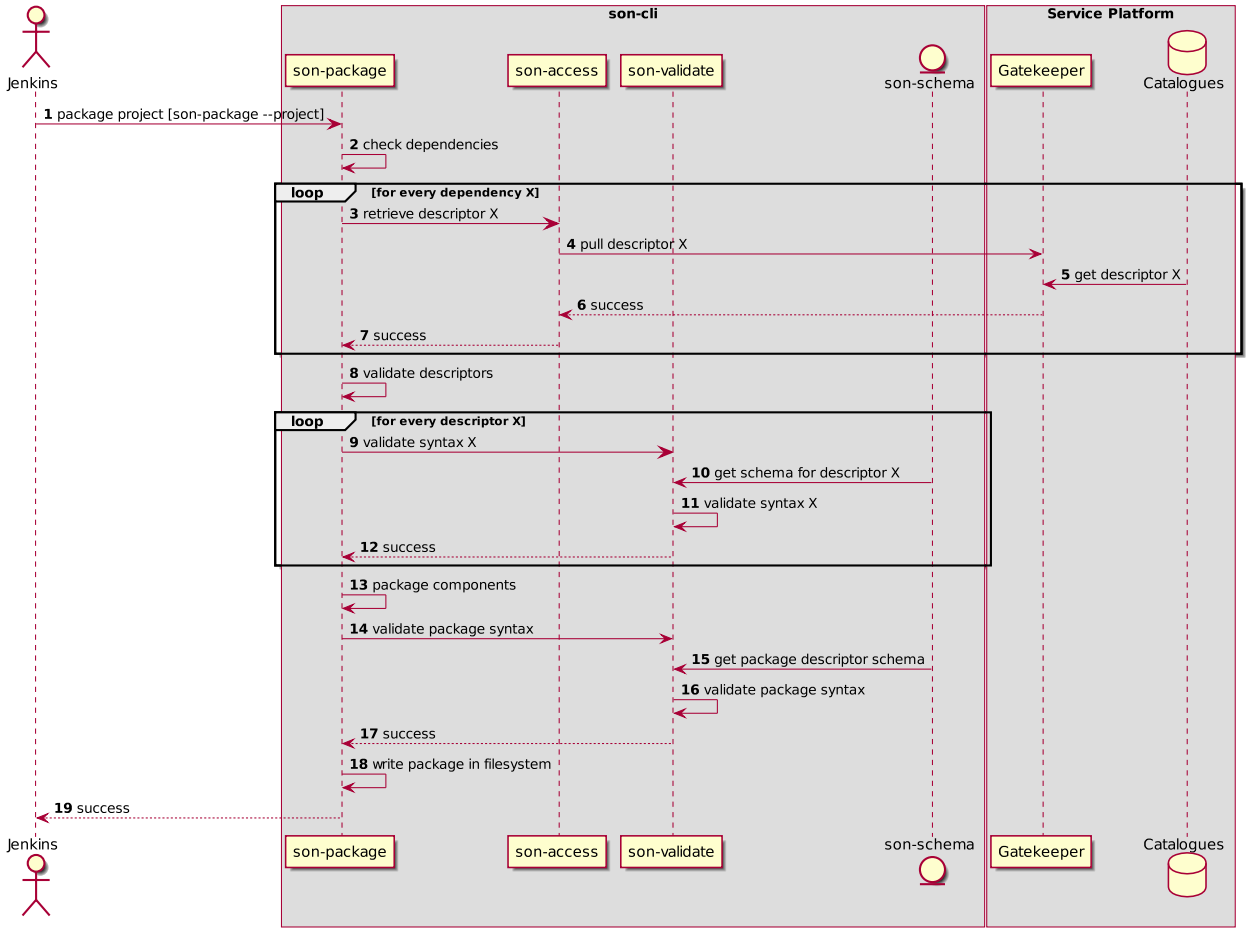

Figure 3.2: Integration test - Service packaging

3.1.2 Tests requirements

List of modules affected and corresponding repository:

- **Son-workspace**: son-cli;
- **Son-package**: son-cli;
- **Son-validate**: son-cli;
- **Son-access**: son-cli;
- **Son-schema**: son-schema;
- **Son-gtkapi**: son-gkeeper;
- **SP Catalogue**: son-catalogue-repos.

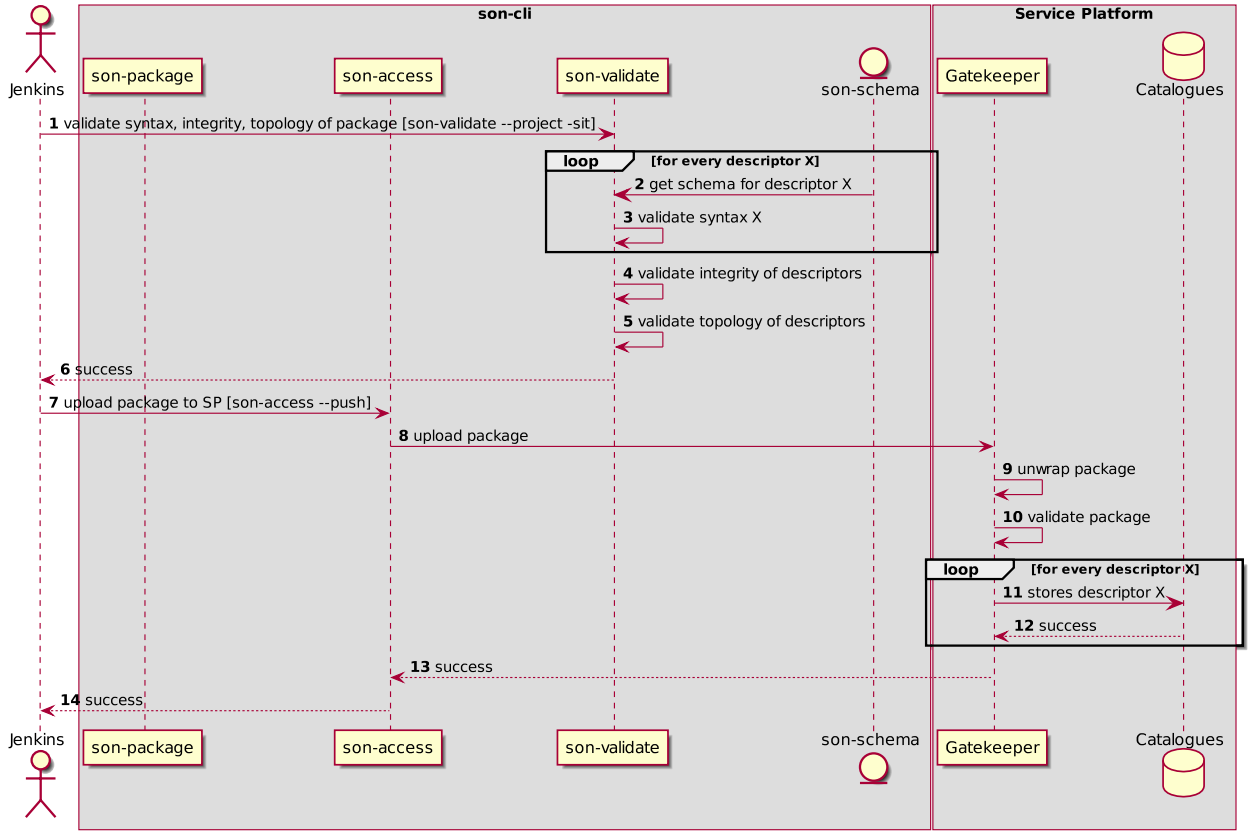

Figure 3.3: Integration test - Service package validation and upload

3.1.3 Test triggers

- Jenkins;
- Scheduled on need basis.

3.1.4 Actions after the test

- Send an email to *lead developers*;
- Start other integration tests related to service deployment.

3.2 Service/Function lifecycle management, specific manager registry and service/function specific management

3.2.1 Test description

This test targets the integration of the MANO framework plugins including the Service Lifecycle Manager (SLM), the Function Lifecycle Manager (FLM), the Specific Manager Registry (SMR), the Service Specific Manager (SSM) and the Function Specific Manager (FSM) for the purpose of FSM/SSM onboarding and instantiation. To start the test, a message is injected from the Jenkins Job into the service platform message bus on the service.instance.create topic. This message triggers the SLM to start the 'service deploy' workflow. The descriptors included in the message should describe a service and a VNF that require the use of an SSM and an FSM. This will trigger the SLM to instruct the SMR to onboard the needed SSMs, by sending a message on the specific.manager.registry.ssm.on-board topic. The request message is sent to SMR along with the service descriptor (NSD) of the service. This message triggers the SMR to obtain the SSM registry URI from the NSD and to on-board the SSM. The result of this action will be sent back to SLM.

In the next step, the SLM sends an SSM instantiation request to the SMR using the message bus. This triggers the instantiation function of the SMR which starts the SSM as a docker container. Once the SSM is up and running, it sends a registration message to the SMR. The SMR receives the message, registers the SSM, and sends the result to the SSM. The SMR also notifies the SLM about the result of the SSM instantiation.

After this, the SLM will perform some tasks that are listed in the 'service deploy' workflow, which are not relevant to this task. After these processes are completed, the SLM will instruct the FLM to deploy the VNFs of the service. Once the FLM receives such a request, which includes the VNF descriptor (VNFD), a similar process as the one described above takes place. The FLM requests the SMR to onboard, and then instantiate the FSM associated with the VNF. The entire test process is depicted in Figure 3.4

3.2.2 Infrastructure requirements

This test makes use of the following repository:

- Son-mano-framework
- RabbitMQ
- MongoDB

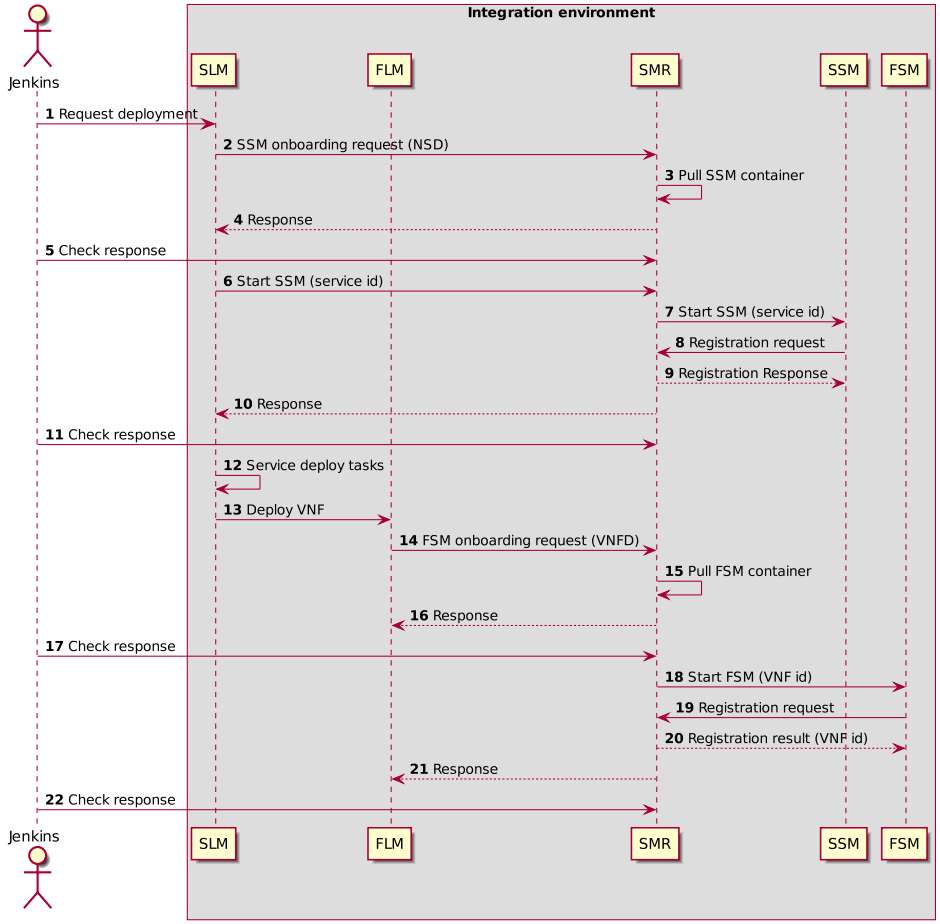

Figure 3.4: Integration test - Interactions among plugins involved in SSM/FSM deployment

3.2.3 Tests requirements

3.2.3.1 Initial configuration of the environment

- The MANO framework (Message Broker, Plugin Manager, SLM, FLM and SMR) should be running;
- The descriptors of the service (NSD) and the VNFs (VNFD) should be available. The NSD contains the instructions to deploy one SSM, and one of the VNFDs contains an instruction to deploy one FSM;
- The images of the referenced SSM and FSM in the descriptor should be reachable by the SMR.

3.2.3.2 Expected results from modules:

- The SMR responses to the SLM on the onboarding request for the SSM with a message that contains status "ON-BOARDED";
- The SMR responses to the SLM on the instantiation request for the SSM with a message that contains status "INSTANTIATED";
- The SMR responses to the FLM on the onboarding request for the FSM with a message that contains status "ON-BOARDED";
- The SMR responses to the FLM on the instantiation request for the FSM with a message that contains status "INSTANTIATED".

3.2.4 API method tested

- SLM:
  - Trigger the onboarding of an SSM;
  - Trigger the instantiation of an SSM.
- FLM:
  - Trigger the onboarding of an FSM;
  - Trigger the instantiation of an FSM.
- SMR:
  - Handle SSM and FSM onboard requests;
  - Handle SSM and FSM instantiation requests;
  - Download SSM and FSM images;
  - Start SSMs and FSMs;
  - Register SSMs and FSMs.

3.2.5 Test triggers

- Jenkins;
- Scheduled on need basis.

3.2.6 Actions after the test

- Send an email to *lead developers*;
- Start other integration tests related to service deployment.

3.3 Service lifecycle management, function lifecycle management and infrastructure abstraction

3.3.1 Test description

This test targets the integration between the MANO framework, in particular the Service Lifecycle Management (SLM) plugin, the Function Lifecycle Management (FLM), and the Infrastructure Adaptor (IA). The purpose of this test is to trigger the SLM to deploy a new service instance. For this, a message is injected from the Jenkins Job into the service platform message bus. This triggers the SLM, which uses the message bus to publish a message to infrastructure.compute.list, in order to retrieve a list of the VIMs available to host the service, along with resource availability information, such as resource quota limits and utilisation.

The SLM uses this resource information to calculate a placement of the service, and sends a request to the IA with the list of selected PoP (VIM) where the service will be deployed, along with the list of the VM images that need to be pre-loaded on the PoPs. The VIM adaptor receives the message and connect to each VIM listed in the payload to pre-load the given VM images and to create a stack template with the basic networks and environment resource and pushes it to the OpenStack Heat endpoint. Once the deployment is over, it returns a response message to the SLM.

The SLM can then start the VNF deployment. For each VNF in the service, the SLM will trigger the FLM to deploy it, by sending it the relevant VNF descriptor (VNFD) on the mano.function.deploy topic. To request the deployment of the VNF, the FLM will publish a message on the topic "infrastructure.function.deploy" containing the VNFD, the UUID of the VIM where the function should be deployed, as calculated by the placement, and other context information. The Infrastructure Abstraction layer uses the VIM Adaptor to translate the descriptors in the VIM specific language and connects to OpenStack Heat for the deployment. The VIM Adaptor waits for the stack to be updated or for an error message from the VIM. Once the deployment is completed, the VIM Adaptor collects the relevant instance information from OpenStack, forges the response message and sends it to the FLM. The FLM builds the record associated with this running VNF, and sends it to the SLM.

Next, the SLM requests the IA the chain the deployed VNFs together into a service. This is done by sending the NSD to the IA on the infrastructure.service.chain topic. Once finished, the IA responses with an indication whether the chaining was successful. If successful, the SLM requests the IA to configure the WAN, so that traffic can start flowing through the VNFs. This is done on the infrastucture.wan.configure topic, to which the IA again replies with a status indicating the whether the WAN was configured successfully or not. All these interactions are depicted in Figure 3.5 and Figure 3.6.

3.3.2 Infrastructure requirements

This test makes use of the following repositories:

- Son-mano-framework;
- RabbitMQ;

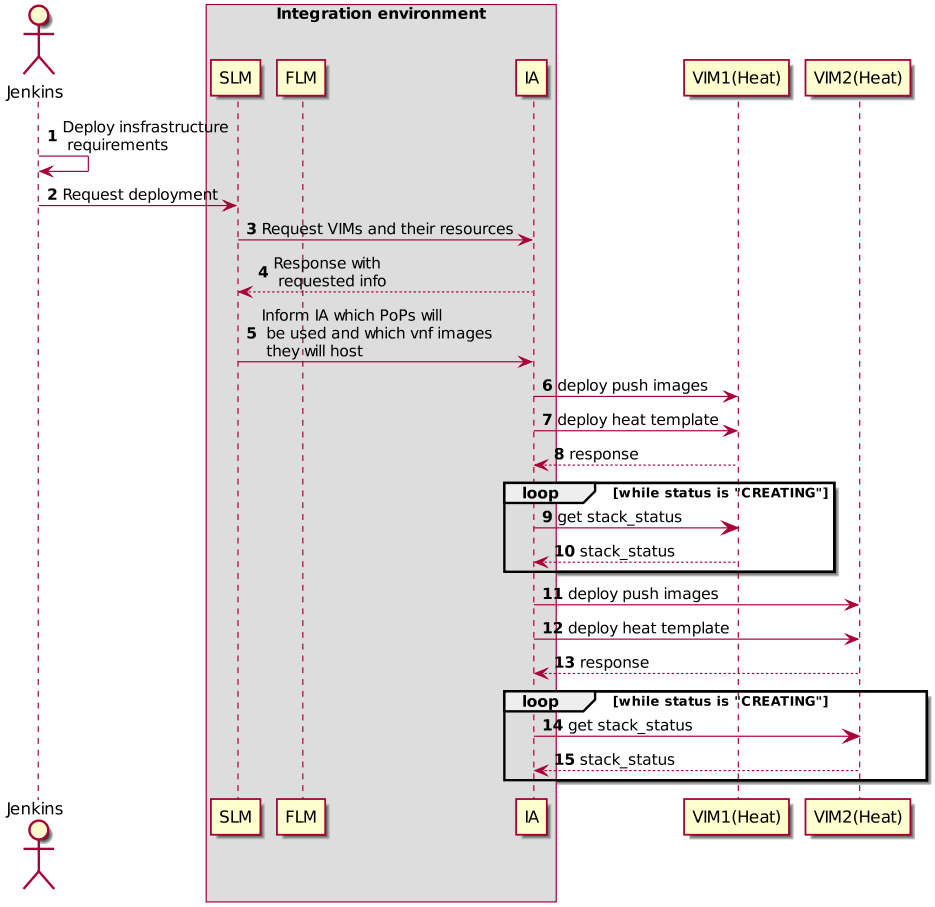

Figure 3.5: Integration test - SLM and FLM deploying a service IA

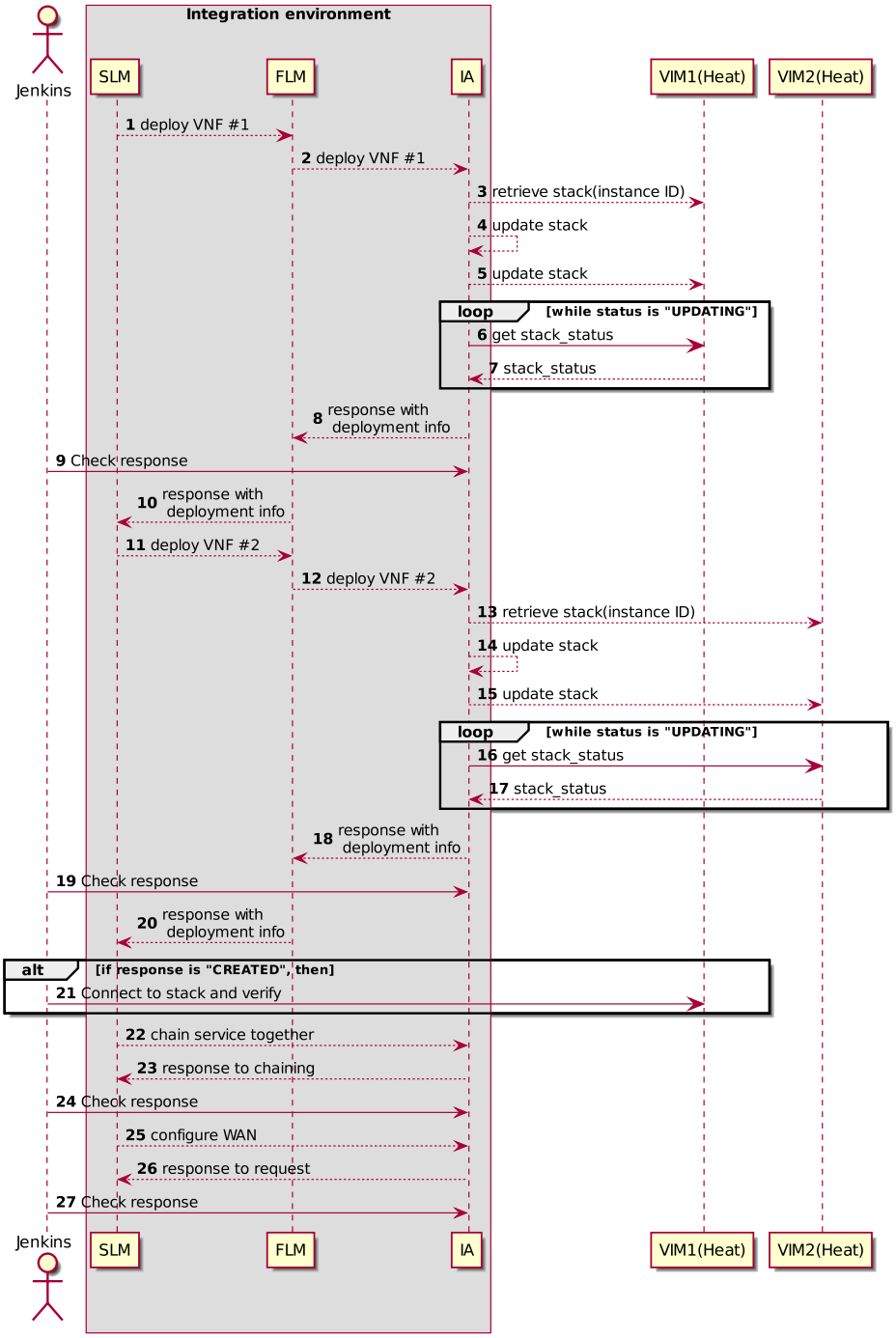

Figure 3.6: Integration test - SLM and FLM deploying a service IA

- MongoDB;
- Son-sp-infrabstract;
- OpenStack instance with Heat.

3.3.3 Tests requirements

3.3.3.1 Initial configuration of the environment

- The MANO framework (Message Broker, Plugin Manager, Service Lifecycle Manager and Function Lifecycle Manager) and the Infrastructure Adaptor should be running;
- OpenStack VIM should be installed and running;
- The OpenStack VIM is registered to the service platform;
- The descriptors of the service (NSD) and the VNFs (VNFD) should be available.

3.3.3.2 Expected results from modules:

- The Infrastructure Adaptor responses with status “RUNNING”, error “None” and other vnf instance information on each of the VNF deploy requests;
- The Infrastructure Adaptor responses with status “RUNNING”, error “None” and other service instance information on the service chain request;
- The Infrastructure Adaptor responses with status “CONFIGURED” and error “None” on the WAN configure request.

3.3.3.3 API method tested

- Infrastructure Abstraction:
  - List available VIMs with available resources;
  - Prepare Service Environment;
  - Pre-Load VM images;
  - Deploy VNF.
- SLM:
  - Requesting the deployment of a new service;
  - Requesting the topology of the available infrastructure, with usage information;
  - Requesting the deployment of a VNF;
  - Requesting to chain VNFs together into a service;
  - Requesting the configure the WAN.
- FLM:
  - Requesting the deployment of a VNF.

3.3.4 Test triggers

- Jenkins;
- Scheduled on need basis.

3.3.5 Actions after the test

- Send an email to *lead developers*;
- Start other integration tests related to service deployment.

3.4 KPI module - monitoring system

This chapter contains information about the integration test between the SONATA's Monitoring System and the new Gatekeeper KPI module.

3.4.1 Test description

This test targets the integration between the next components: Gatekeeper's API, Gatekeeper's KPI module and the monitoring system. Specifically it tests the functionalities needed to register the relevant actions performed by the gatekeeper through its API module, and to elaborate metrics and key performace indicators that can be accessed through the monitoring system. The test flow is shwon in Figure 3.7

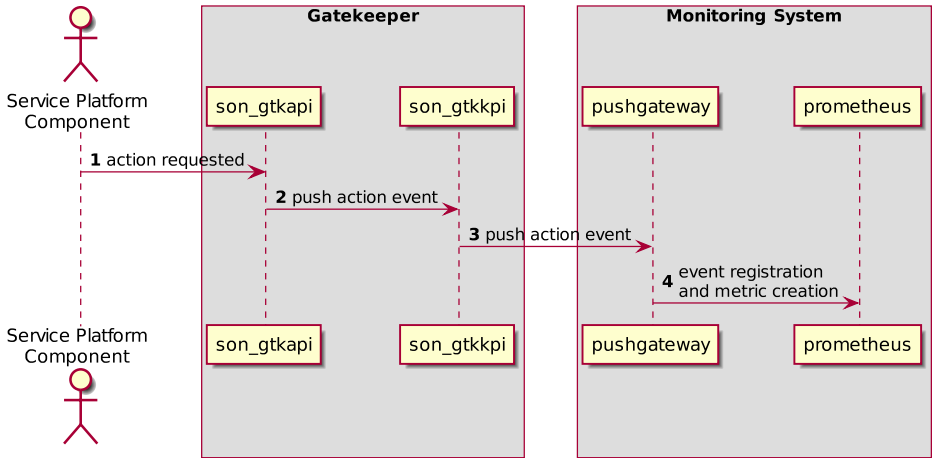

Figure 3.7: Integration test - KPI creation

3.4.2 Infrastructure requirements

This test makes use of the following components:

- Gatekeeper:
  - API: son-gkeeper/son-gtkapi;
  - KPI: son-gkeeper/son-gtkkpi.
- Monitoring System:

- Prometheus: son-monitor/prometheus;
- Pushgateway: son-monitor/pushgateway;
- InfluxDb: son-monitor/influxDB;

3.4.3 Tests requirements

This sections reflects the initial configuration needed to perform the test and the expected results to achieve.

3.4.3.1 Initial configuration of the environment

Installed and running Gatekeeper API, Gatekeeper KPI module and the monitoring system.
From Gatekeeper

- KPI method to receive the API event creation requests.

From Monitoring System

- `pushgateway` component that creates the metrics based in the GK's KPI push requests;
- `prometheus` module properly configured to receive the pushgateway events;
- `influxDB` database to store the data.

3.4.3.2 Expected results from modules

From Monitoring System:

- Stored list of the relevant Gatekeeper events and metrics;

3.4.4 Test Triggers

- Jenkins (daily);
- Scheduled on need basis.

3.4.5 Actions after the test

- Send an email to *lead developers*;
- Start other integration tests related to service deployment.

4 Qualification tests

In this section we detail the qualification tests we designed to ensure the functional behaviour and the compliance to the functional requirements elicited in Deliverables D2.2 [4] and D2.3 [5]. A brief introduction to this phase is given in Section 1.2 and a full description of the qualification environment and scope can be found in Deliverable D5.2 [10].

4.1 Validation of the qualification environment

4.1.1 Test description

The target of this qualification test is to check the correct execution of SONATA service platform installation scripts. For this qualification tests, multiple VMs will be created with each Operative System supported by SP. At this moment, the service platform can be installed in the following Linux versions: Ubuntu 14.40, Ubuntu 16.04 and CentOS 7

The qualification test will start, running the ansible installation script in the Virtual Machine under test. When the installation script have had finished, then a battery of test will start. The goal of this tests is check that each component of the SONATA service platform were deployed successfully. The tests will check the APIs, databases, GUI and monitoring system. All these interactions are depicted on Message Sequence Chart of Qualification Test 1, SONATA service platform Deployment, in Figure 4.1 and Figure 4.2.

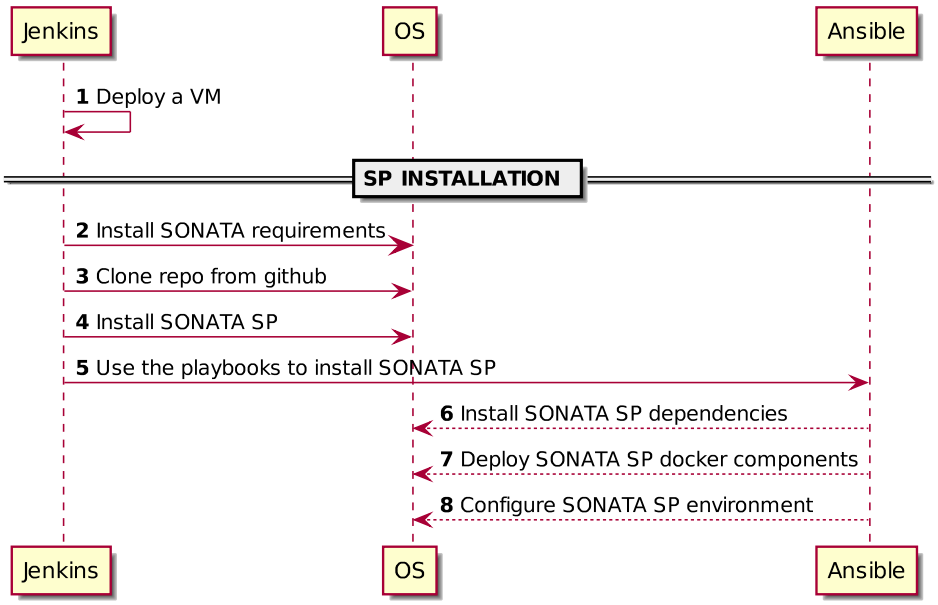

Figure 4.1: Qualification test - Install SONATA service platform

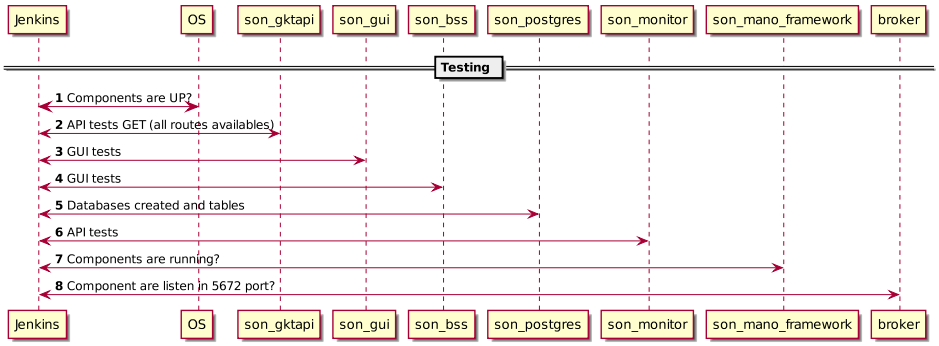

Figure 4.2: Qualification test - Test SONATA service platform installation

4.1.2 Infrastructure requirements

This test makes use of the following repositories:

- Son-install;
- Service platform Repositories;
- Openstack Instance.

4.1.3 Tests requirements

- The Openstack environment should be available to instantiate the VM;
- Multiple Operating Systems images in glace (Ubuntu 14.04, Ubuntu 16.04 and Centos 7).

4.1.3.1 Initial configuration of the environment

- A virtual machine to install the SONATA service platform.

4.1.3.2 Expected results from modules:

A successful deployment of the Qualification environment instantiates a set of dedicated VMs running the following services:

- VM for the SP framework: running a full set of SP services, namely Gatekeeper, Repositories, MANO framework, Infrastructure Adapter.

To verify the global functionality of the Qualification environment, just execute the following playbook:
`son-cmud.yml -e “environ=qual operation=test service=all”`
This playbook provides a list of the running services and its status.
To verify a single service on the Qualification environment, just pass the service identification to the playbook and inquire for its status:
`son-cmud.yml -e “environ=qual operation=test service='SVC_ID'”`

4.1.4 Test triggers

- Jenkins;
- Scheduled periodically.

4.1.5 Actions after the test

- Send an email to *lead developers*;
- Start other qualification tests related to service deployment.

4.2 Onboard a service from SDK to SP

4.2.1 Test description

This qualification test, performs a package onboarding from SDK to service platform. In an initial stage, the service platform database is empty and is verified. A Package is created in the SDK using the SDK tools and this package is uploaded to the service platform by the Package onboarding procedure. Once the package is uploaded, a set of test is triggered in order to verify if the services, functions and packages match with the services,functions and packages uploaded.

4.2.2 Infrastructure requirements

- Installed and configured SONATA SDK;
- Installed and configured SONATA service platform.

4.2.3 Tests requirements

- Custom template for package creation;

4.2.3.1 Initial configuration of the environment

- service platform without package, functions and services;
- NS and VNF template available for package creation.

4.2.3.2 Expected results from modules:

- service platform database empty;
- Once the package is uploaded to the service platform, services, functions and packages must match the services, functions and packages created in the SDK.

4.2.4 Test triggers

- Jenkins;
- Scheduled periodically.

4.2.5 Actions after the test

- Send an email to *lead developers*;
- Start other qualification tests related to service deployment.

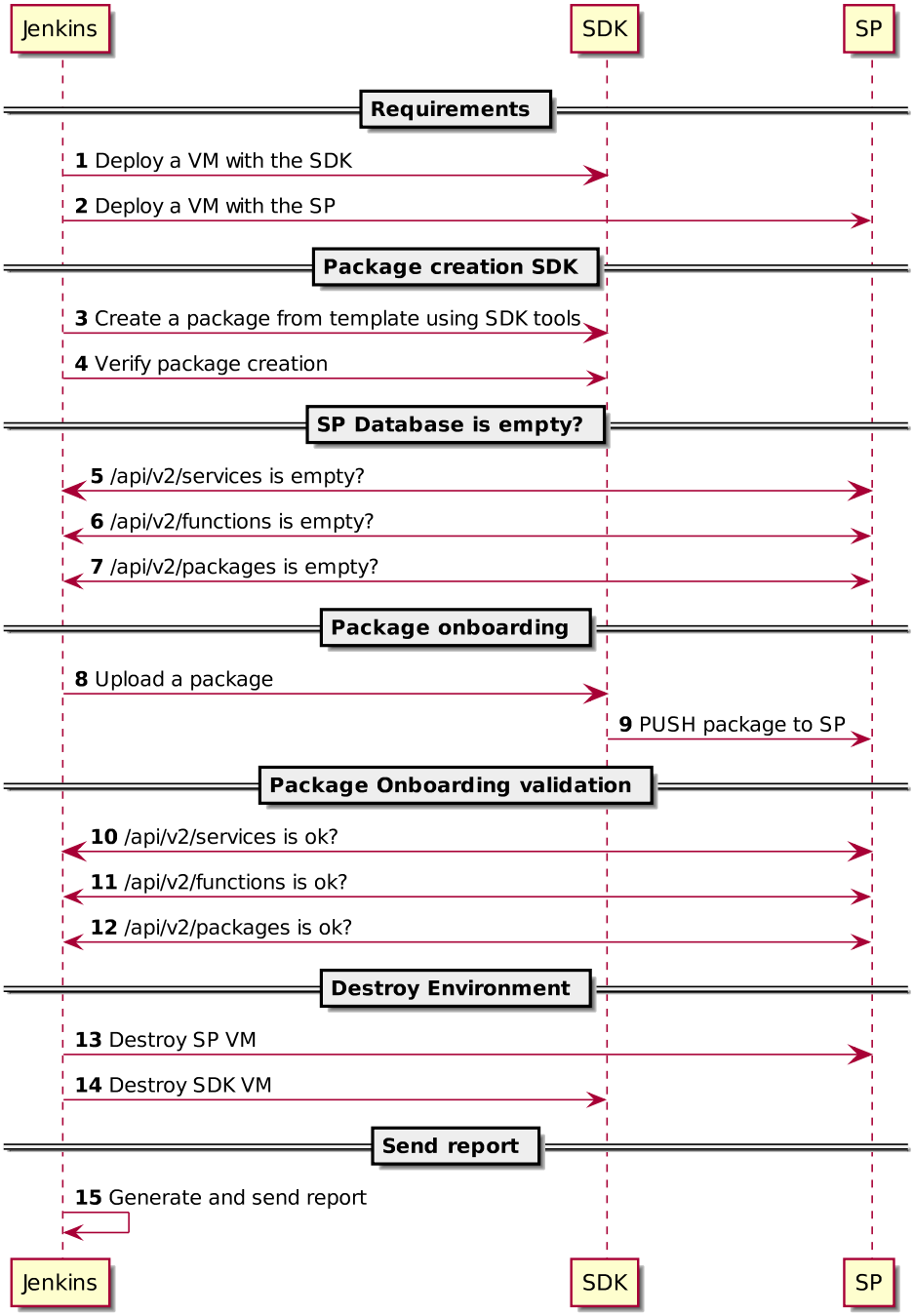

Figure 4.3: Qualification test - Onboard a service from SDK to service platform

4.3 Deploy a network service composed by 1 VNF

4.3.1 Test description

This qualification test performs a Network Service deployment of a service composed by a single VNFs with a public port. The service deployment request is sent to the BSS submodule. When the service is ready, the test checks that the VNF is reachable on the public port.

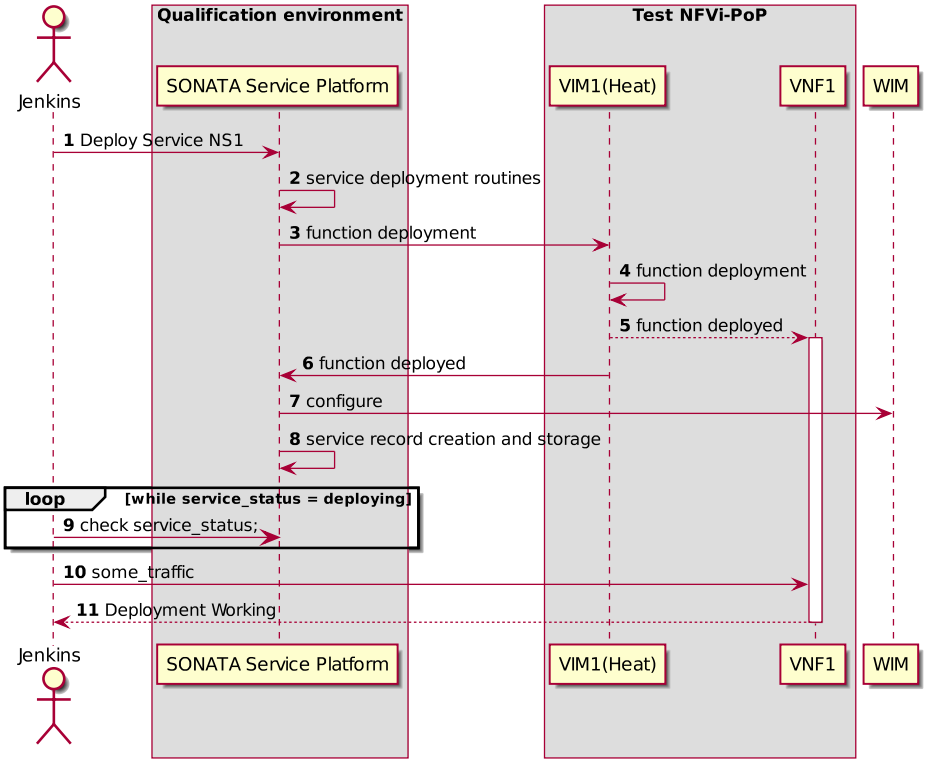

Figure 4.4: Qualification test - NS deploy 1VNF

4.3.2 Infrastructure requirements

- Installed and configured SONATA service platform;
- Custom VNF which listens on a specific port;

4.3.3 Tests requirements

- NS composed of one VNFs;
- Descriptors are stored in the catalogue.

4.3.3.1 Initial configuration of the environment

- OpenStack VIM should be installed and running;
- The OpenStack VIM is registered to the service platform;
- NSD & VNFD.

4.3.3.2 Expected results from modules:

- The VNF receives the ping, and it sends a report to Jenkins.

4.3.4 Test triggers

- Jenkins;
- Scheduled periodically.

4.3.5 Actions after the test

- Send an email to *lead developers*;
- Start other qualification tests related to service deployment.

4.4 Deploy a network service composed by 2 VNF

4.4.1 Test description

This qualification test performs a Network Service deployment of a service composed by a two VNFs, configured as a chain. The service deployment request is sent to the BSS submodule. When the service is ready, the test checks that the traffic is routed through the VNFs.

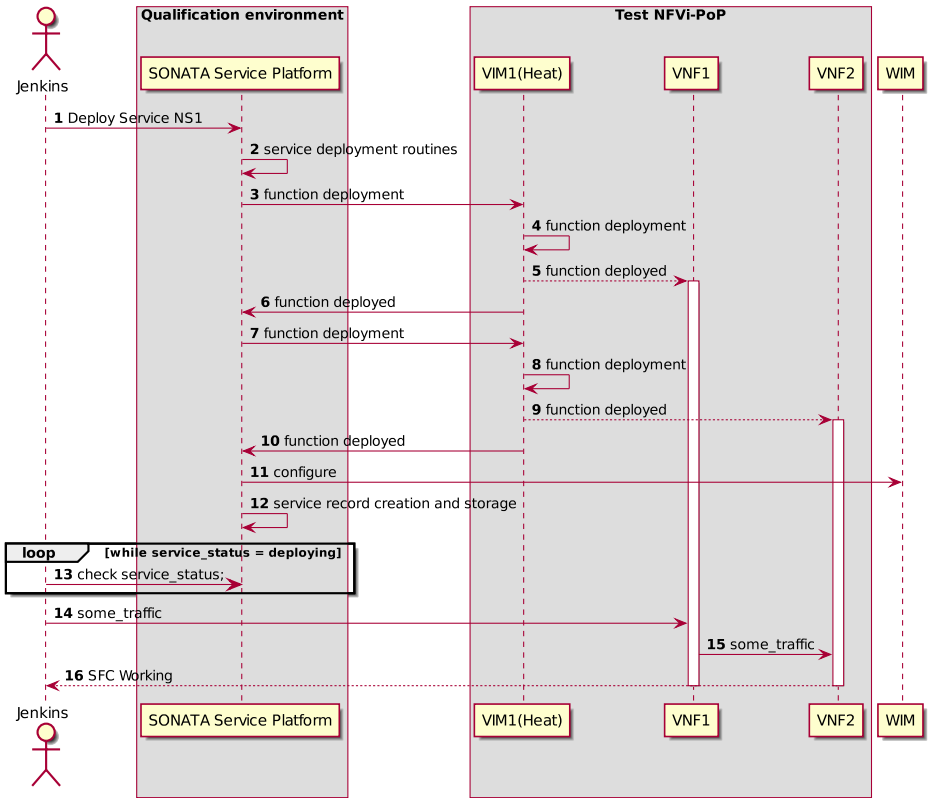

Figure 4.5: Qualification test - NS deploy 2VNF 1PoP

4.4.2 Infrastructure requirements

- Installed and configured SONATA service platform;
- Custom VNFs to sent SFC report.

4.4.3 Tests requirements

- NS composed of two VNFs;
- Descriptors are stored in the catalogue.

4.4.3.1 Initial configuration of the environment

- OpenStack VIM should be installed and running;
- The OpenStack VIM is registered to the service platform;
- NSD & VNFDs.

4.4.3.2 Expected results from modules:

- The traffic flows through the VNFs, and VNF2 sends report to Jenkins.

4.4.4 Test triggers

- Jenkins;
- Scheduled periodically.

4.4.5 Actions after the test

- Send an email to *lead developers*;
- Start other qualification tests related to service deployment.

4.5 Deploy a network service composed by 2 VNF, distributed on two PoP

4.5.1 Test description

This qualification test performs a Network Service deployment of a service composed by a two VNFs, configured as a chain, which are deployed on two NFVi PoP. The service deployment request is sent to the BSS submodule. When the service is ready, the test checks that the traffic is routed through the VNFs.

4.5.2 Infrastructure requirements

- Installed and configured SONATA service platform;
- Two Openstack PoP configured in the IA;
- Custom VNFs to sent SFC report.

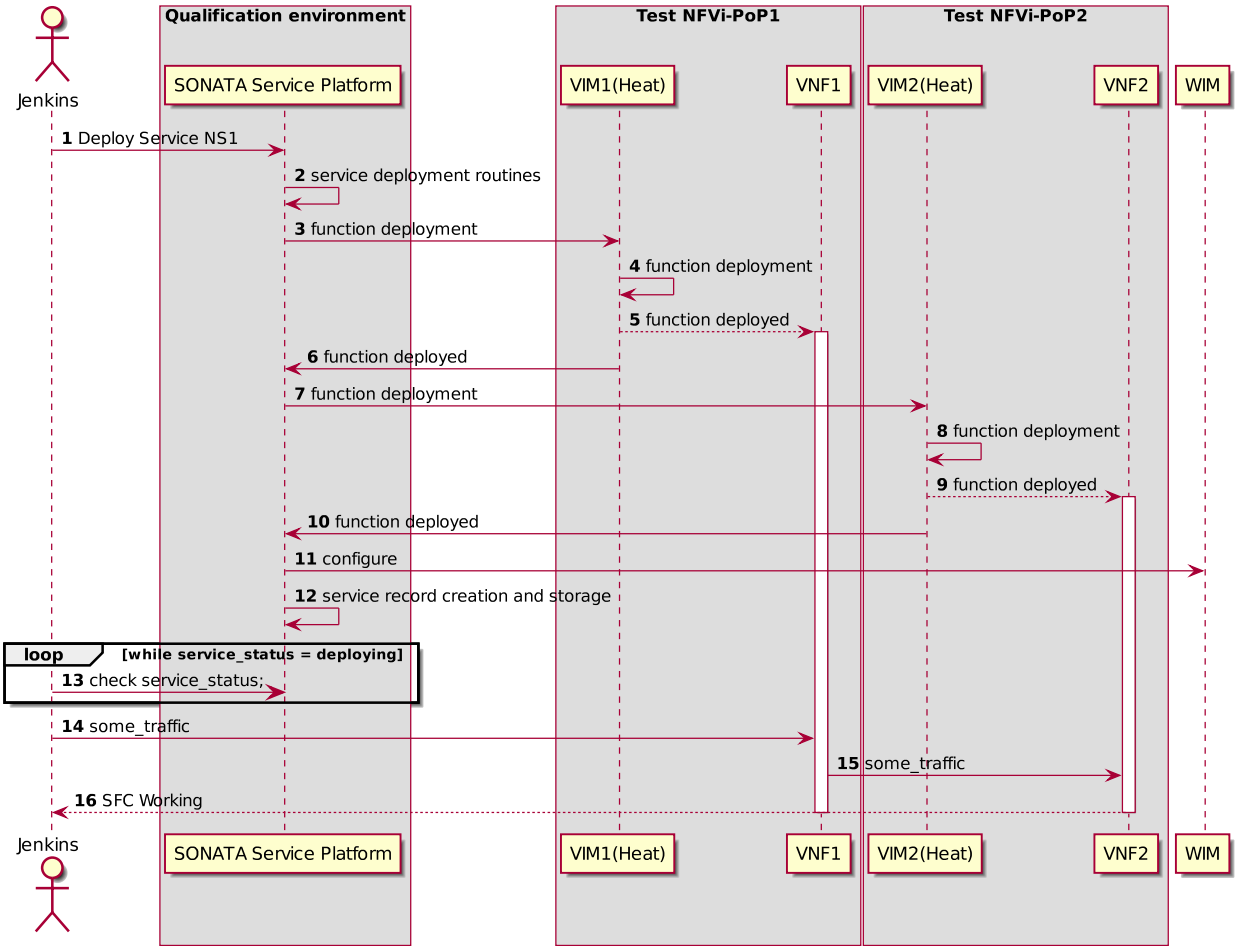

Figure 4.6: Qualification test - NS deploy 2VNF 2PoP

4.5.3 Tests requirements

- NS composed of two VNFs;
- Descriptors are stored in the catalogue.

4.5.3.1 Initial configuration of the environment

- OpenStack VIM should be installed and running;
- The OpenStack VIM is registered to the service platform;
- NSD & VNFDs.

4.5.3.2 Expected results from modules:

- The traffic flows through the VNFs, and VNF2 sends report to Jenkins.

4.5.4 Test triggers

- Jenkins;
- Scheduled periodically.

4.5.5 Actions after the test

- Send an email to *lead developers*;
- Start other qualification tests related to service deployment.

4.6 Deploy a network service composed by 2 VNF and SSM, distributed on two PoP

4.6.1 Test description

This qualification test performs a Network Service deployment of a service composed by two VNFs, configured as a chain, and including a SSM. The VNFs are deployed on two different NFVi PoP. The service deployment request is sent to the BSS sub-module. When the service is ready, the test inject high volume of traffic through the VNFs, causing CPU overload. The monitoring system issues an alert and the SSM that is shipped with the service configures the VNFs to throttle traffic. Jenkins check the VNFs for the CPU overload and for the throttle to be working.

4.6.2 Infrastructure requirements

- Installed and configured SONATA service platform;
- VNFs: VNF1 is firewall, VNF2 a traffic classifier;
- SSM: firewall manager.

4.6.3 Tests requirements

- NS composed of the custom VNFs and the SSM.

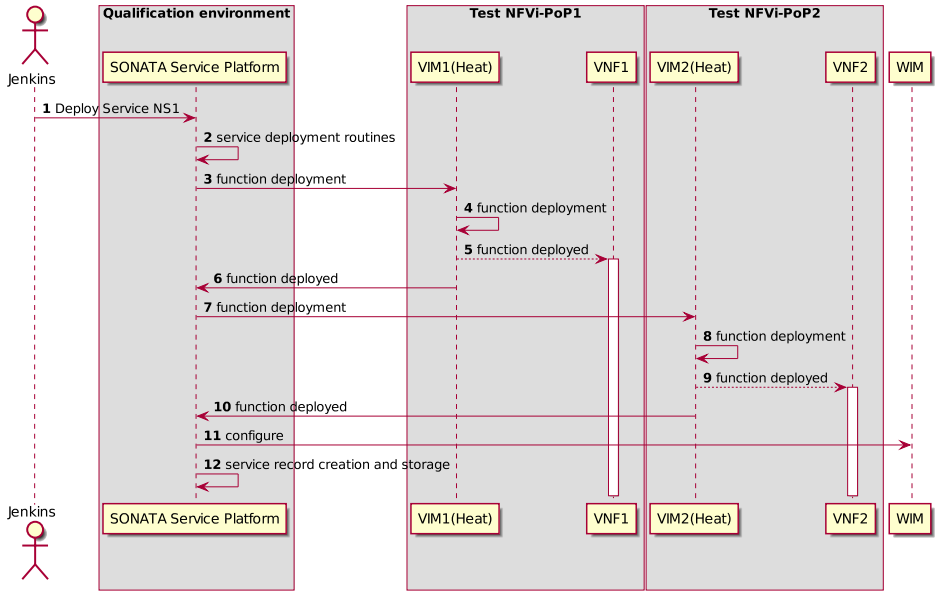

Figure 4.7: Qualification test - NS deploy 2 VNF and SSM in 2 PoP - Deployment phase

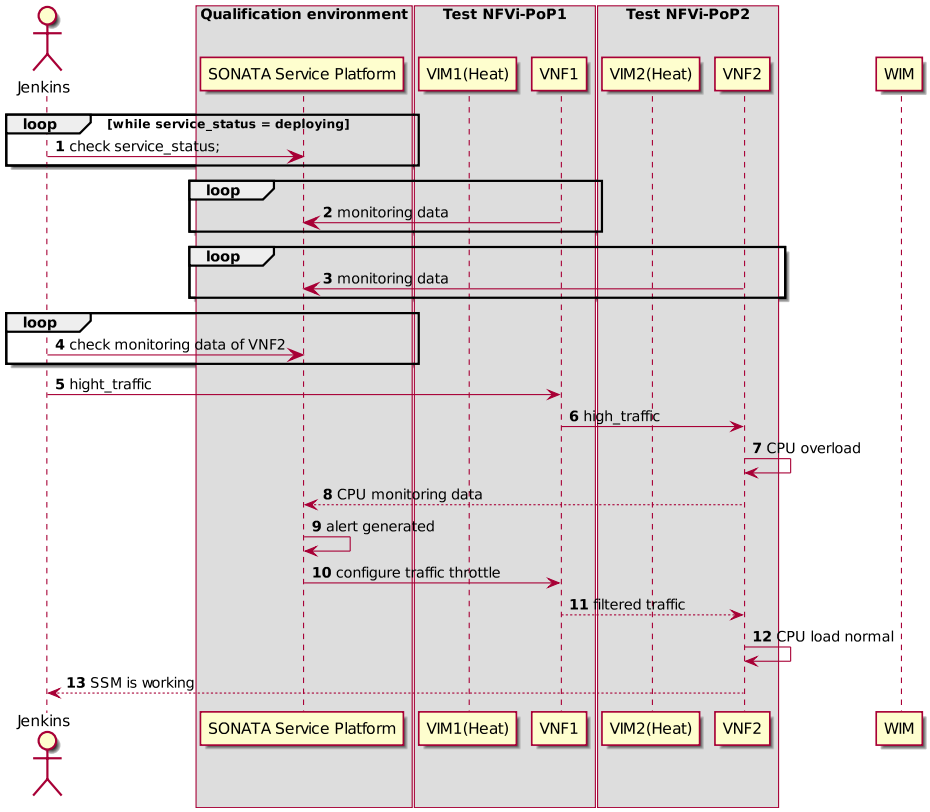

Figure 4.8: Qualification test - NS deploy 2 VNF and SSM in 2 PoP - Ops phase

Initial configuration of the environment

- OpenStack VIM should be installed and running;
- The OpenStack VIM is registered to the service platform;
- NSD & VNFDs.

Expected results from modules:

- Traffic flows through the SFC;
- The SSM handles alerts and configure the firewall;
- CPU load of VNF2 increases for the high traffic volume and decreases after VNF1 configuration.

4.6.4 Test triggers

- Jenkins;
- Scheduled periodically.

4.6.5 Actions after the test

- Send an email to *lead developers*;
- Start other qualification tests related to service deployment.

5 Final considerations

5.1 Considerations on CI/CD added value

Our DevOps approach, implemented through a CI/CD (continous integration/continous delivery) development pipeline, has been documented and detailed in Deliverable D5.2 [10]. Here we outline the benefit this management choice which has been brought into the project. The SONATA software base has been developed by a large, distributed team of around 50 developers, belonging to 15 different partners scattered around Europe. They have contributed continously to the development. With such an heterogeneous and largely distributed team, software integration issues were considered a high risk for the outcome of the project. By embracing the DevOps approach, and especially the CI/CD way of working, combined with the endemic micro-service nature of our development team, has allowed all the partners to rely on an automated and tested way of contributing their code.

Another aspect of our chosen approach is that as SONATA is an open source software project, it aims to produce impact also through attracting developers from other open source communities in order to: contribute to the development of features; bug-tracking; fixing issues; eliciting requirements for new functionalities, and so on. Without a solid and stable software integration mechanism, this approach would simply be impossible for two reasons. First, in order to attract other developers and suscitate interest from other communities, the code base of SONATA should have a high level of quality. Code documentation, stability, and modularity are crucial to engurage people to join efforts. Second, contributions coming from external developers need to be thoroughly checked and integrated, following a rigid, and above all, automated integration flow.

In conclusion, the lessons we have learned from our experience in DevOps is that **breaking the silos** works. By letting Development and Operational teams work more closely together, and using a CI/CD approach can really help achieve the level of stability, reliability, responsivenes, flexibility, and openess that the software industry demands from current software products.

5.2 Future plans

A definition of use case scenario relevant to the project is carried out in parallel to this work in the framework of the project work package 6. Further requirements will be elicited and fetched as an input to the development pipeline, so that new features and functions will be developed, integrated and qualified following our CI/CD approach, as described in [10]. As these new features emerge, qualification and integration phase will be expanded to ensure the stability of the software base. In addition, the use case scenario definition will produce also a list of NSs and VNFs that should be designed and developed, in order to be used to validate the outcome of the SONATA development.

Integrating and implementing network slicing in the SONATA platform will be considered as on of the main future development. As shown in reports by Focus Group of ITU-T IMT-2020 [12], network slice is defined as a complete end-to-end logically partitioned network providing dedicated telecommunication services and network capabilities. The behaviour of the network slice is realised via network slice instance(s). A network slice instance may also be shared across multiple service instances provided by the network operator. The network slice instance may be composed by

none, one or more sub-network instances, which may be shared by another network slice instance. Network slicing enables the operator to create logically partitioned networks customised to provide optimised solutions for different market scenarios. These scenarios demand diverse requirements in terms of service characteristics, required functionality, performance and isolation issues. For this purpose, some modules of the SONATA environment will need further extension, which include:

- Catalogues: new functionality in these modules will be added to enable slicing capacity exposure to service providers and users;
- Introduction of Slice Management entity: this module will enable the composition and decomposition of network slices;
- Infrastructure Adaptor: efficient methods need to be added to this module to enable end-to-end service set-up on slice spanning multiple PoPs;
- SONATA user management: in order to expose resources provided by slicing to different user in a secure and flexible way, the user management of SONATA will need to be extended to consider the slice capability and functions exposure.

In addition, some new modules will need to be implemented. For example, Resource Management will be needed to handle complicated resource exposure, discovery, composition, and allocation to various service providers and users. The built in resource manager in each PoP will not be enough, as it does not handle resource allocation at the slice level.

Another aspect of the integration work in the future would be the investigation of the possible integration of SONATA outcome in other relevant projects. Here we just give the bullet points of the envisioned collaboration and plans:

- Open Source MANO [13] is an open source Management and Orchestration (MANO) stack aligned with ETSI NFV Information Models. Since also SONATA is aligned with the same Information Model, SDK and Gatekeeper module could be integrated to the OSM stack to provide VNF development facilities and access control for developers and users;
- Open O `https://www.open-o.org` - the OPEN-Orchestrator Project whose goal is to enable end-to-end service agility across SDN, NFV, and legacy networks via a unified orchestration platform supporting NFV orchestration (NFVO and VNFM) and SDN orchestration. Open O is targeting telecommunications and cable operators, OEMs, systems integrators, and software companies, similar to the SONATA focus;
- OpenStack Tacker [2] is an official OpenStack project building a Generic VNF Manager (VNFM) and a NFV Orchestrator (NFVO) to deploy and operate Network Services and Virtual Network Functions (VNFs) on an NFV infrastructure platform like OpenStack. Also in this regard, SONATA environment could offer compatibility in terms of VNF development tools and also in term of descriptors.

On another side, works are in progress to allow external tools to be integrated in SONATA architecture as plugin. In particular we are targeting the integration of OpenStack Mistral [1] as a workflow manager to implement SSM/FSM or even Function Life-cycle management.

6 Conclusion

In this Deliverable we have described the second integrated release of the SONATA environment. The document documents the new features and functionality offered by this second release, also presenting new modules introduced in the architecture and integrated in the code-base. At the very core of our development stands our CI/CD methodology and the use of software management and continuous integration tools such as Jenkins and GitHub. This has the development team to keep on improving the SONATA components in a independent, *agile* way, continuously pushing new features and functions, and leveraging our automated system of integration to grant the stability of the software released. On this matter, we listed and detailed the updates to several Integration tests, which have been modified or created in order to cope with the new features foreseen by SONATA v2.0. Finally, qualification phase has been documented, with a range of qualification tests able to validate the outcome of the integration process in terms of reliability and end-to-end interaction with the users. In the future, along with the feature development described in Section 5.2, more effort will be put on the development of VNFs related to the project use cases, so to validate the outcome of the software integration and qualification also in terms of the project pilots.

A SONATA Release 2.0

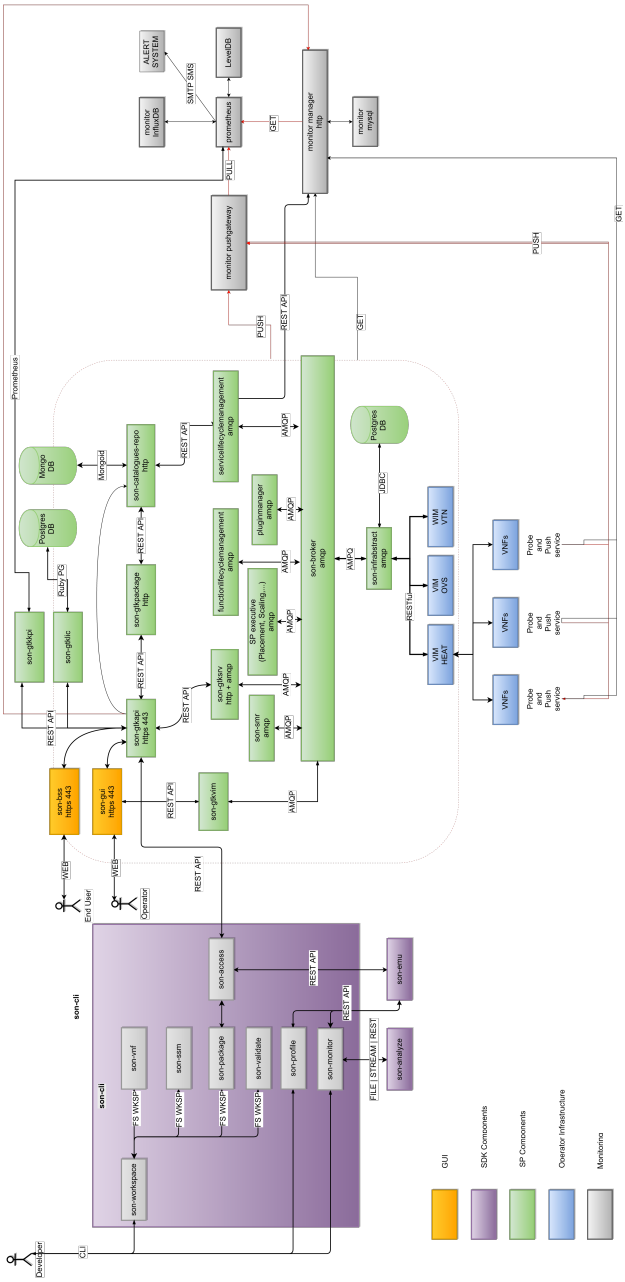

Figure A.1: SONATA Release 2.0

B Bibliography

[1] The OpenStack Community. Openstack mistral project. Website, Dec 2016. Online at `https://wiki.openstack.org/wiki/Mistral`.

[2] The OpenStack Community. Openstack tacker project. Website, May 2016. Online at `https://wiki.openstack.org/wiki/tacker/`.

[3] SONATA consortium. H2020-ict-2014-2 - proposal submission forms. Website, November 2014.

[4] SONATA consortium. D2.2 architecture design. Website, December 2015. Online at `http://www.sonata-nfv.eu/content/d22-architecture-design-0`.

[5] SONATA consortium. D2.3 updated requirements and architecture design. Website, December 2016. Online at `http://www.sonata-nfv.eu/`.

[6] SONATA consortium. D3.1: Basic sdk prototype. Website, May 2016. Online at `http://www.sonata-nfv.eu/content/d31-basic-sdk-prototype`.

[7] SONATA consortium. D3.2 sdk operational release and documentation. Website, December 2016. Online at `http://www.sonata-nfv.eu/`.

[8] SONATA consortium. D4.1: Orchestrator prototype. Website, May 2016. Online at `http://www.sonata-nfv.eu/content/d41-orchestrator-prototype`.

[9] SONATA consortium. D4.2: Service platform operational release and documentation. Website, December 2016.

[10] SONATA consortium. D5.2: Integrated lab based sonata platform. Website, June 2016. Online at `http://www.sonata-nfv.eu/content/d52-integrated-lab-based-sonata-platform`.

[11] The leading tool for querying and visualizing time series and metrics. Online at `http://grafana.org/`.

[12] ITU-T. Website. http://www.itu.int/en/ITU-T/focusgroups/imt-2020/Pages/default.aspx.

[13] OSM. Google guice: Agile lightweight dependency injection framework, 2016. Online at `https://osm.etsi.org/`.

[14] Manuel Peuster, Holger Karl, and Steven van Rossem. MEdiCinE: Rapid Prototyping of Production-Ready Network Services in Multi-Pop Environments. In *Network Function Virtualization and Software Defined Network (NFV-SDN), 2016 IEEE Conference on*. IEEE, 2016.