



Escola d'Enginyeria de Telecomunicació i
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Developing and Deploying NFV solutions with OpenStack, Kubernetes and Docker

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Adrià Martí Luque

ADVISOR: Jesus Alcober

DATE: July, 10th 2019

Abstract

This document explains the deployment of virtualized network functions in a private cloud through ONAP, together with the architecture used, the machines employed and the software required.

ONAP (Open Networking Automation Platform) is an open source project of the Linux Foundation, which provides a platform for real-time orchestration and automation of physical and virtual network functions. ONAP allows designing, orchestrating and managing all the elements related to virtualized network functions.

In order to deploy ONAP, we acquired and installed a cloud infrastructure from scratch following the latest market best practices, which included setting up the servers and network devices, configuring their NICs and installing all the software required: OpenStack, ONAP, etc.

Moreover, these software solutions also have to be properly configured and installed, something that will be clarified in this paper along with the configuration of the cloud network.

ONAP, being a novel solution, is not fully developed. Meaning it has presented many challenges and setbacks we have had to overcome in order to deploy it successfully in our environment.

The final goal of this project is to create an environment in a private cloud with working virtualized network functions and to serve as a guide for other people who desire to deploy ONAP, overcoming all the obstacles we had to face and providing solutions, or at least, alternatives to the plethora of issues they might stumble across, just as we did.

Everything discussed in this document can be replicated in environments of greater scope than the ones defined in the paper, to the point that what is explained can even be applied in production scenarios by telephone operators.

Special thanks to

The Bampla's research group, specially to Leo and Adri for all the support they gave us during the project

Imgflip for providing the best memes when we needed them most

Kyoki Chiho for dealing with the most fingerbreaker task when bending the steel

The big owl we had every day in the lab working tirelessly with ONAP

And finally, thanks to Toni and Jesus for the given support during all the project, always available when needed

INDEX

INTRODUCTION	5
CHAPTER 1. OPPORTUNITIES AND GOALS OF THE PROJECT	6
CHAPTER 2. ONAP	7
2.1. ONAP [4]-[6]	7
2.2. ONAP's users management	10
2.3. ONAP's service orchestration [10]-[12]	12
2.4. VPN Setup	15
CHAPTER 3. VIRTUAL FIREWALL USE CASE	17
3.1. Virtual Firewall use case explanation [14]-[16]	17
3.2. Virtual Firewall use case analysis	19
3.2.1. Virtual Firewall service creation	19
3.2.2. Virtual Firewall instantiation	34
3.2.3. Virtual Firewall deployment	37
3.2.4. Virtual Firewall issues.....	40
CONCLUSIONS	41
BIBLIOGRAPHY	42
ANNEX I: ONAP MODULES	46
I.1. Portal	46
I.2. AAI.....	46
I.3. SO	47
I.4. SDC	48
I.5. SDNC	49
I.6. DCAE.....	49
I.7. DMAAP.....	50
I.8. VID.....	51
I.9. Robot	52
I.10. APPC.....	52
I.11. Policy	53
ANNEX II: VIRTUAL FIREWALL IMPLEMENTATION.....	55
II.1. vSINK and vPKG interaction	57

INTRODUCTION

Nowadays, one of the most heard buzzwords is virtualization, but this recurring term has been around since the 1960s when it simply meant to divide a computer's resources between different applications. With the years, the actual meaning of the term has substantially broadened [1].

ONAP (Open Networking Automation Platform) is an open source project of the Linux Foundation, which provides a platform for real-time orchestration and automation of physical and virtual network functions that allows IT producers, cloud providers, and developers to quickly automate new services and support the complete management of their life cycle [2].

Its architecture is very innovative because it is divided into two environments; a run-time framework dedicated to the design of the infrastructure (virtualized services, virtual network functions, or VNF, and physical network functions, or PNF), and another framework employed during the execution time (creation and management of service instances, VNF and PNF), making ONAP one of the preferred options of the different network providers [3].

ONAP, being a novel solution, is not fully developed. Meaning it has presented many challenges and setbacks we have had to overcome in order to deploy it successfully in our environment.

The document is organized as follows:

Firstly, we describe the architecture designed to offer these virtualized network functions.

Secondly, we explain in detail how ONAP works.

Afterward, we describe the steps we followed in order to deploy network functions and then, finally, the results obtained and the overall conclusions.

Unfortunately, due to unsolved problems coming from ONAP itself, we haven't been able to make a full deployment of the network services.

This paper is the follow-up of another project called "Developing and deploying advanced NFV solutions", which had the goal of deploying ONAP's infrastructure.

That infrastructure has been employed in this work in order to create ONAP's virtualized network functions.

CHAPTER 1. OPPORTUNITIES AND GOALS OF THE PROJECT

The aim of this project is to take advantage of virtualization technologies in order to implement cutting-edge solutions, especially in the field of network service providing and service management in the cloud, looking to get rid of the nowadays monolithic solutions for clouds and get closer to a portable, efficient, scalable, and virtualized approach.

From this remarkable opportunity, this project is born. The objective is to offer NFV services, such as DNS or Firewall, using the Open Network Automation Platform (ONAP) running on top of an OpenStack private cloud emulating a realistic scenario.

Basically, this project has to serve as a guide for other people who desire to deploy ONAP in a real environment, overcoming all the obstacles we had to face and providing solutions, or at least, alternatives to the plethora of issues they might stumble across, just as we did.

Even if virtualization has clear benefits, many companies have qualms in using virtualized solutions because it is a very different way to do what they are currently doing.

With this in mind, this project has to show the clear advantages virtualization has over legacy solutions implementing fully functional virtualized network functions.

Also, instead of implementing them with any of the ONAP's automated scripts their team use for the known use cases, we are facing the project as a realistic scenario, following all the steps a real operator should perform in order to design virtualized functions.

However, to implement these functions, a deep insight into ONAP is required. Everything related to ONAP and the way it works will be explained in detail in the following chapter.

In order to deploy ONAP, we employed a brand new cloud infrastructure with enough resources to be able to fulfill ONAP's requirements.

Moreover, ONAP's network functions have to be properly configured and implemented, something that will be clarified in one of the following chapters along with the explanation of how they work.

Additionally, the results of this project have been presented to JITEL, an important Spanish telecom forum, so they can publish a paper with our work and further study ONAP's advantages.

CHAPTER 2. ONAP

In this chapter we explain what ONAP is and how it works, including how ONAP manages multiple users, how it orchestrates services, and how to configure a VPN to access the ONAP internal services.

2.1. ONAP [4]-[6]

ONAP, which stands for Open Network Automation Platform, is an open source networking project hosted by the Linux Foundation that brings the capabilities for designing, creating, orchestrating and handling the full lifecycle management of VNF, Software Defined Networking (SDN), and all of the services related to these.

In short, ONAP is the platform above the physical infrastructure layer that automates the network.

ONAP allows connecting products and services through the physical infrastructure, deploy VNFs and scale the network in a fully automated way.



Fig. 2.1 ONAP logotype

The high-level architecture of ONAP has many different software subsystems that are part of the design environment as well as part of an execution time environment that runs what is designed in the previous environment.

Since ONAP enables VNFs, other network functions, and services to be interoperable in an automated, policy-driven, real-time environment, this allows the ability to create, design and deploy automated network services anywhere.

ONAP has the goal to bring the DevOps best practices and dynamic AGILE deployment methods to the Telecom world. This will reduce the barriers for new users to enter the world of cloud technologies and network virtualization.

It will also enable operators to quickly add new features, deploy them in real time, reduce operational costs and have much more control over the network and the services that are available. This way, the network will be able to work in a much more efficient manner and consumers will benefit from the new services and the improved experience.

Moreover, as ONAP evolves, the ONAP community also defines blueprints for key use cases such as 5G, CCVPN (Cross Circuit Virtual Private Networks), Voice over LTE (VoLTE), and vCPE (virtual Customer Premises Equipment).

As a cloud-native application based in numerous services, ONAP requires a sophisticated initial deployment as well as some post-deployment management.

The ONAP Operations Manager (OOM) is responsible for orchestrating the lifecycle management and monitoring of ONAP components. It is integrated with the Microservices Bus, which provides service registration/discovery and support for internal and external APIs and SDKs.

OOM uses Kubernetes to provide CPU efficiency and platform deployment. Its architecture and the main operations OOM can perform can be seen in Figure 2.2. In addition, OOM enhances the ONAP platform by providing scalability and resiliency enhancements to the components it manages.

The platform on itself provides tools for service designing as well as a model-driven, run-time environment, with monitoring and analytics to support automation and service optimization.

Both design-time and run-time environments are accessed through the Portal Framework, with role-based access for service designers and operations.

The design-time framework provides a development environment with tools, techniques, and repositories for defining and describing resources, services, and other products that include policy design and implementation, SDK with tools for VNF packaging and validation.

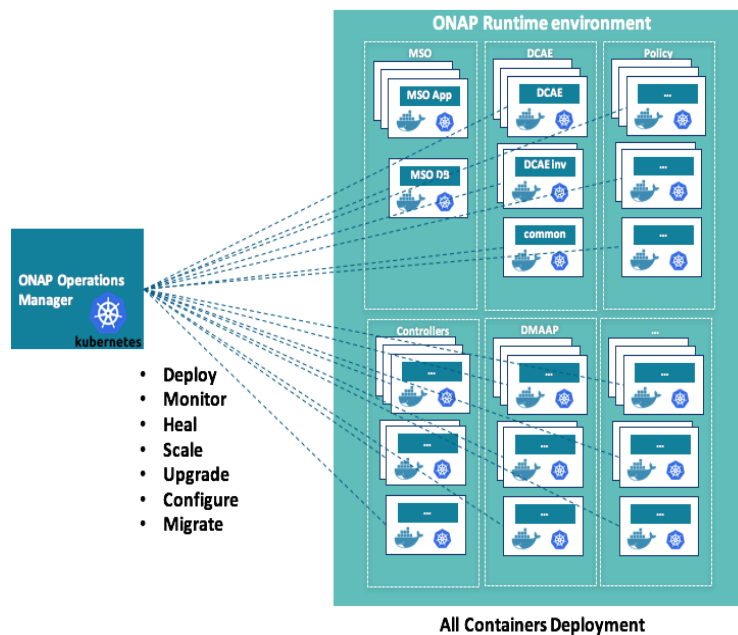


Fig. 2.2 OOM architecture

The run-time environment applies the rules and policies distributed by the design and creation environment, as well as the controllers that manage physical and virtual networks.

The Active & Available Inventory (A&AI or AAI) component provides real-time information of a system’s resources, services, products and the relationships with each other.

In a fast-moving environment with speedy deployments and teardown of virtual resources, these monitoring and mapping features are paramount. The run-time service execution components are in constant communication with the automation modules, which provide real-time monitoring, analytics, alarms, and event correlation.

The complete ONAP architecture can be observed in the Figure 2.3.

These modules provide continuously updated data about existing services to service designers, who will deem if it is necessary to tune them, replicate high-performing portions of deployed services or directly creating new ones.

ONAP has a very long list of modules, however, not all of them are needed to implement the minimal functionality required to deploy virtualized network functions. The most important modules are explained in the Annex I.

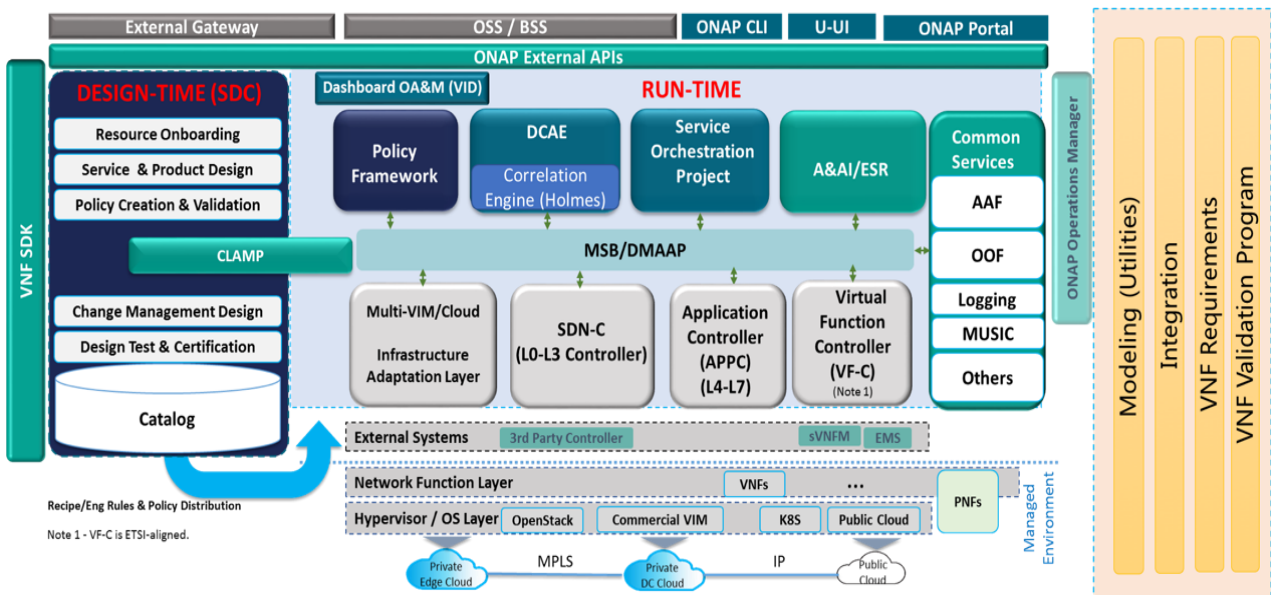


Fig. 2.3 ONAP’s architecture

2.2. ONAP's users management

In order to access the different ONAP modules, user management techniques are required to make the whole process safer and more organized from the administrator's point of view, as each kind of user only can perform a set of defined actions.

The user roles are the following:

- Administrator: Triggers the actions to ultimately deploy the virtual function module.
- Designer: Responsible of designing and creating all the SDC resources.
- Tester: In charge of running the tests for the designed services and approving the ones in which the tests were successful.
- Governor: Responsible of approving or denying the designs created by the designer after they have been verified by the tester.
- Operation: It distributes the service throughout the platform.

Each user role is pre-assigned by ONAP's setup in order to ease the user management tasks newcomers have to perform when they join for the first time.

These pre-defined users can be found in the Figure 2.4, all with the same password, which is "demo123456!" [7].

User	Role
jh0003	ADMIN
cs0008	DESIGNER
jm0007	TESTER
op0001	OPS
gv0001	GOVERNOR

Fig. 2.4 ONAP pre-defined users

Moreover, for more advanced users or if the network operator wants to enable access for more than one user, the platform also offers a user management interface. However, in order to use it, we must login as administrator.

From this interface, more users can be added, modified, and deleted, selecting even the different modules to which the user will have access to, thus enabling the ability of working with different users at the same time performing similar roles [8].

On the other hand, different users accessing the same module at the same time will share the environment's resources, meaning that they will be able to interact with the service design and instantiation cycle of other users as if they were using the same application.

This can be compared to having multiple end users sharing the same edge computer in the Telco world.

An example of the management view is shown in Figure 2.5.

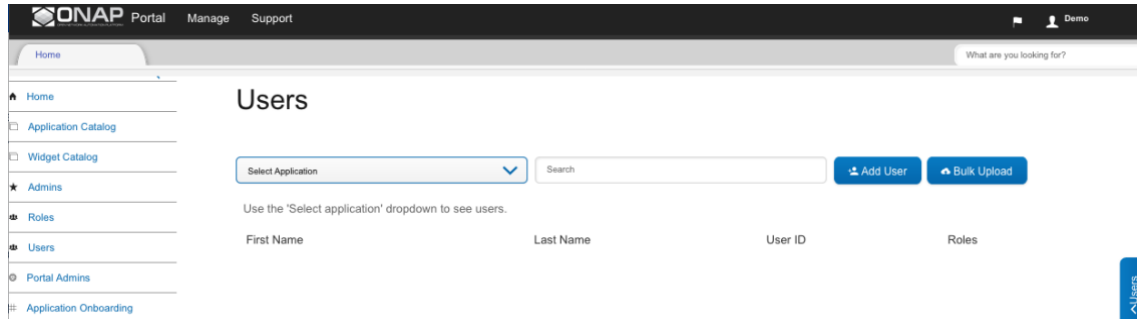


Fig. 2.5 ONAP’s user management

As this way of managing users can be a serious shortcoming, which also affects user privacy, ONAP has developed a native multi-tenancy proposal by exploiting the capabilities of kubernetes of creating multi-tenant clusters.

With this capability in action, the tenant’s resources will no longer be accessible by other foreign users, hence ensuring the privacy of the services created.

An example of the proposed slicing by ONAP can be seen in Figure 2.6. [9]

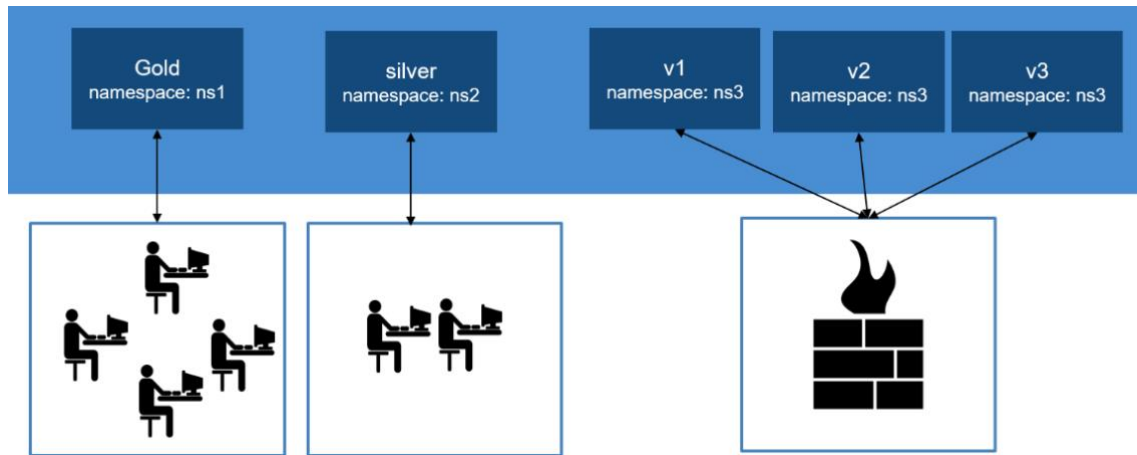


Fig. 2.6 Block diagram representation of tenant resource slicing

This multi-tenancy proposal also implies more resources available from the cluster’s point of view, as each tenant will have its own resources and namespace in the kubernetes cluster so as to allocate the different services.

Lastly, as it has been mentioned before, this is only a proposal, meaning that it has not been implemented by ONAP yet. It is expected to be supported in some of the final releases of Dublin (June 6, 2019) or in El Alto upcoming release (30 Sep, 2019).

2.3. ONAP's service orchestration [10]-[12]

ONAP, from the network provider's point of view, is a platform that offers automatic orchestration and virtualization of network functions in the cloud in real time, allowing the quick deployment of services.

These services are based on artifacts, independent models that band together without requiring the manual intervention of highly qualified network personnel.

As it has been commented in the previous point, ONAP can be used to automate SDN and NFV services, but to reduce complexity and allow any provider to use the platform, the ONAP community has created ready-to-use blueprints/artifacts with the most useful use cases.

These curated artifacts are:

- Virtual firewall (vFW)
- Residential virtual customer premise equipment (vCPE)
- Voice over LTE (VoLTE)
- Virtual Load Balancer (vLB)
- Virtual DNS (vDNS)
- Cross Domain and Cross Layer VPN (CCVPN)

ONAP, in the latest release, which is Dublin, is developing and expanding these artifacts to include 5G use cases such as network slicing.

Out of the many currently working artifacts, the virtual firewall is the simplest, and because of that, it's the use case we are going to explain and implement.

However, to run through the vFW use case, two sets of infrastructure are required, the first one being a cloud environment, preferably deployed using Kubernetes, and the second one being OpenStack to manage the underlying NFV infrastructure (NFVI) layer for the vFW network service.

Moreover, ONAP, in order to deploy services, requires some modules. ONAP has a very extensive list of modules, which vary given the use case, but since we wanted to make a full deployment, we have installed them all at the beginning.

Once we ensured the correct functioning of the different modules, as the virtual Firewall use case only requires a simpler deployment, we decided to use only the modules necessary for this use case to work, thus not overloading the servers.

A summary of the modules can be seen in Figure 2.7

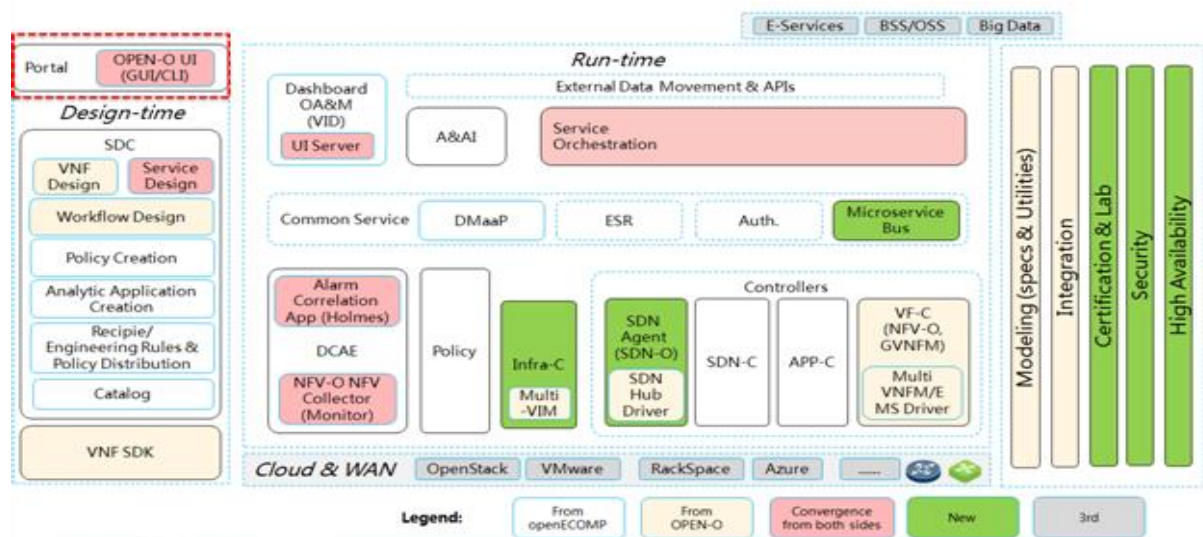


Fig. 2.7 ONAP’s modules and environments

Of all the modules shown in Figure 2.7, the most important one is Portal.

Principally, the ONAP Portal is a platform that offers the ability to integrate the different applications into a centralized core. The goal is to allow decentralized applications to run within their own infrastructure while providing common management services and connectivity.

The Portal core provides many capabilities, including application onboarding and management. Portal, regarding both users and administrators, is a web-based interface that provides access to all of the subsystems of an ONAP’s instance, that is to say, it gives design-time tools and run-time monitoring and control.

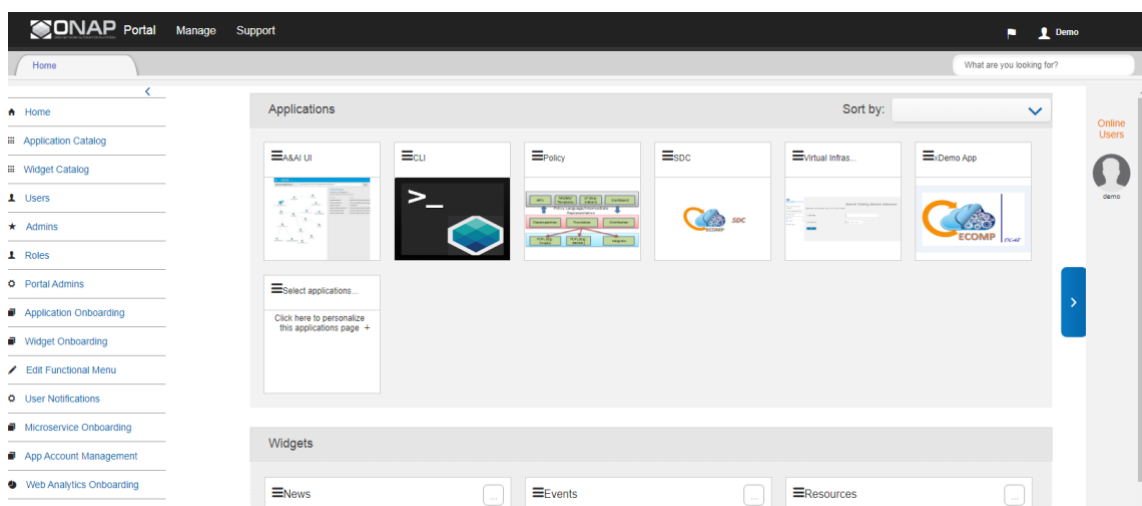


Fig. 2.8 Portal’s web-based graphic interface

Any user looking for access to an ONAP application will have to visit first the Portal, where an authentication procedure will be performed to determine the type of the user.

Based on the user's role previously defined, the Portal will let it access different application widgets, or even redirect the user to a specific run-time environment. When any user wants to deploy a service, they have to follow a specific life-cycle based on 4 steps:

- Design: Based on model resources and needs, it defines the service and resource's behavior.
- Testing: It monitors policies and it ensures its correct functioning.
- Approval: With the recorded events gathered in the testing stage, it confirms or refuses its deployment.
- Distribution: It designs the final templates and policies to be deployed in the cloud.

Additionally, from the Portal, users can access applications and Key Performance Indicators. On the other hand, administrators manage these applications, hosted application widgets, and user access.

In general, from a user's perspective, the operation of service deployment in ONAP consists of an ordered set of actions performed through scripts or using the Portal to trigger instantiation in the service orchestrator (SO).

To sum up, a service in ONAP is the composition of several elements that are based on VNFs, Networks, VF-modules and volume groups.

In order to fully deploy a service, three levels of instantiation are identified:

- Service instantiation
- VNF and/or network instantiation
- VF-module and/or volume group instantiation

Each step of instantiation corresponds to a set of orchestration tasks performed by the SO.

When the SO receives a request to create a service instance, it selects a process to define a workflow of actions in order to accomplish the service instantiation. This process includes series of tasks to be executed by ONAP execution-time components.

Three main user actions are performed in the Portal in order to fully deploy the service. These actions correspond to three HTTP requests sent to the SO, which are the following:

- Instantiate service: Service instance creation request.
- Add node: VNF instance creation request.
- Add virtual function module: VF-module creation request.

As it can be seen, these petitions refer to each level of instantiation previously shown.

From a high level point of view, each request toward the SO triggers the execution of an orchestration workflow. When receiving a request, the API Handler stores an orchestration request and selects a workflow to be executed by the orchestration engine.

To each workflow corresponds a script to be executed by the orchestration engine, which allows performing actions or even invoking other components.

To conclude, the workflow specifies a sequence of orchestration tasks to be carried out by ONAP's execution-time components. The process includes recursive calls to other orchestration sub-processes.

A representation of these processes can be seen in Figure 2.9.

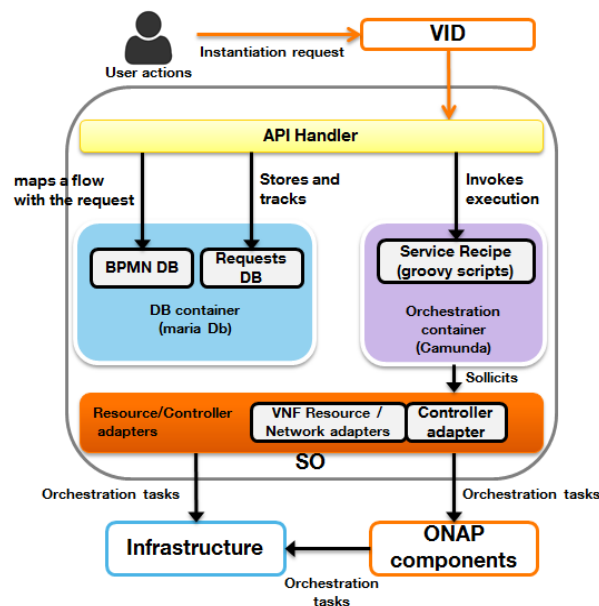


Fig. 2.9 Service orchestration workflow

2.4. VPN Setup

The different ONAP modules, as they are instantiated as pods inside the Kubernetes cluster, have 2 IP addresses, called “external” (the IP of the worker where the pod is deployed) and “internal” (the specific IP of that service inside kubernetes).

We analyzed our cluster setup and kubernetes' characteristics, and observed that, the internal IP address of the different ONAP modules is only accessible to other workers of the cluster.

Not even the rancher-server's virtual machine has access to any of the kubernetes services, which implies that the infrastructure is very resilient to external attacks, as no unauthorized person can access any of ONAP's services.

This supposed a drawback for us when working with ONAP, as many of the APIs and some of the GUIs are only offered by accessing directly via the internal IP.

Because of that, there were some characteristics we were unable to use when designing and instantiating services in ONAP, such as SDNC.

The Portal is the only available module we can access with the default cluster configuration, but only due to a set of specific rules we installed in our MikroTik, which exposed the Portal to the Internet.

In order to solve this obstacle and obtain the ability to access those specific modules, we deployed an OpenVPN server in one of the kubernetes workers and created a VPN.

OpenVPN is an open-source commercial software that implements virtual private network techniques to create secure point-to-point or site-to-site connections in routed or bridged configurations, like the one we have in our scenario. [13]

With this OpenVPN server, we were able to create a tunnel between our own laptops and one of the workers from the cluster, attaining as a result the possibility to connect to any internal IP from any of the ONAP's modules from our own web browser.

This way, we were also able to access visual interfaces for those specific modules not available with the default configuration from the cluster.

CHAPTER 3. VIRTUAL FIREWALL USE CASE

In this chapter we explain in detail the steps we followed in order to create, instantiate, and ultimately deploy network functions.

3.1. Virtual Firewall use case explanation [14]-[16]

The virtual firewall use case is composed of three virtual functions: packet generator (vPKG), firewall (vFW) and packet sink (vSNK). These functions run in three separate virtual machines.

Fundamentally, in this use case, the packet generator sends packets to the packet sink through the firewall, and the firewall reports the volume of traffic passing through to the ONAP collector.

To check the traffic volume that lands at the packet sink, it's possible to access through the browser and enable automatic page refresh to see the packets arrive in real time.

This scenario can be seen in Figure 3.1.

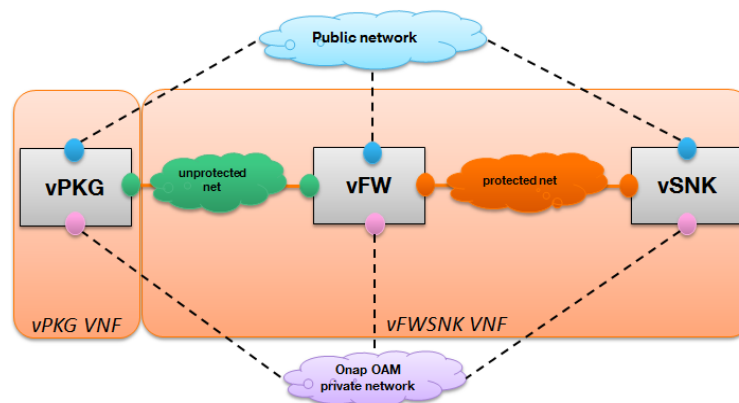


Fig. 3.1 Virtual firewall's scenario

In order to make the scenario more realistic, the packet generator includes a script that periodically generates different volumes of traffic. The vFW artifact is located in the ONAP Portal's Policy Portal, along with its configuration and operation policies.

The configuration policy is the one that sets the thresholds for generating an onset to the Policy engine. Moreover, the virtual firewall use case implemented in this document follows ONAP's "Closed Loop" paradigm.

This concept allows an automatic recovery of faults reported by traps or alarms and provides automated capacity management when performance thresholds are crossed.

It uses feedbacks to control and optimize the behavior of monitored elements in order to automatically respond to service conditions without manual intervention. The execution of a closed loop involves several components.

Basically, all the data is collected by the DCAE collector and then published to the DMaaP module. The policy engine allows triggering a response according to configuration policies when detecting critical events.

Lastly, an operational policy is executed by ONAP orchestrator or controllers.

This closed loop paradigm can be seen in Figure 3.2.

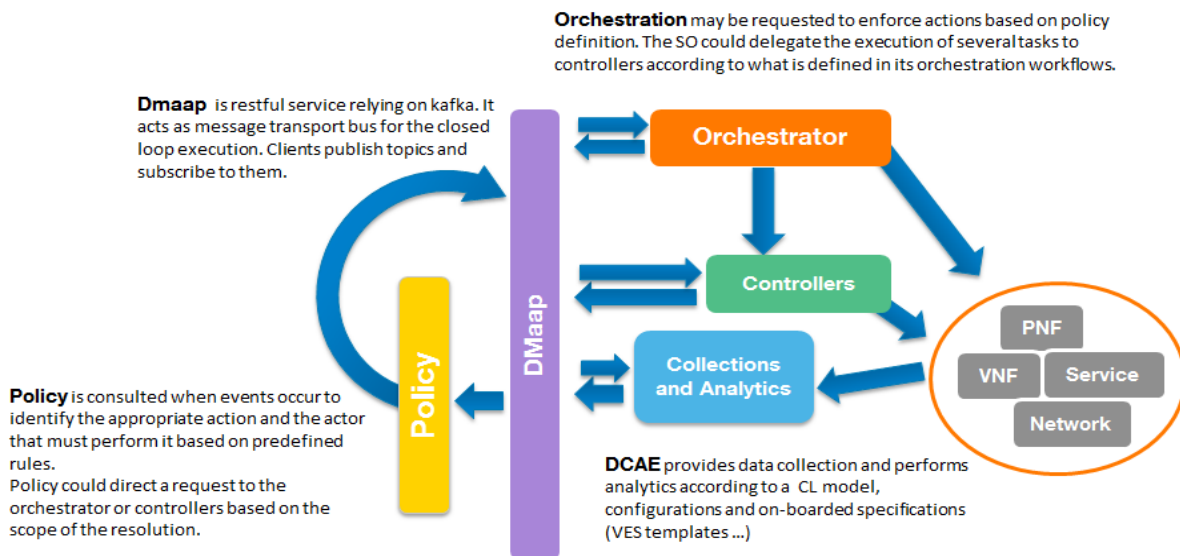


Fig. 3.2 Closed loop's architecture

By default, the closed-loop policy has been configured to re-adjust the traffic volume when high-water or low-water marks are crossed, where the high-water mark is 700 packets and the low-water mark is 300 packets. The measurement interval is 10 seconds.

When a threshold is crossed, that is to say, the number of received packets is below 300 packets or above 700 packets per 10 seconds, the Policy engine executes the operational policy to adjust the traffic volume to 500 packets per 10 seconds.

The packet generator has 10 streams, and each stream generates 100 packets per 10 seconds. A script on the packet generator automatically alternates between high traffic, 10 active streams at once, and low traffic, only 1 active stream, every 5 minutes.

3.2. Virtual Firewall use case analysis

In this point we describe which were the steps we followed in order to create, instantiate, and deploy the virtual firewall use case. We also explain the issues we found and how we solved them.

3.2.1. Virtual Firewall service creation

To create any service, the AAI and VID models have to be populated first with the right data required by the SDC so it can generate, connect, and distribute the services across the different ONAP modules afterward [16] [17] [18] [19].

In order to do so, we had to send HTTP petitions to the AAI's internal pod's IP by using the VPN explained before. Also, some special headers are needed in order to do such requests. The specific headers can be seen in Figure 3.3.

KEY	VALUE
Accept	application/json
Content-Type	application/json
X-FromAppld	AAI
X-TransactionId	get_aai_subscr
Authorization	Basic QUfJOkFBSQ==

Fig. 3.3 Headers for the AAI populating

We populated the AAI with 4 fields, which are the following:

- Service
- Cloud-region
- Model
- Customer

In the service petition, we sent the service's ID and the service's description via postman request, as it can be seen in the Figure 3.4.

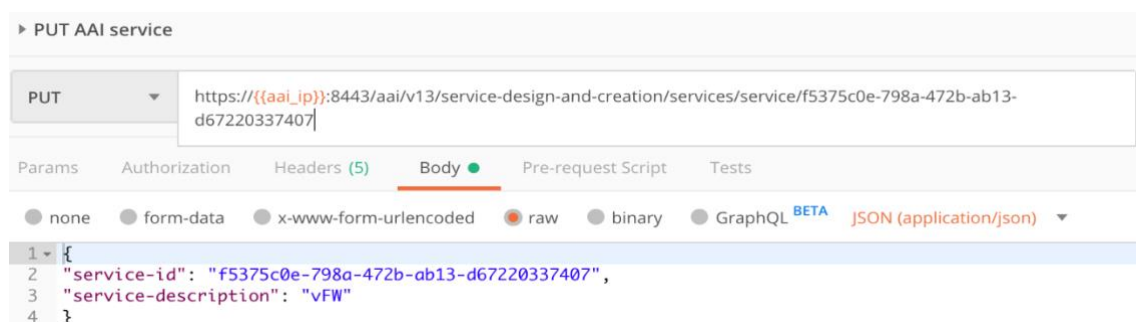


Fig. 3.5 PUT service request

This service will be specified in the SDC module as “service-type” so it can properly instantiate the service.

In the cloud-region petition, we specified the owner, the region where the cloud is located, the version of the cloud, the type of cloud, the zone inside the cloud, the type of ownership, and the tenants of the cloud. The whole body can be seen in Figure 3.6.

This will be used when creating the service's instance so we can define the region where the virtualized service has to be deployed.

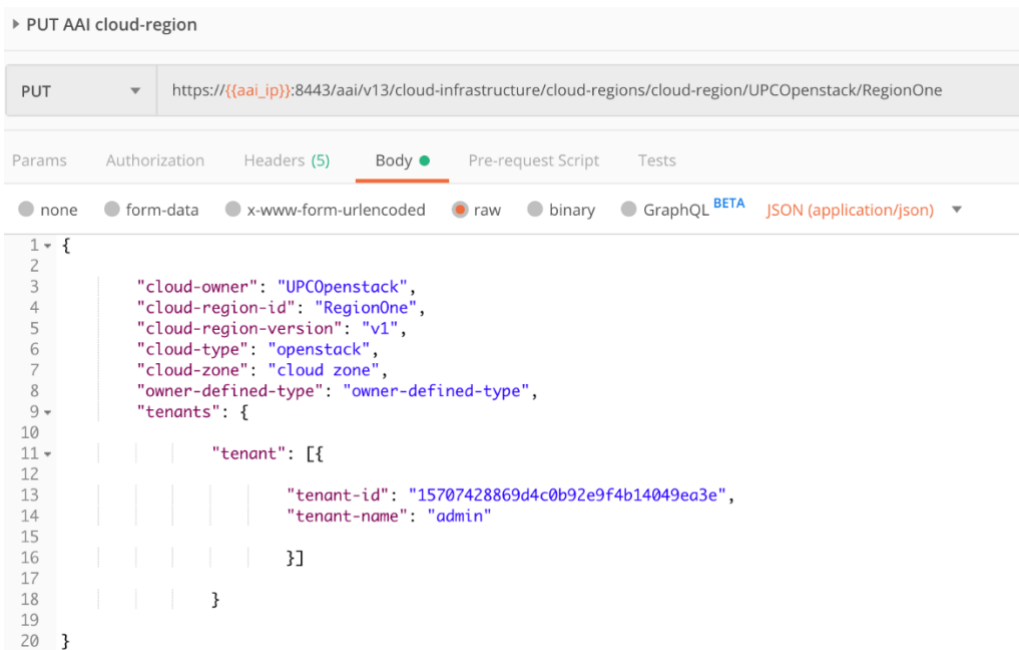


Fig. 3.6 PUT cloud-region request

In the model HTTP request, we defined the model variant, the model type, and the model version, which included the model version ID, the model name, and the model version on itself. Its whole body can be observed in Figure 3.7.

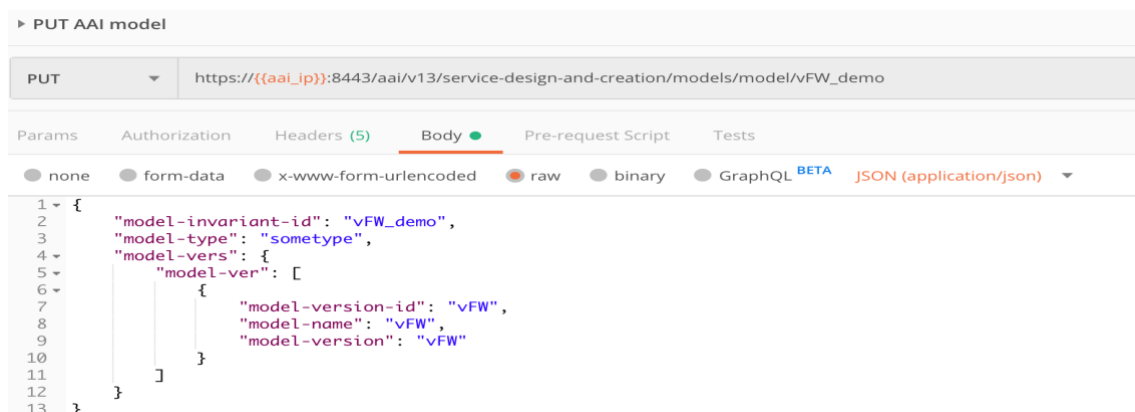
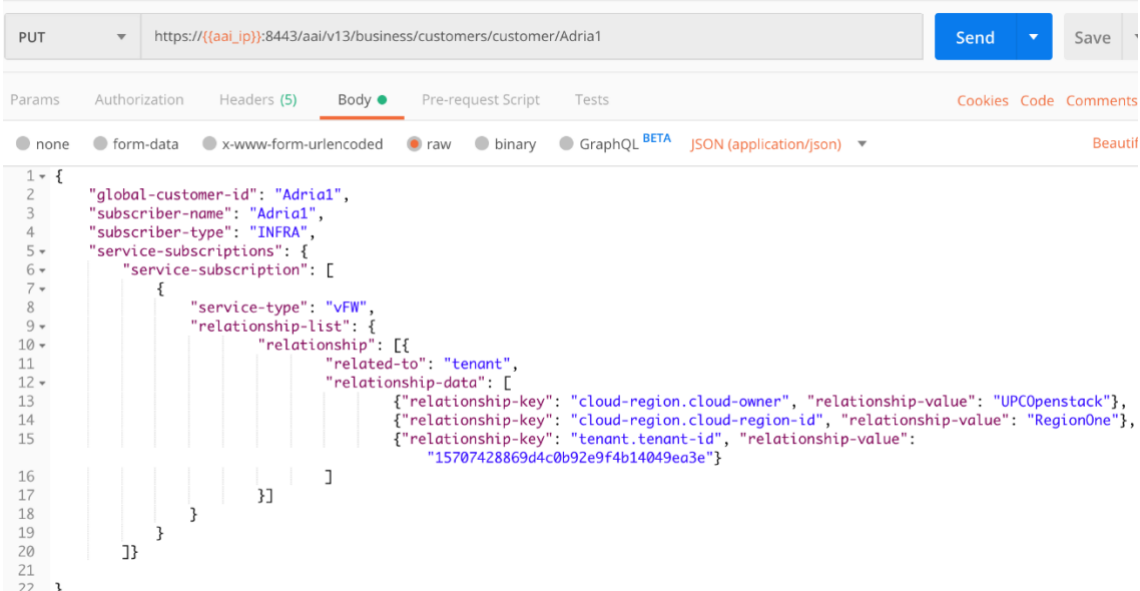


Fig. 3.7 PUT model request

Finally, inside the customer petition, we sent the customer's ID, the customer's name, the type of subscription, the services to which the customer is subscribed,

and its relationships to the rest of the cloud. The whole body can be seen in Figure 3.8.

This customer is used to indicate to which client we want to bind the instance to be deployed when creating the service instance. This is a very convenient way to differentiate between customers and their instances.



The screenshot shows a REST client interface with a PUT request to the URL `https://(aai_ip):8443/aai/v13/business/customers/customer/Adria1`. The body is a JSON object with the following structure:

```

1 {
2   "global-customer-id": "Adria1",
3   "subscriber-name": "Adria1",
4   "subscriber-type": "INFRA",
5   "service-subscriptions": {
6     "service-subscription": [
7       {
8         "service-type": "vFW",
9         "relationship-list": {
10          "relationship": [{
11            "related-to": "tenant",
12            "relationship-data": [
13              {"relationship-key": "cloud-region.cloud-owner", "relationship-value": "UPCOpenstack"},
14              {"relationship-key": "cloud-region.cloud-region-id", "relationship-value": "RegionOne"},
15              {"relationship-key": "tenant.tenant-id", "relationship-value":
16                "15707428869d4c0b92e9f4b14049ea3e"}
17            ]
18          }
19        ]
20      }
21    ]
22  }

```

Fig. 3.8.PUT customer request

Once we sent all the HTTP petitions, we fed the VID module with all the different fields required so we could select them afterward from the graphic interface when creating a service instance and VNF.

These fields are the ones we use to fill the “Platform”, “OwningEntity”, and “LineOfBusiness” VID’s formularies, each one of them having a different, yet very simple, HTTP petition.

Again, some special headers are needed in order to perform such requests. The specific headers can be seen in Figure 3.9.

KEY	VALUE
Accept	application/json
Content-Type	application/json
X-FromAppId	robot-ete
X-TransactionId	robot-ete-bd65600d-8669-4903-8a14-af88203add38
USER_ID	demo

Fig. 3.9 Headers for the VID populating

The owning entity, as the name implies, is the owner of the service functions to be ultimately deployed in order to have control over the different functions in the cloud.

It's important to highlight that we populate the VID using the graphic interface like a real operator instead of using the script. Also, different user roles are required [16] [17] [18] [19].

The steps to be followed in order to create a service regardless the role are represented in Figure 3.10.

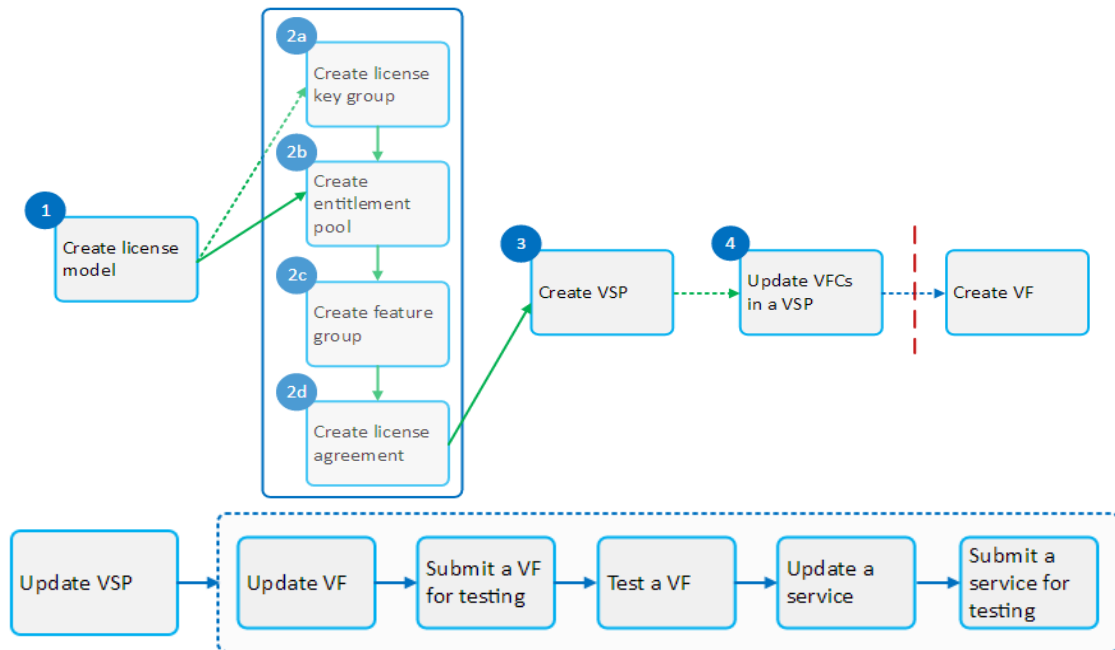


Fig. 3.10 Steps required in order to create a service

It's possible to deploy the vFW use case using a script provided by ONAP, but in order to recreate a realistic scenario, we have deployed and configured the service using mostly ONAP's web interface.

We have only used the command line interface to make HTTP petitions in order to populate ONAP's modules, namely VID and AAI, with the required information so as to be able to create, design, and instantiate network functions.

This way, by following the steps a real operator would have to perform, we have obtained more insight into how ONAP designs, configures, and deploys virtualized network functions.

A detailed list of the steps we took to deploy the service like a real operator would have done with its infrastructure can be found in the subsequent points.

3.2.1.1 Designer's steps

Services are first created by Designers since they are the ones in charge of making all the SDC resources.

First of all, we had to login in the ONAP's Portal as a Designer.



Fig. 3.11 ONAP's Portal log in page with the Designer's credentials

Then, once inside, we had to access the SDC panel to onboard the service.

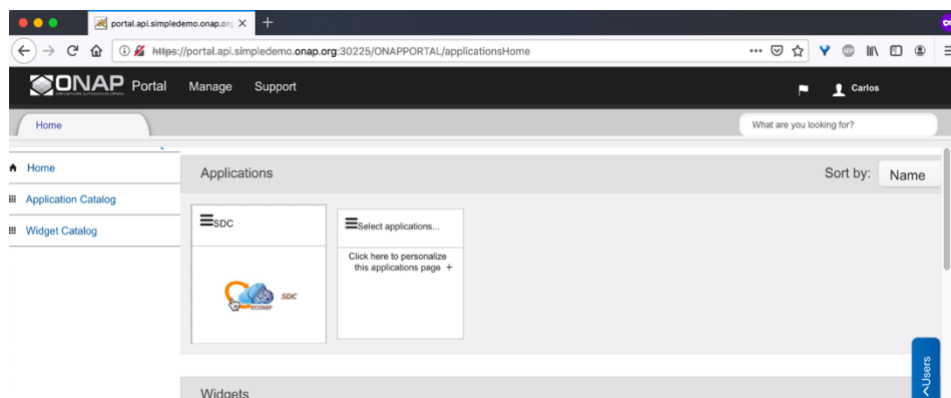


Fig. 3.12 ONAP's Portal dashboard

However, in order to onboard the service, first we had to create a new license model or VLM.

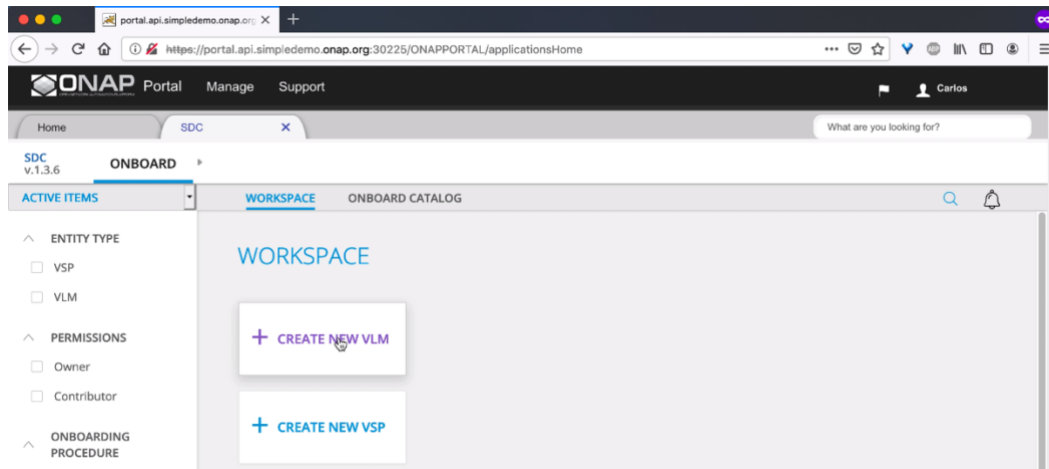


Fig. 3.13 Designer's workspace

Virtual license models are used to prevent unlicensed use and distribution of software products and services, that is to say, they deny the access to the applications for anyone who is not entitled to access them. Moreover, with virtual licenses it's possible to offer trial versions of the services and easily ensure quick and easy product renewal [20].

In order to make a VLM in ONAP, we had to create a License Agreement, an Entitlement Pool, and a Feature Group.

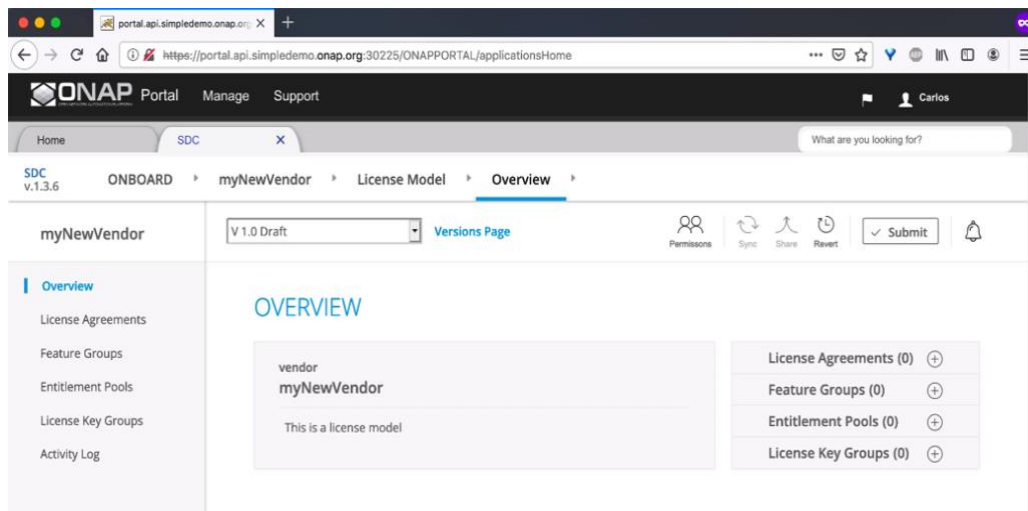


Fig. 3.14 VLM's overview menu

A License Agreement is a formal contract that gives the users the permission to use a service under a specific set of parameters. In our case, we didn't specify any particular requirements. Its creation can be seen in the Figure 3.15 [16].

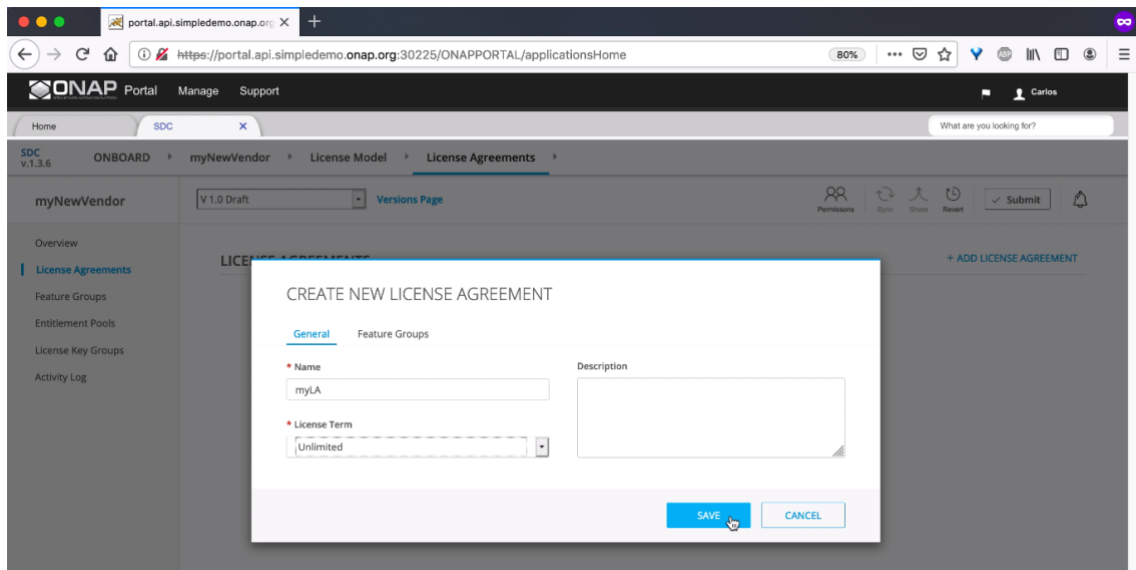


Fig. 3.15 License Agreement's creation page

An Entitlement Pool are the users or group of users who have access to a certain License Agreement. It's a pre-requisite for creating Feature Groups. The necessary form for creating it can be seen in Figure 3.16 [17].

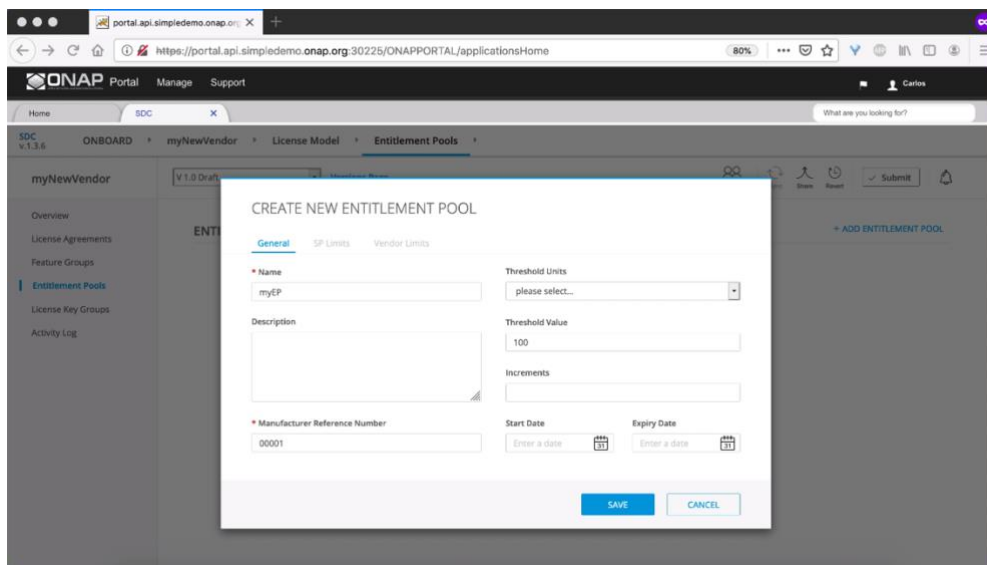


Fig. 3.16 Entitlement Pool's creation page

A Feature Group are the users within an Entitlement Pool who share the same requisites and limitations. Its creation can be seen in the Figure 3.17 [18] [19].

It's important to remark that Feature Groups are assigned to License Agreements, which means that our previously created License Agreement has to be modified.

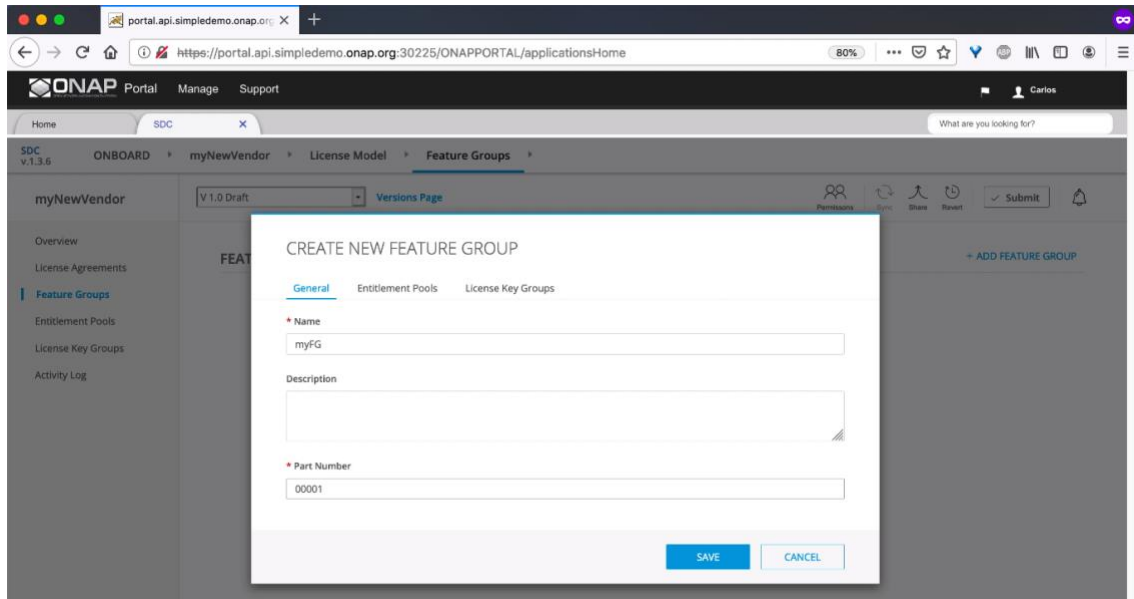


Fig. 3.17 Feature Group's creation page

Once created, we could see the new VLM inside the onboard catalog.

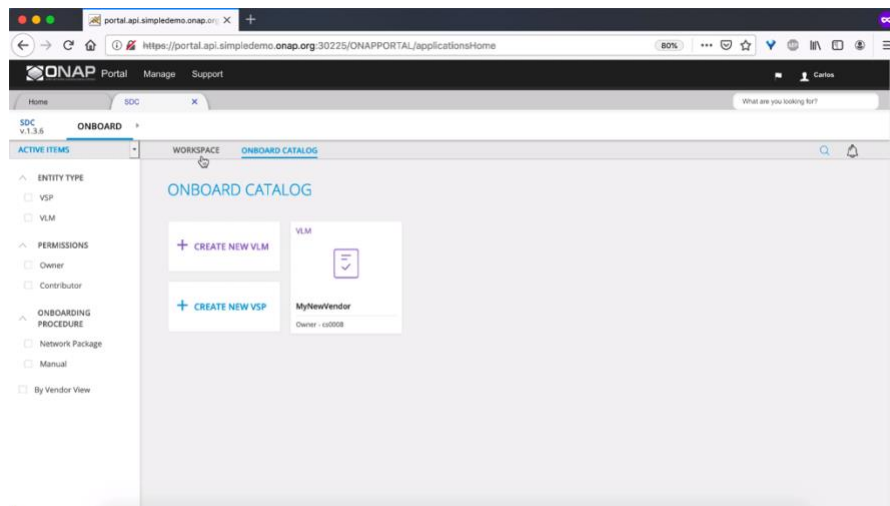


Fig. 3.18 Designer's onboard catalog overview

With the VLM properly configured, we proceeded to create a new virtual software product or VSP.

As it has been commented in the previous point, the vFW use case has 3 services, which are the packet generator (vPKG), the firewall (vFW) and the packet sink (vSNK). These functions have to be instantiated as VSP [16] [17] [18] [19].

Firstly, we created the packet sink along with the virtual firewall.

In ONAP, a VSP requires a name, a description, a VLM, an onboarding procedure, and a category.

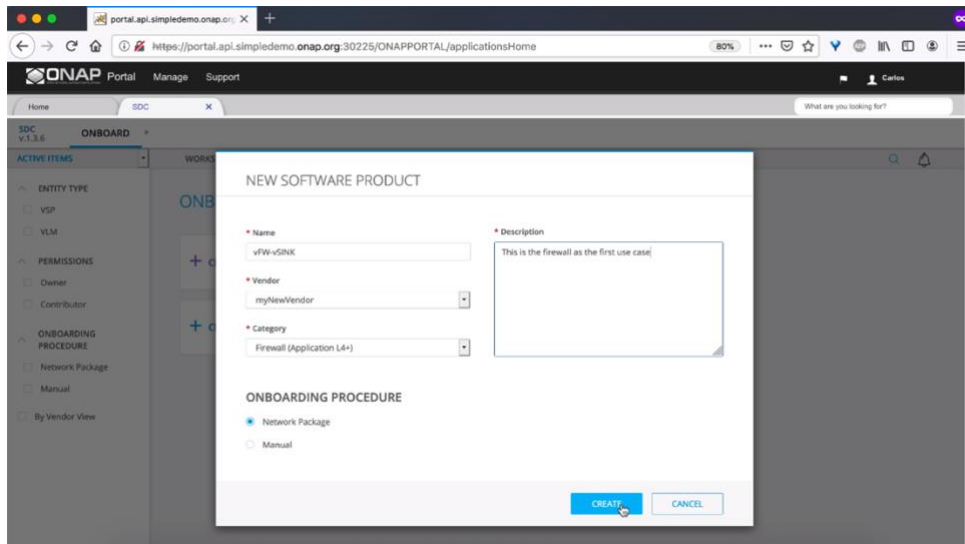


Fig. 3.19 VSP's creation page

There were many categories, but since we were implementing the vFW use case, the category had to be a firewall application so as to have it clearly identified and grouped. The onboarding procedure specifies which the functions of the VSP are. Without any procedure, the VSP would just be a blank model without any functionality [21].

Manual means implementing the functions on a console inside ONAP, and the "Network Package" option allows uploading files from the computer.

We selected Network Package, and then, we uploaded a ZIP file containing the HEAT template with the functions to be performed.

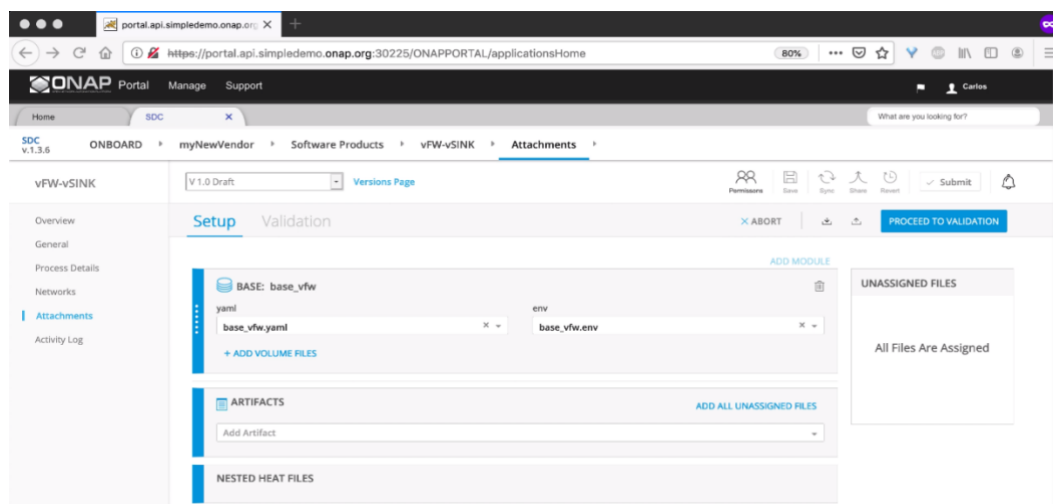


Fig. 3.20 VSP's setup page

As it can be seen in Figure 3.20, the HEAT template consists of a YAML and an ENV file.

The YAML contained the field definitions, which are given a value in the ENV file.

We didn't modify the YAML file, but in the case of the ENV file, we had to customize it with values from our OpenStack's private cloud.

With the VSP properly created and with a working template, we proceeded to import the service.

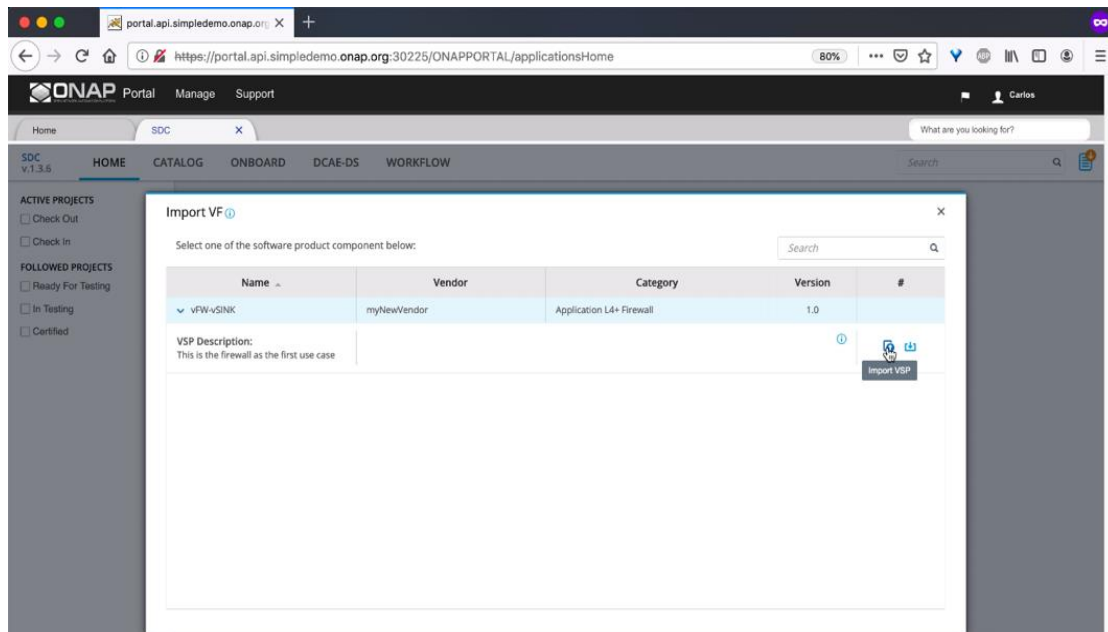


Fig. 3.21 VSP's import page

Once imported, we created the service with the click of a button.

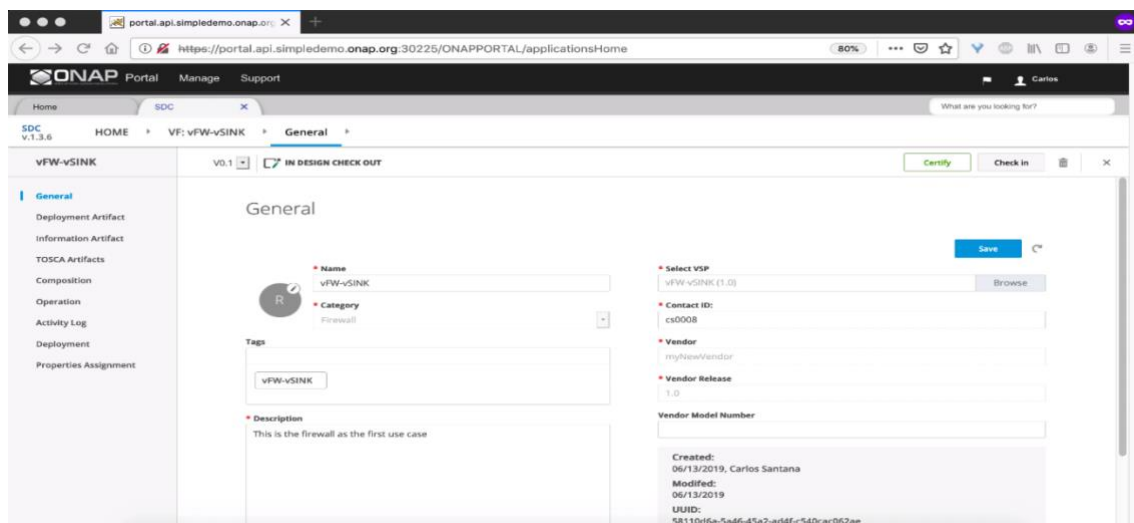


Fig. 3.22 VSP's general overview

And then, we certified the service also just by clicking a button.

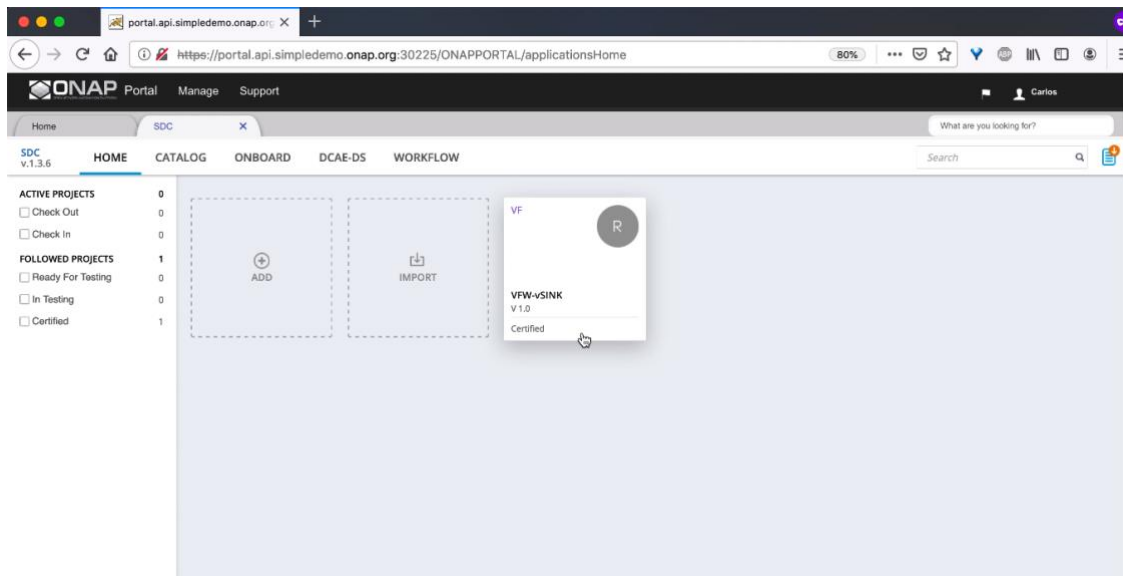


Fig. 3.23 Designer’s dashboard with the vSNK VSP

With the vSNK created, we repeated the same steps but for the vPKG, which had its own HEAT template.

In the end, the scenario consisted of only 2 VSP because the firewall capabilities were embedded within the vSNK functions in order to make the service more light-weight.

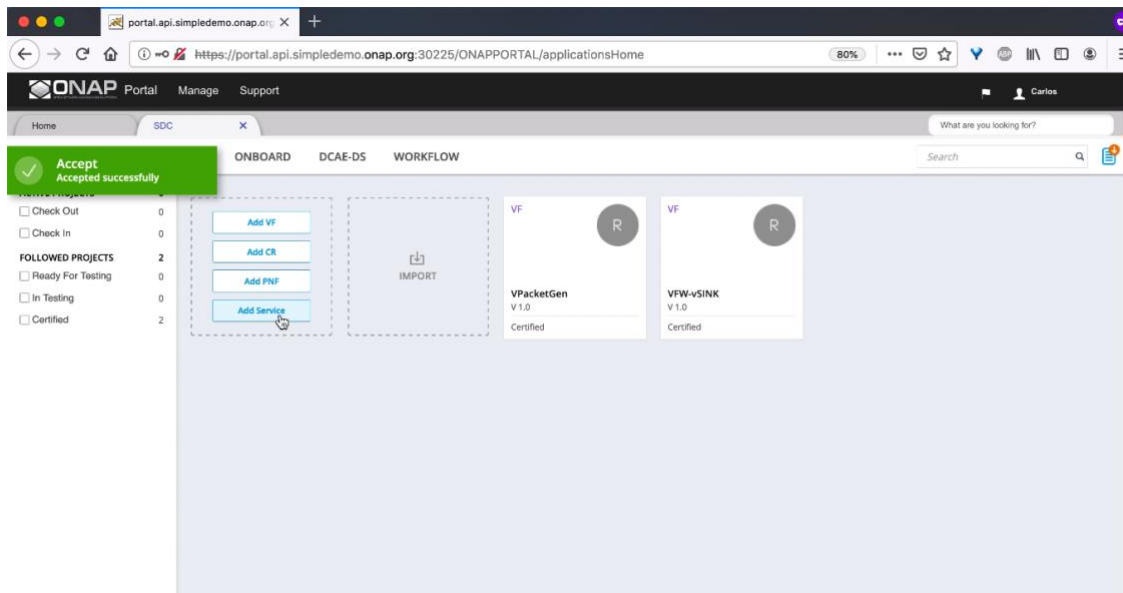


Fig. 3.24 Designer’s dashboard with both VSP

Afterward, with both VSP certified, our next step was to create the service, shown in Figure 3.25.

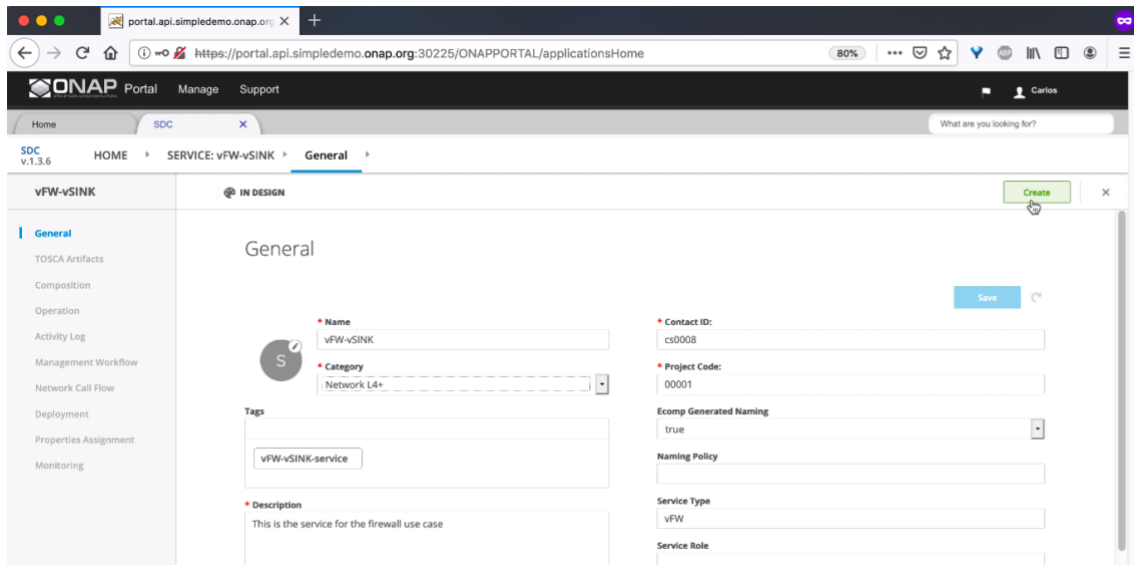


Fig. 3.25 Service's general overview page

After having created the service, inside its workspace, we had to create the composition of the service with the previous VSP.

In order to do that, we only had to drag and drop the functions to the workspace, as it can be seen in Figure 3.26.

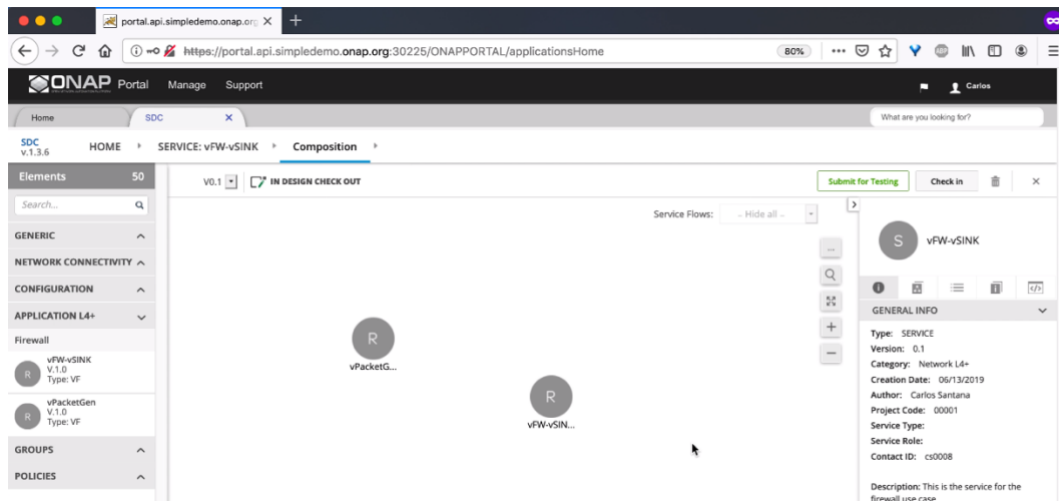


Fig. 3.26 Service's composition page

Afterward, we submitted the service for testing by clicking one button.

3.2.1.2 Tester's steps

To make sure the service worked as expected, we had to change of role, and to do that, we had to log out and then log in again as a Tester.

As a reminder, the Tester is the user responsible for testing a proposed distribution.

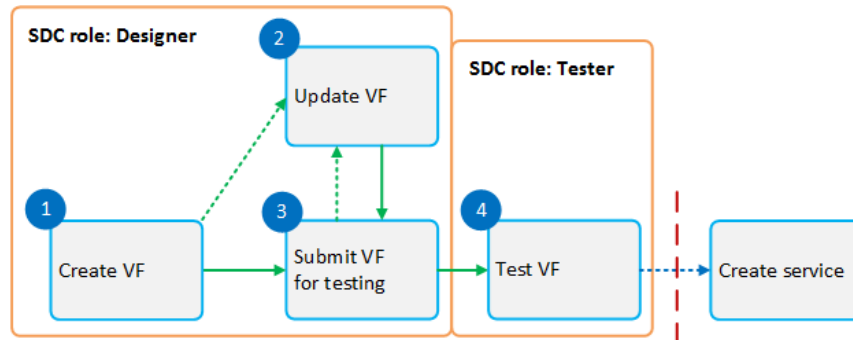


Fig. 3.27 Tester's steps in order to test a service

As it's shown in Figure 3.28, the Tester can see in its dashboard the services that have to be tested. As expected, our service is the only one that appears.

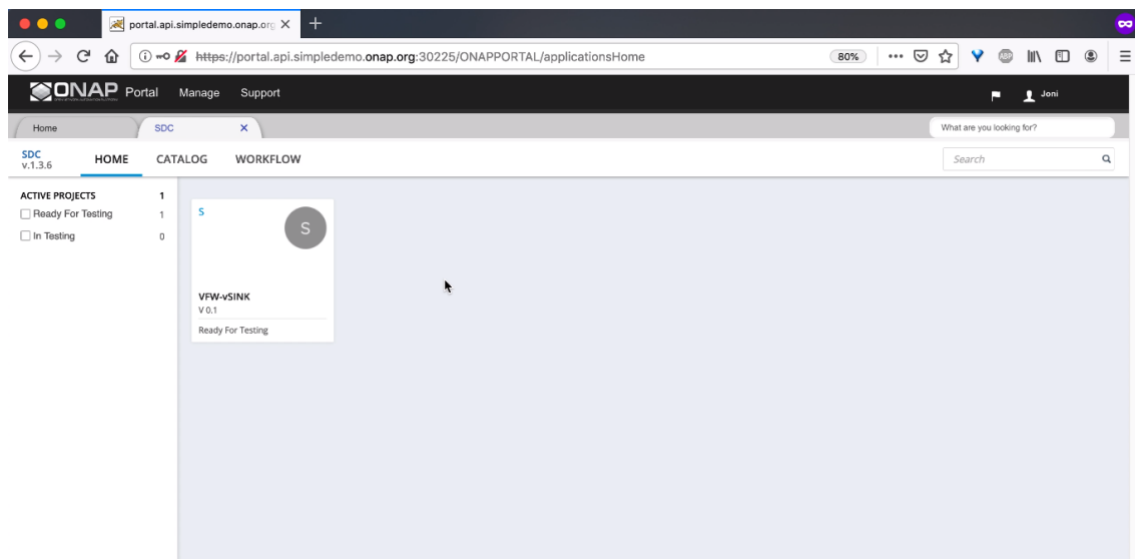


Fig. 3.28 Tester's dashboard with the vFW service

3.2.1.3 Governor and Operator's steps

Once tested and submitted, in order to deploy the service, we needed approval from the Governor.

Again, we had to change the role by logging out and then logging in again in the Portal's page since using multiple concurrent connections is highly unadvised.

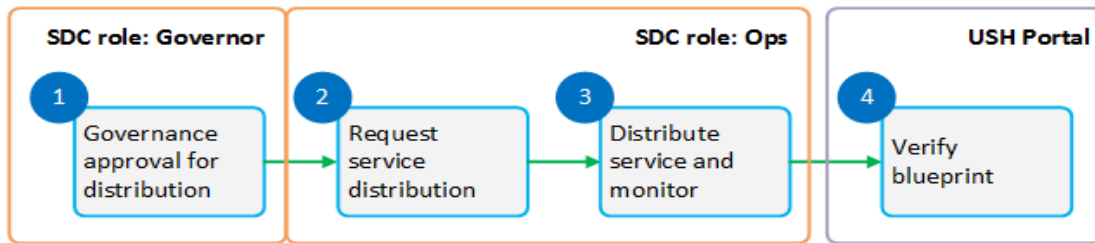


Fig. 3.29 Governor and Operator's steps to approve and distribute a service

Similarly to the Tester, the Governor can see in its dashboard the services that have to be approved. Again, expectedly, our service is the only one that appears.

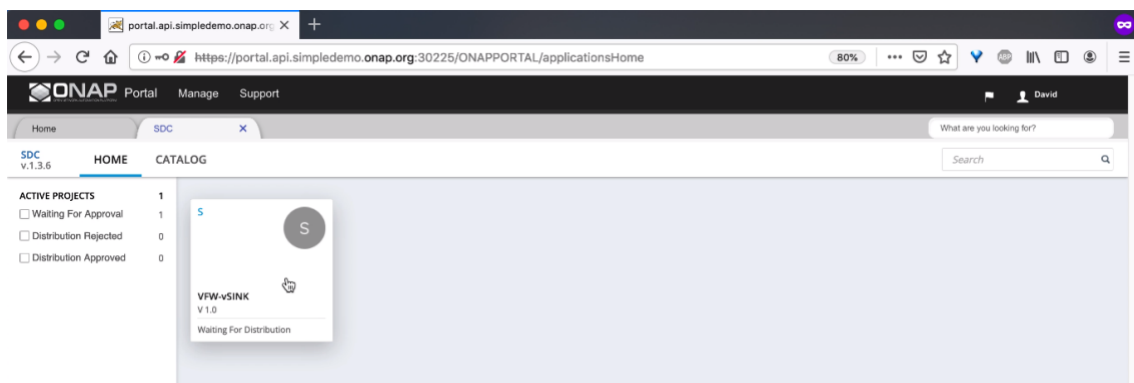


Fig. 3.30 Governor's dashboard with the tested vFW service

Finally, in order to distribute the approved service, we needed to be an Operator. One more time, we had to switch of account.

Yet again, the Operator is able to see the services that need to be distributed in its dashboard, and once more, the only service that appears is ours.

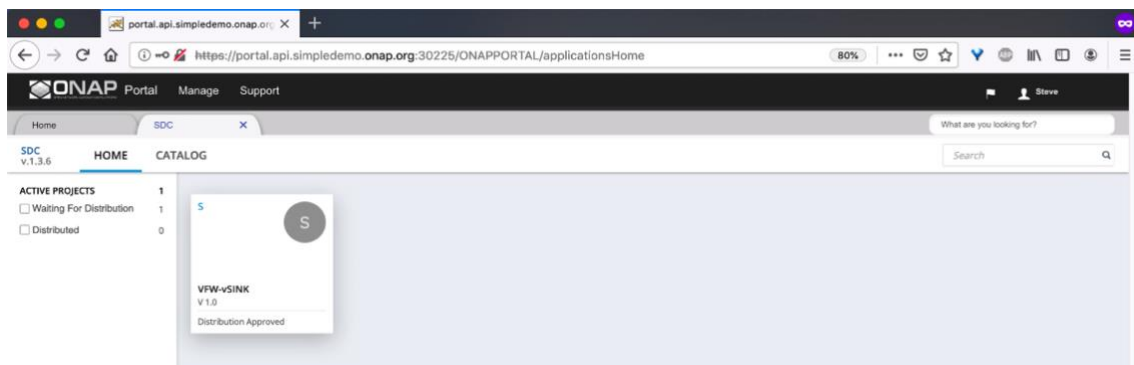


Fig. 3.31 Operator's dashboard with the approved vFW service

With the click of a button, we distributed the service, making it available to be used.

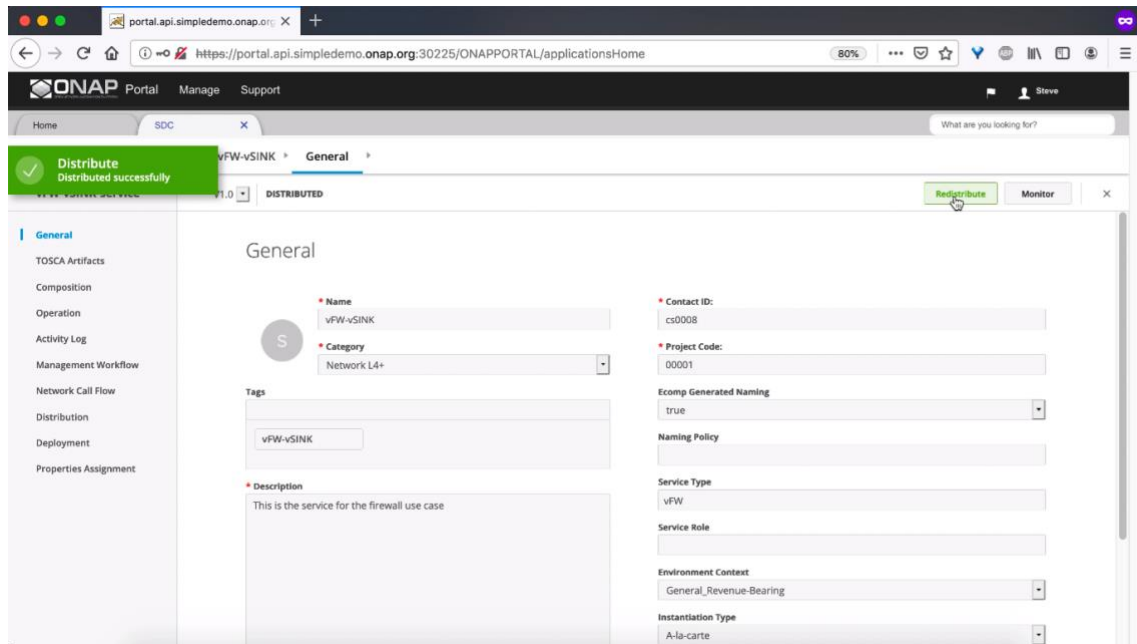


Fig. 3.32 Operator's distribution page.

Subsequently, in order to see the distribution, we clicked on the Operator's distribution tab, where we saw that the service we had created had been successfully distributed without errors.

This distribution is shown in Figure 3.33.

Distribution

Distribution ID	User id	Time(UTC)	Deployed
7fb30f9d-6124-44ec-a2f7-bdda752aac2	Oper P(top0001)	06/20/2019 1:33PM	Deployed
Total Artifacts: 60 Notified: 15 Downloaded: 10 Deployed: 10 Not Notified: 45 Deploy Errors: 0 Download Errors: 0			
SO-COpenSource-Env11	12	Notified: 7 Downloaded: 7 Deployed: 7	Not Notified: 5 Deploy Errors: 0 Download Errors: 0
policy-id	12	Notified: 3 Downloaded: 1 Deployed: 1	Not Notified: 9 Deploy Errors: 0 Download Errors: 0
sd-cOpenSource-Env11-sdnc-dockero	12	Notified: 1 Downloaded: 1 Deployed: 1	Not Notified: 11 Deploy Errors: 0 Download Errors: 0
aal-ml	12	Notified: 1 Downloaded: 1 Deployed: 1	Not Notified: 11 Deploy Errors: 0 Download Errors: 0
clamp	12	Notified: 3 Downloaded: 0 Deployed: 0	Not Notified: 9 Deploy Errors: 0 Download Errors: 0

Fig. 3.33 Operator's distribution tab

3.2.2. Virtual Firewall instantiation

With the service created and correctly distributed, the next step was to instantiate it [22] [23].

To do so, with the Demo role, we went to the VID's dashboard, where we found the service we had created and properly distributed, as it's shown in figure 3.34.

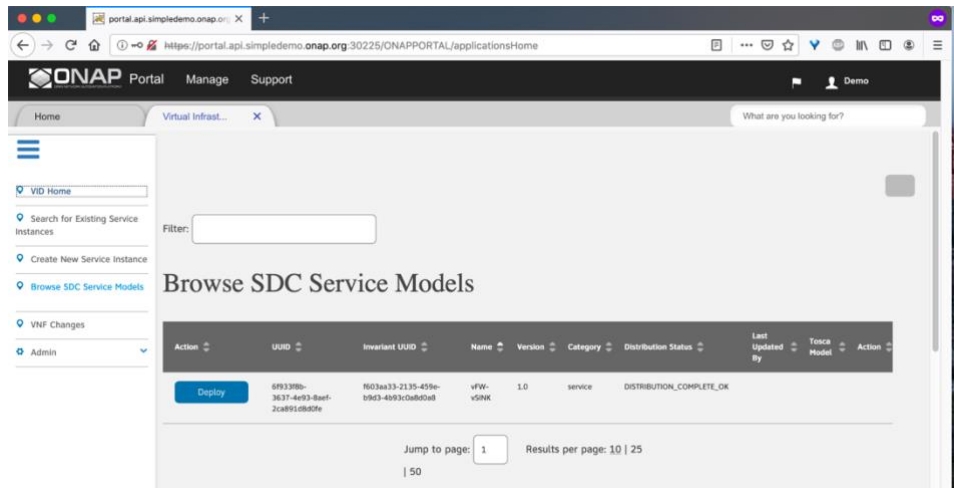


Fig. 3.34 SDC's dashboard with the vFW service in a distributed state

Since the service was in a distributed state and without errors, otherwise ONAP doesn't let you continue, we proceeded to deploy the service using VID's web user interface.

Figure 3.35 shows the formulary we had to fill in order to create the service instance.

To do so, we had to specify the instance's name, the subscriber's name, the service type, the project to which it belongs to and the owning entity.

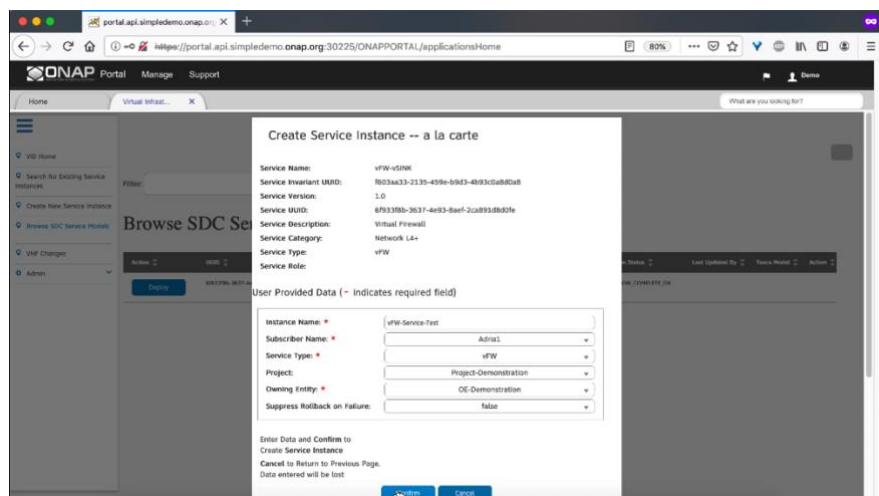


Fig. 3.35 VID's service instance creation

Figure 3.36 shows how the instantiation of the service was a success.

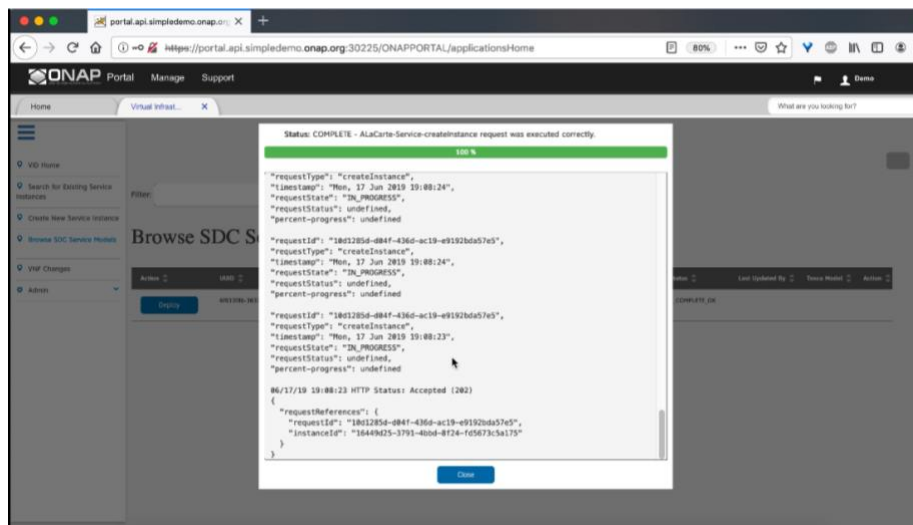


Fig. 3.36 Successful service instantiation

Then, we proceed to instantiate the service's functions, starting with the packet sink. Again, we had to fill another questionnaire, as it can be seen in Figure 3.37.

This time, we had to put the instance's name, the product's family, the cloud's region, the tenant, the line of business, and the platform.

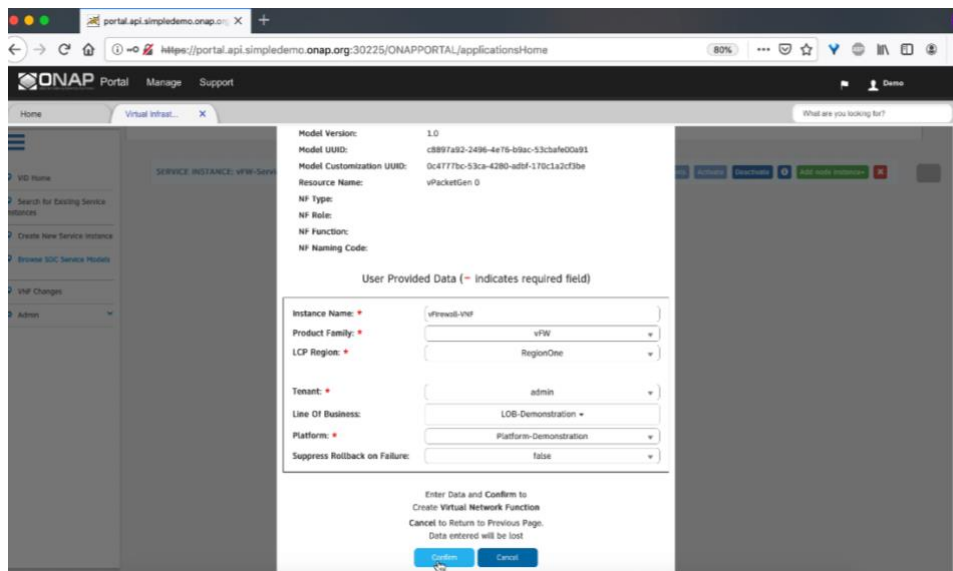


Fig. 3.37 VID's service function instantiation

It's very important to highlight that the values specified in the formularies are the same values we had used in the creation of the models.

Also, in order to instantiate the service's functions, we had to create an owning entity by sending an HTTP petition to the VID's module with the following body:

```
{
  "options": ["Test-Entity"]
}
```

Figure 3.38 shows how the instantiation of the service's function was a success.

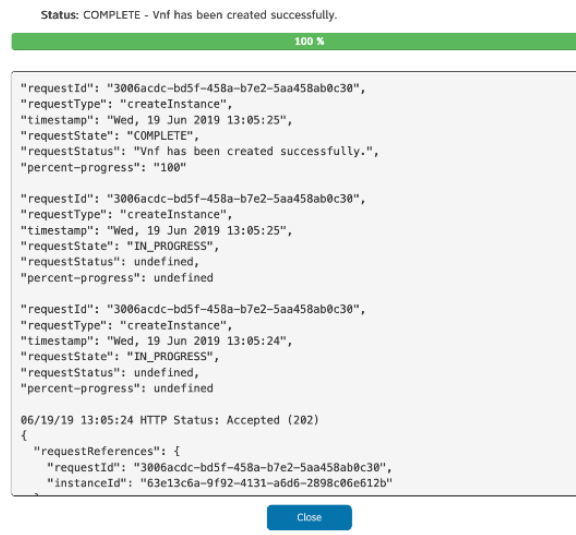


Fig. 3.38 Successful service's function instantiation

Once we had correctly instantiated the packet sink, we repeated the same steps for the packet generator.

As it can be seen in Figure 3.39, both functions had correctly been instantiated.

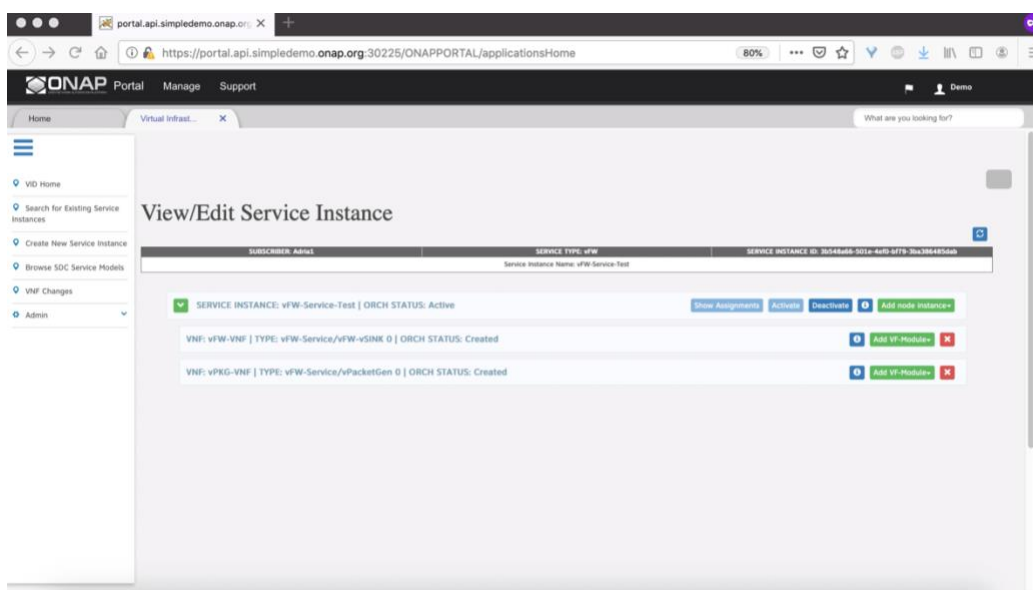


Fig. 3.39 VID's dashboard with the instantiated service's functions

3.2.3. Virtual Firewall deployment

In order to deploy the vFW service, we had to access first the SDNC module, only reachable from within the internal private cloud network. We have to access the SDNC module in order to pre-load the VNFs, adding specific information required by the cloud where the service is going to be run [19].

With the VPN explained in the previous chapter up and running, we accessed the SDNC's portal, as it's shown in Figure 3.40.

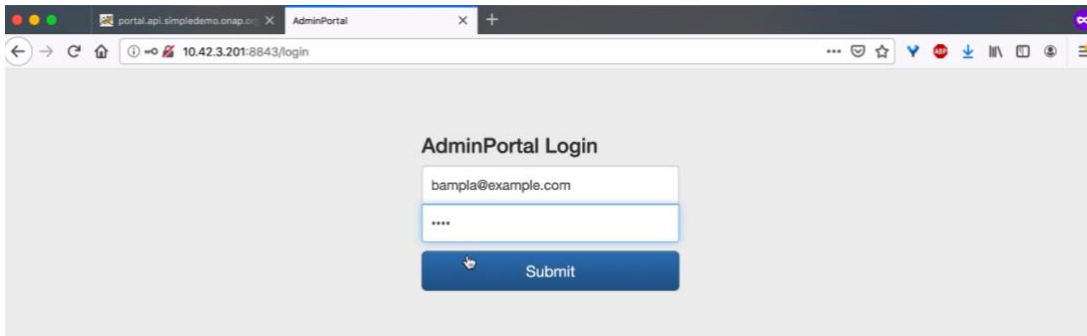


Fig. 3.40 SDNC's login page

After that, with the same information we used in the instantiation of the services, which can easily be retrieved from the VID module, we added two VNF profiles, one for the packet generator and one for the sink, as it can be seen in Figure 3.41.

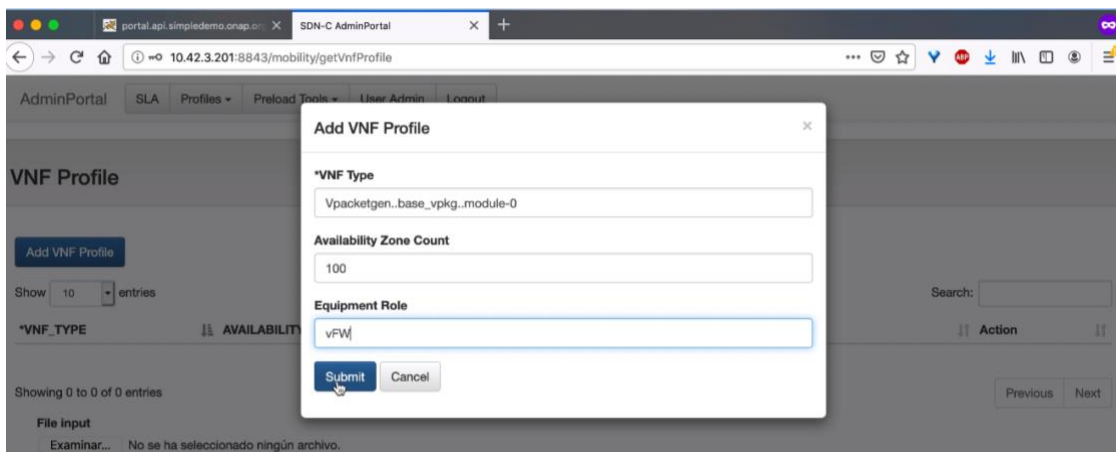


Fig. 3.41 VNF profile addition from the SDNC module

For both cases, we had to specify the VNF type, the availability zone count, and the role of the equipment. The type parameter is extracted from the VID module and it's the one we previously had created.

Figure 3.42 shows the correct creation of both profiles.

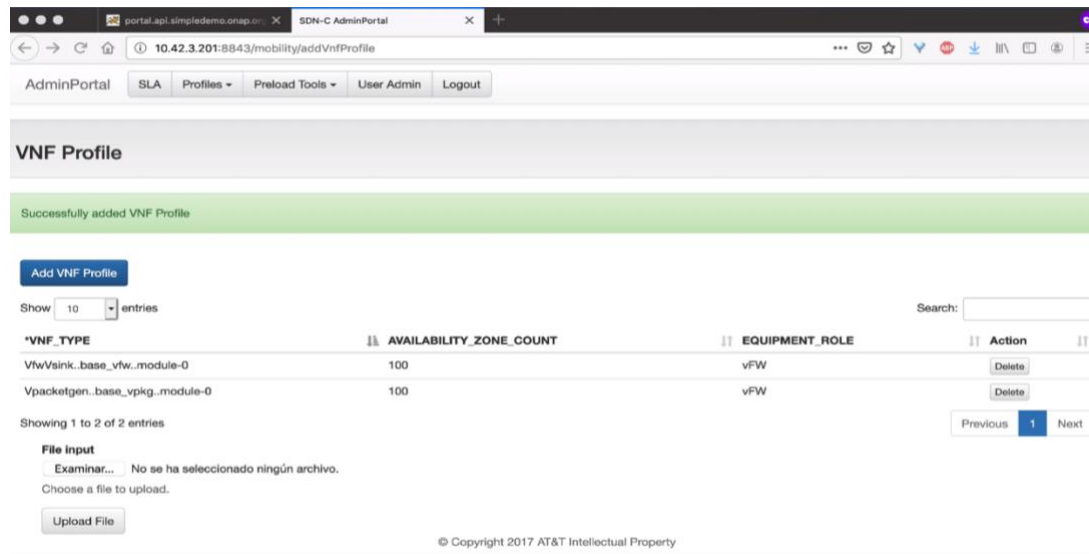


Fig. 3.42 VNF profile's dashboard

It's important to highlight that the SDNC module is, in fact, an OpenDaylight controller with its own API.

Then, with both profiles created, we proceeded to preload the SDNC's models by using Swagger, an open-source software framework used to design, build, document, and consume RESful Web services, as it's shown in Figure 3.43.

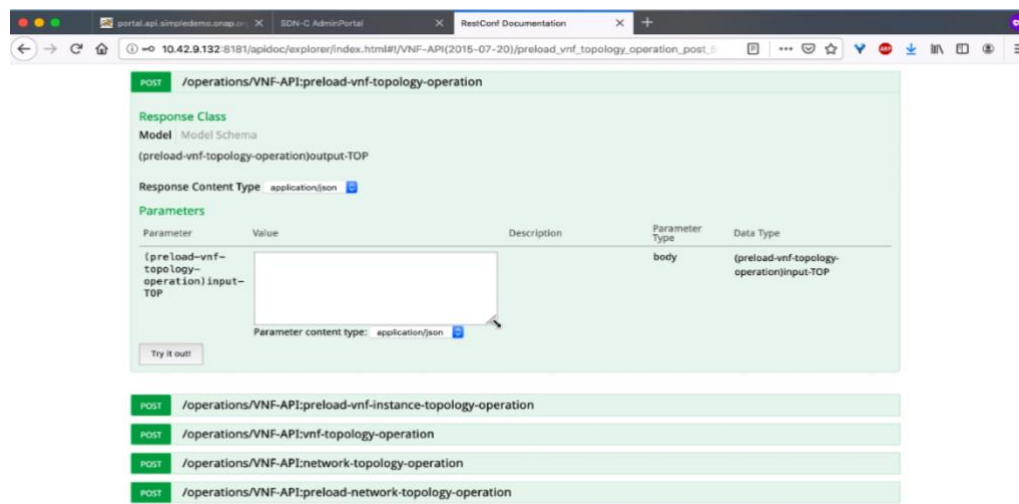


Fig. 3.43 SDNC's API (Swagger's representation)

We had to send the petition twice, one for the packet generator and another one for the sink.

In both cases, in the body of the HTTP petition we had to define the VNF's name and the VNF's type with the same information as in the VID's service instances.

```
"input": {
  "vnf-topology-information": {
    "vnf-topology-identifier": {
      "service-type": "7175e497-5768-4dc3-8f65-9d2d6a8bc3b0",
      "vnf-name": "vFirewall-Test-1",
      "vnf-type": "VfwVsink..base_vfw..module-0",
      "generic-vnf-name": "vFW-VNF-Testing",
      "generic-vnf-type": "vFW-vSINK 0"
    },

```

Also, we had to feed the HTTP's request body with the image type, the virtual machine's flavor, the private and public network identifiers, the private and public IP's of the cluster, the service's name, the service's ID, the service's module, the DCAE collector's IP and port, the artifact repository, and the public SSH key.

Names and IDs must be unique.

```
{
  "vnf-parameter-name": "public_net_id",
  "vnf-parameter-value": "971040b2-7059-49dc-b220-4fab50cb2ad4"
},
{
  "vnf-parameter-name": "onap_private_net_id",
  "vnf-parameter-value": "oam_network_erMV"
},
{
  "vnf-parameter-name": "onap_private_subnet_id",
  "vnf-parameter-value": "oam_network_erMV"
},
{
  "vnf-parameter-name": "vfw_name_0",
  "vnf-parameter-value": "vFW-VNF-Testing"
},
},
```

As it has been aforesaid, we sent the petition twice, and in both cases we got a correct response, as it can be seen in Figure 3.44.



Fig. 3.44 Successful callback from SDNC's API

With both VNF service instances preloaded, we went back to the VID's interface, where we instantiated the function as it's shown in Figure 3.45.

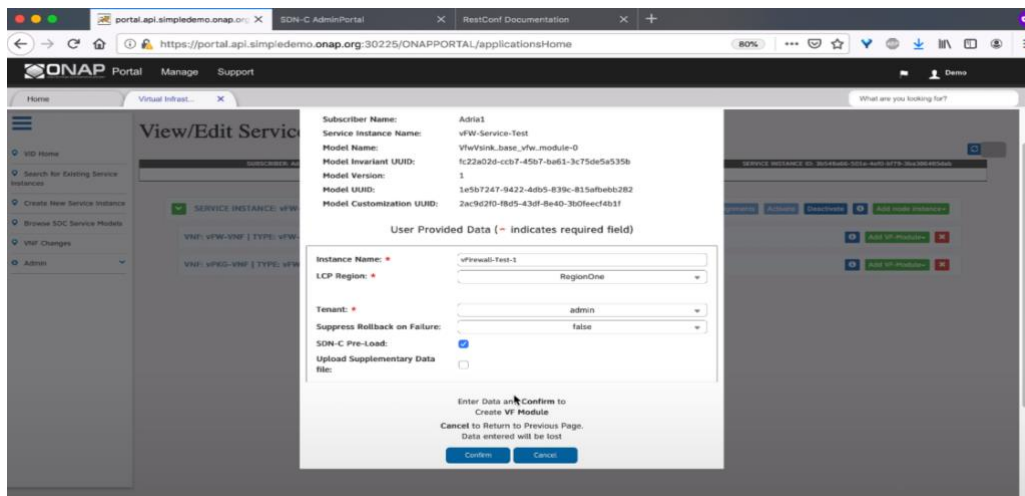


Fig. 3.45 VID's VF-module deployment

It's important to remark that the SDNC's pre-install checkbox has to be checked in order to use all the information we had pre-loaded in the Swagger screen.

3.2.4. Virtual Firewall issues

The next step was to make a VID deployment of the modules, but due to incompatibility issues between ONAP's and OpenStack's Keystone versions, we weren't able to proceed with the instantiation of the service.

To know more about this error, you can find more information in the paper "Containerizing ONAP using Kubernetes and Docker", the project in which the infrastructure used for this ONAP is deployed.

However, in order to help any operator or individual wishing to continue with the installation and deployment of services, we have included the steps to follow and the results expected in Annex II, this steps are a summary of all the relevant information collected from the different references to the vFW use case inside the ONAP official documentation.

CONCLUSIONS

This project has explained step by step how a real operator would be able to instantiate and deploy virtualized network functions in a simple and controlled way thanks to ONAP's platform.

It's imperative to remark the importance of having been able to deploy virtualized services and functions without having had to use ONAP's scripts, like a real operator would have done in a real production scenario, thus achieving a much more realist workflow and use case scenario.

Another important point to comment are the different errors we have come across during the development of this project. Most of them made us push back up to the deployment stages in order to solve them.

Moreover, due to the lack of proper documentation from ONAP regarding these issues, we've had to ultimately solve them by trial and error techniques, turning this into a critical point for the ONAP development team.

Related to the environmental aspects of the project, a direct consequence of this paper are the potential environmental improvements due to the virtualization of services and infrastructure. This virtualization reduces the electricity consumption of operators, ultimately reducing the environmental impact of their networks.

This project is very explanatory as is, but in the future, it would be very interesting to further check ONAP's capabilities by tuning the virtual firewall use case or even deploying harder and more realistic scenarios, such as the VoLTE use case.

With the VoLTE scenario, we would have multiple VNFs related between them, a more complete and interesting use case we haven't been able to deploy due to the issues faced regarding the keystone versions. Our servers were deployed with a too much new OpenStack version for ONAP's standards.

With a less restricted scenario, deploying the vCPE use case would have been a possibility.

As a final point, as it has been commented at the beginning of the essay, all the results have been sent to JITEL, which will ultimately result in the publication of a paper explaining our work. That will draw more attention to ONAP, rendering this project as a complete success.

BIBLIOGRAPHY

[1] Charles David Graziano. A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project, <https://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=3243&context=etd> [Accessed: 15/05/2019]

[2] The Linux Foundation Projects. ONAP Platform, <https://www.onap.org/platform-2> [Accessed: 15/05/2019]

[3] Cloudify. What is ONAP and what does it mean for you? <https://cloudify.co/onap/what-is-onap/> [Accessed: 15/05/2019]

[4] ONAP. Official ONAP webpage, <https://www.onap.org/> [Accessed: 17/06/2019]

[5] SearchNetworking. What is ONAP, and how does it tie into NFV architecture?, <https://searchnetworking.techtarget.com/tip/What-is-ONAP-and-how-does-it-tie-into-NFV-architecture> [Accessed: 17/06/2019]

[6] Metaswitch. All Abuzz about ONAP <https://www.metaswitch.com/blog/all-abuzz-about-onap> [Accessed: 17/05/2019]

[7] ONAP. Official ONAP Wiki <https://wiki.onap.org/display/DW/Tutorial%3A+Accessing+the+ONAP+Portal> [Accessed: 18/06/2019]

[8] ONAP. Official ONAP Wiki <https://wiki.onap.org/display/DW/Users> [Accessed: 18/06/2019]

[9] ONAP. Official ONAP Wiki <https://wiki.onap.org/display/DW/ONAP+Cloud+Native+Multi+tenancy+proposal> [Accessed: 18/06/2019]

[10] ONAP. Official ONAP Wiki <https://wiki.onap.org/display/DW/Developer+Wiki> [Accessed: 19/06/2019]

[11] ONAP. Official ONAP Wiki <https://wiki.onap.org/display/DW/ONAP+Portal> [Accessed: 28/06/2019]

[12] ONAP. Official ONAP Wiki <https://wiki.onap.org/pages/viewpage.action?pageId=4719246> [Accessed: 28/06/2019]

[13] OpenVPN. Official OpenVPN webpage <https://openvpn.net/> [Accessed: 24/06/2019]

- [14] ONAP. Official ONAP Documentation
https://docs.onap.org/en/casablanca/submodules/integration.git/docs/docs_vfw.html [Accessed: 24/06/2019]
- [15] ONAP. Official ONAP Wiki
<https://wiki.onap.org/display/DW/vFW+CDS+Casablanca>
[Accessed: 24/06/2019]
- [16] ONAP. Official ONAP Wiki
<https://wiki.onap.org/display/DW/ONAP+Beijing%3A+Understanding+the+vFW+CL+use-case+mechanism> [Accessed: 25/06/2019]
- [17] AARNA Networks. Official AARNA Networks webpage
<https://www.aarnanetworks.com/single-post/2018/06/04/A-Cost-Effective-Way-to-Try-the-ONAP-vFW-Blueprint> [Accessed: 25/06/2019]
- [18] ONAP. Official ONAP Wiki
<https://wiki.onap.org/display/DW/vFWCL+instantiation%2C+testing%2C+and+d+ebuging> [Accessed: 26/06/2019]
- [19] ONAP. Official ONAP Wiki
[https://wiki.onap.org/display/DW/vFW+Closed+Loop+step-by-step#vFWClosedLoopstep-by-step-CreateserviceinstanceandthenVNFinstanceinVID\(https://wiki.onap.org/display/DW/Tutorial+vIMS%3A+VID+Instantiate+the+VNF\)](https://wiki.onap.org/display/DW/vFW+Closed+Loop+step-by-step#vFWClosedLoopstep-by-step-CreateserviceinstanceandthenVNFinstanceinVID(https://wiki.onap.org/display/DW/Tutorial+vIMS%3A+VID+Instantiate+the+VNF)) [Accessed: 28/06/2019]
- [20] Gemalto. Secure Software Licensing <https://sentinel.gemalto.com/software-monetization/secure-software-licensing/>
[Accessed: 26/06/2019]
- [21] ONAP. Official ONAP Wiki
<https://wiki.onap.org/display/DW/Resource+Onboarding#ResourceOnboarding-CEP> [Accessed: 26/06/2019]
- [22] ONAP. Official ONAP Wiki
https://wiki.onap.org/display/DW/Tutorial_vIMS%3A+Create+AAI+cloud+account [Accessed: 27/06/2019]
- [23] ONAP. Official ONAP Wiki
<https://wiki.onap.org/pages/viewpage.action?pageId=16006682>
[Accessed: 27/06/2019]
- [24] ONAP. Official ONAP Documentation
<https://onap.readthedocs.io/en/latest/submodules/aai/aai-common.git/docs/platform/architecture.html> [Accessed: 28/06/2019]
- [25] ONAP. Official ONAP Wiki
<https://wiki.onap.org/pages/viewpage.action?pageId=1015837>
[Accessed: 28/06/2019]

[26] ONAP. Official ONAP Wiki Link:

<https://wiki.onap.org/pages/viewpage.action?pageId=3247281>

[Accessed: 28/06/2019]

[27] ONAP. Official ONAP Wiki Link:

<https://wiki.onap.org/pages/viewpage.action?pageId=1015831>

[Accessed: 28/06/2019]

[28] ONAP. Official ONAP Wiki Link:

<https://wiki.onap.org/pages/viewpage.action?pageId=3247130>

[Accessed: 28/06/2019]

[29] ONAP. Official ONAP Wiki Link: <https://wiki.onap.org/display/DW/VID>

[Accessed: 28/06/2019]

[30] ONAP. Official ONAP Wiki Link

<https://wiki.onap.org/display/DW/Using+Robot+Command+Line>

[Accessed: 28/06/2019]

[31] ONAP. Official ONAP Wiki Link

<https://onap.readthedocs.io/en/latest/submodules/appc.git/docs/index.html>

[Accessed: 28/06/2019]

[32] ONAP. Official ONAP Wiki Link

<https://wiki.onap.org/display/DW/The+ONAP+Policy+Framework#TheONAPPolicyFramework-1.Overview> [Accessed: 28/06/2019]

[33] ONAP. Official ONAP Wiki Link

https://wiki.onap.org/display/DW/Running+the+ONAP+Demos?preview=/1015891/16010290/vFW_closed_loop.mp4 [Accessed: 28/06/2019]



ANNEX

TITLE: Developing and Deploying NFV solutions with OpenStack, Kubernetes and Docker

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Adrià Martí Luque

ADVISOR: Jesus Alcober

DATE: July, 10th 2019

ANNEX I: ONAP MODULES

I.1. Portal

The ONAP Portal platform integrates the different ONAP applications into a central core. This platform provides common management services and connectivity, while the applications run separately [11].

From the Portal platform, users can access applications. Also, administrators can manage applications, widgets, and user access. Portal is a basic element of the ONAP's architecture. Every single ONAP use case needs Portal in order to work properly [11].

Its architecture can be seen in Figure 1.

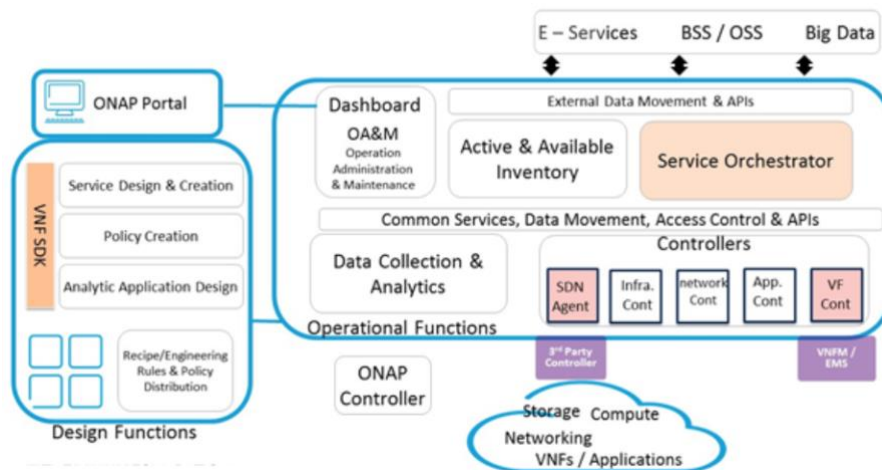


Fig. 1 Portal's architecture

I.2. AAI

As it has already been mentioned, the Active & Available Inventory, or AAI, provides ONAP with real-time information of the system's resources, services, products and relationships along with its logically centralized view of inventory data, taking in updates from orchestrators, controllers, and assurance systems [24].

To sum up, the AAI is where the data converge, where the pictures come together, and where the ONAP actor systems ask questions so they can make their decisions [24].

Its architecture can be seen in Figure 2.

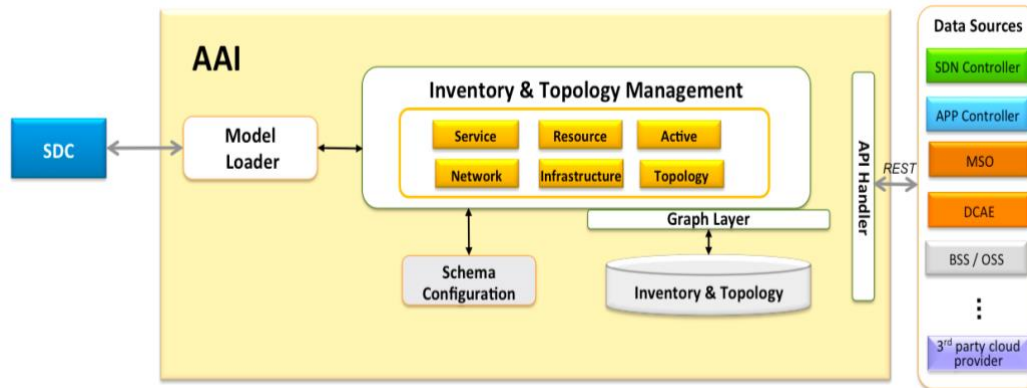


Fig. 2 AAI's architecture

I.3. SO

The Service Orchestrator, or SO, provides the highest level of service orchestration in the ONAP architecture. Its architecture can be seen in Figure 3 [12].

It's currently implemented via Business Process Model and Notation flows (BPMN) that operate on distributed models that describe the services, associated virtual network functions and other resource components [12].

The SO's final goal is to enhance ONAP's overall orchestration capabilities by aligning and integrating its imperative workflows with a TOSCA-based declarative execution environment, chief for deploying and managing virtualized services in the cloud [12].

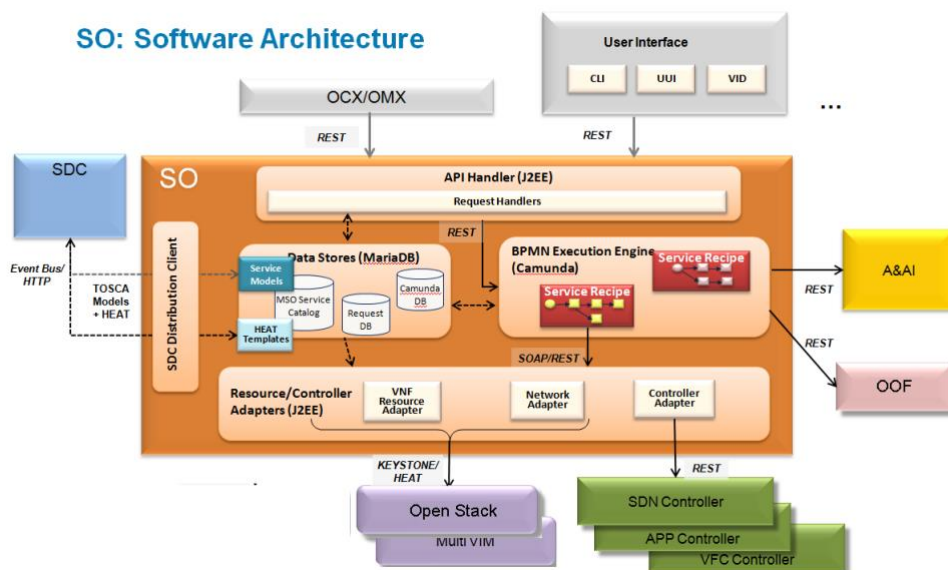


Fig. 3 SO's architecture

I.4. SDC

The Service Design and Creation, or SDC, is a visual modeling and design tool used to create internal metadata that describes assets consumed by all ONAP components, both at design-time and run-time [25].

The SDC manages the content of a catalog and logically assembles the items of the selected catalog to completely define how and when VNFs are realized in a target environment [25].

A complete virtual assembly of specific catalog items, together with selected workflows and instance configuration data, completely defines how the deployment, activation and life-cycle management of VNFs are accomplished [25].

Selected sub-assemblies may also be represented in the catalog and may be combined with other catalog items, including other sub-assemblies [25].

Without SDC it's impossible to implement any VNF. Its architecture can be seen in Figure 4.

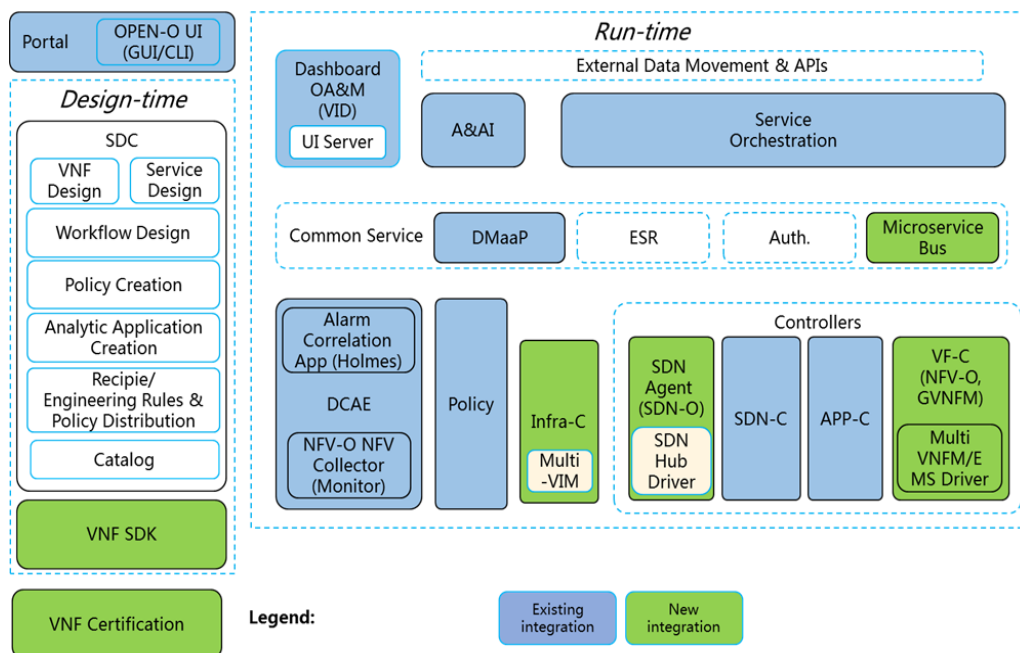


Fig. 4 SDC's architecture

I.5. SDNC

The Software Defined Network Controller, or SDNC, is a module that provides a global network controller built on the Common Controller Framework, which is responsible of managing, assigning and provisioning network resources [26].

As a full-scope controller, the SDNC module is intended to run as one logical instance per enterprise, with potentially multiple geographically-diverse virtual machines and/or containers in clusters to provide high availability [26].

Its architecture can be seen in Figure 5.

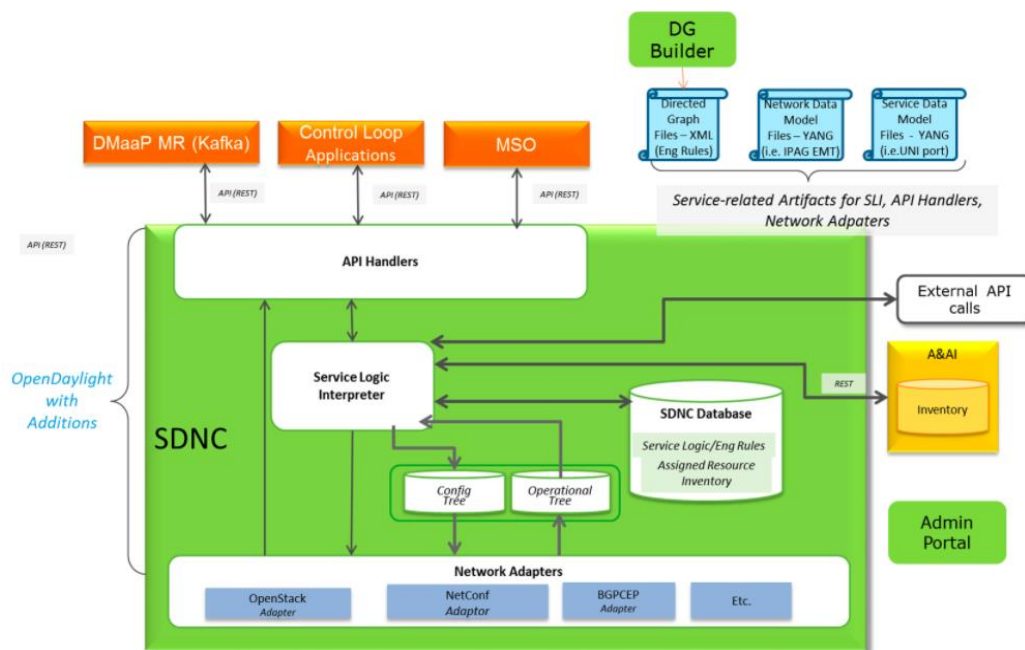


Fig. 5 SDNC's architecture

I.6. DCAE

DCAE, which stands for Data Collection, Analytics, and Events, is a module that gathers performance, usage, and configuration data from the managed virtual environments. This data is then sent to various analytic applications, and if anomalies or significant events are detected, they trigger specific actions [27].

Mainly, the primary functions of the DCAE module are to collect, ingest, transform and store data for analysis and provide a framework for development of analytics [27].

In addition, DCAE also makes the data and events available for higher-level correlation by business and operations activities, including business support systems (BSS) and operational support systems (OSS) [27].

Usage and other event processing applications can be created in the DCAE environment and be related to external BSSs or OSSs. Its architecture can be seen in Figure 6 [27].

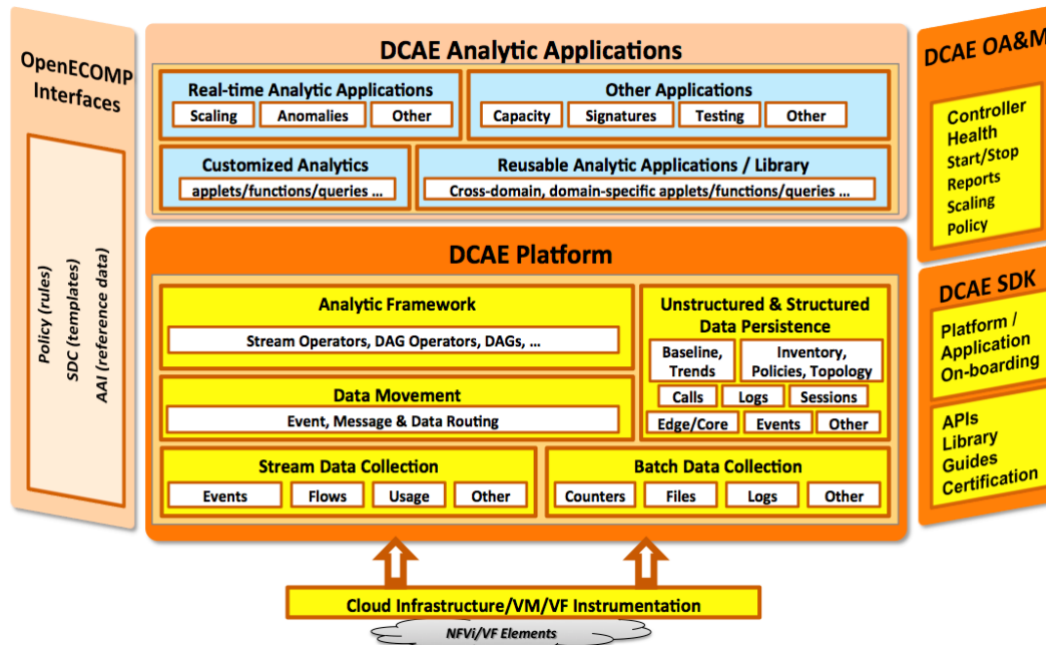


Fig. 6 DCAE's architecture

I.7. DMAAP

The Data Movement as a Platform, or DMaaP, is, as the name implies, a platform for high performing and cost-effective data movement services. It can transport and process data from any source to any target with the format, quality, security, and concurrency required to satisfy both business and customer needs [28].

DMaaP has 3 main functions [28]:

- **Data Filtering:** Data preprocessing at the edge via analytics and compression to reduce the size of the data needed to be sent and processed.
- **Data Transport:** Transport of data intra and inter data centers. Provides the ability to move data from any system to any system with minimal latency and guaranteed delivery.
- **Data Processing:** Low latency and high throughput data transformation, aggregation, and analysis. The processing is elastically scalable and fault-tolerant across data centers.

Its architecture can be seen in Figure 7.

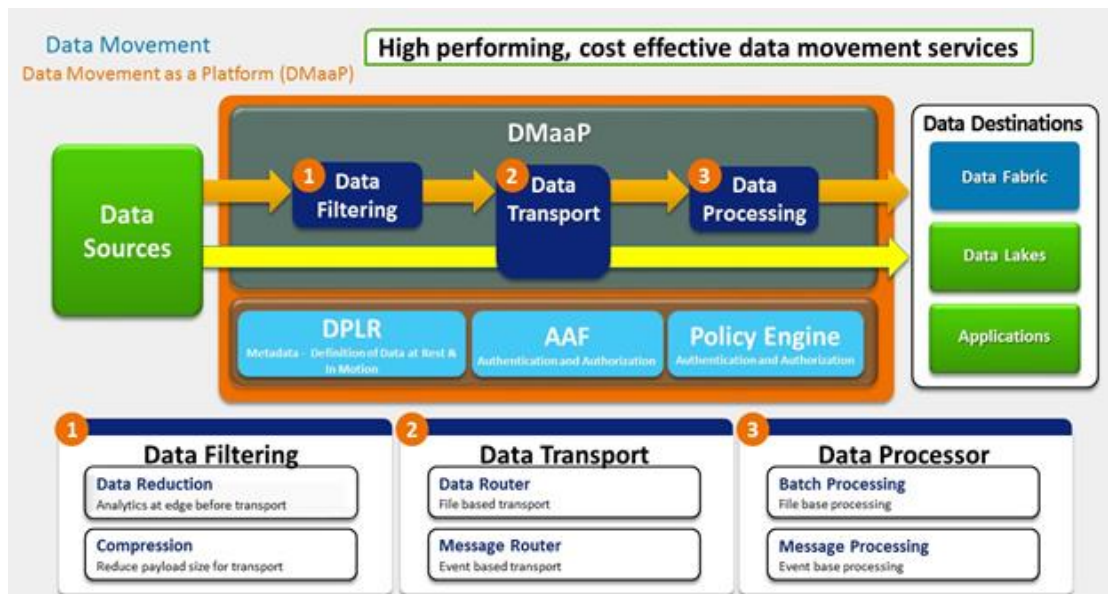


Fig. 7 DMaaP's architecture

I.8. VID

The Virtual Infrastructure Deployment module enables users to launch instances of infrastructure services and their associated components, which have been designed in the SDC [29].

These include [28]:

- Service models
- VNFs
- VF modules
- Volume groups

After an operator inputs the appropriate data, the VID triggers the SO to instantiate the associated service model or component [29].

VID can also be used to [29]:

- Browse the SDC catalog for service models
- Search for and display service instances
- Modify service instances

Its architecture can be seen in Figure 8.

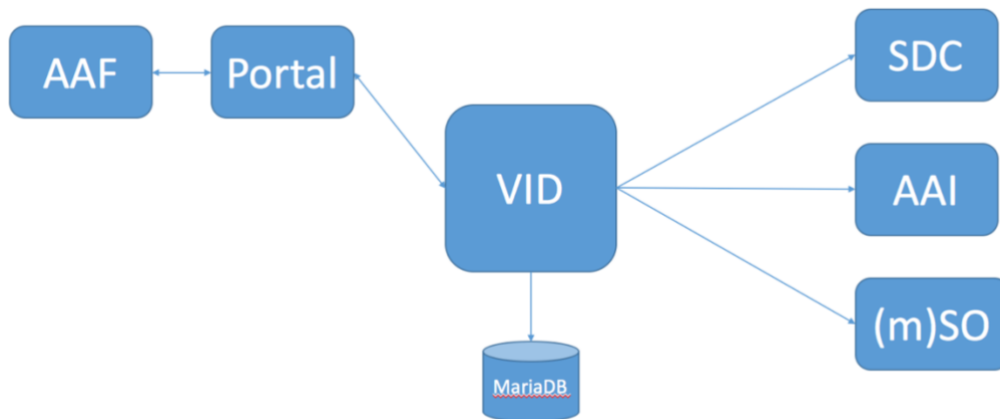


Fig. 8 VID's architecture

I.9. Robot

Basically, Robot offers a console line interface that can be accessed via SSH into the Robot virtual machine. Robot is the only way a network manager has to access the ONAP virtual machines to control them from the inside [30].

Robot is required by all the ONAP's use cases but can be replaced by the Portal's graphic interface and the usage of the different APIs all the modules offer [30].

I.10. APPC

The Application Controller (APPC) manages and controls the life cycle of virtual applications, such as virtual network functions and their components. This module receives commands from internal ONAP components, such as SO, DCAE, or Portal [31].

A virtual application is composed of the following layers of network technology [31]:

- Service
- Virtual Network Function (VNF)
- Virtual Network Function Component (VNFC)
- Virtual Machine (VM)
- A Life Cycle Management (LCM)

An LCM command is sent as a request to the APPC using an HTTP request, or in a message on OpenECOMP's bus (DMaaP). For each request, the APPC returns one or more responses [31]:

An asynchronous command, which is sent as an authorized and valid request, results in at least two discrete response events [31]:

- Accept response: Indicates that the request is accepted for processing
- Final response: Indicates the status and outcome of the request processing

An unauthorized or invalid request results in a single error response. A synchronous command results in a single response that is either success or error [31].

Its architecture can be seen in Figure 9.

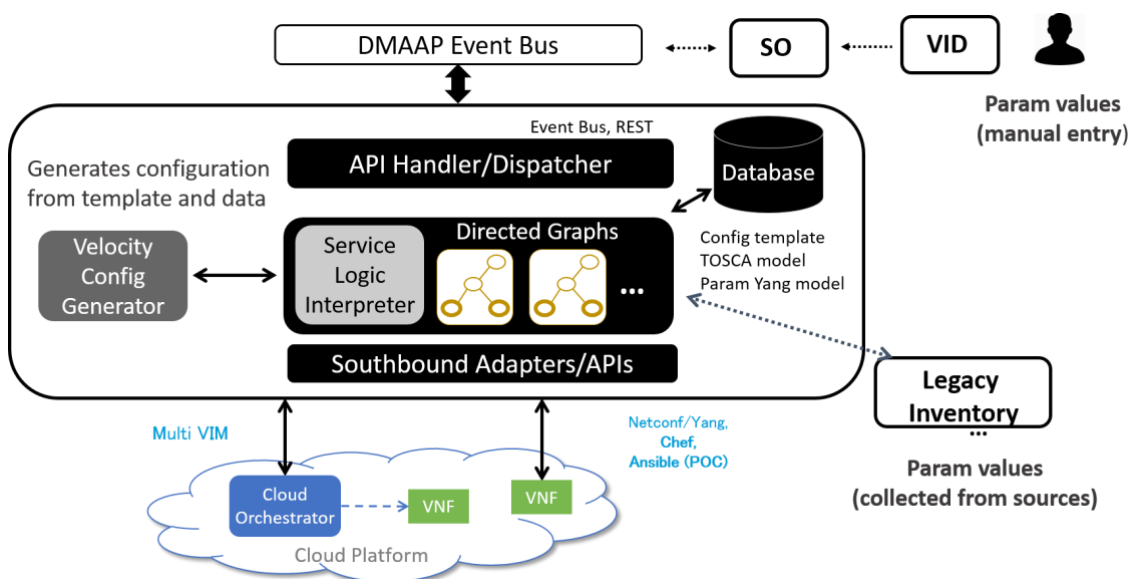


Fig. 9 APPC's architecture

I.11. Policy

The ONAP Policy Framework is a comprehensive policy design, deployment, and execution environment. It's the decision making component in ONAP, which allows specifying, deploying, and executing the governance of the features and functions in ONAP [32].

One of the most important goals of the Policy Framework is supporting Policy Driven Operational Management during the execution of ONAP control loops at run time [32].

Moreover, use case implementations such benefit from the ONAP policy Framework because they can use its capabilities to manage and execute their policies instead of embedding the decision making in their applications [32].

The Policy Framework is deployment agnostic. It manages policies independently of how they are deployed. In one deployment, policy execution could be deployed in a separate Docker container. In another, policy execution could be co-deployed with an application to increase performance [32].

The ONAP Policy Framework architecture separates policies from the platform that is supporting them. The framework supports development, deployment, and execution of any type of policy in ONAP [32].

The Policy Framework is model driven so that policies can be as flexible as possible in order to support agile development procedures. A model driven approach also allows reducing the amount of programmed support required for policies [32].

There are five fundamental capabilities for the framework [32]:

- The framework must be able to be triggered by an event or invoked, and also be able to make decisions at run time.
- It must be deployment agnostic.
- It must be model driven, allowing policies to be deployed, modified, upgraded, and removed as the system executes.
- It must provide a flexible model driven policy design approach for policy type programming and specification of policies.
- It must be extensible, allowing straightforward integration of new policy formats and policy development environments.

Its architecture is shown in Figure 10.

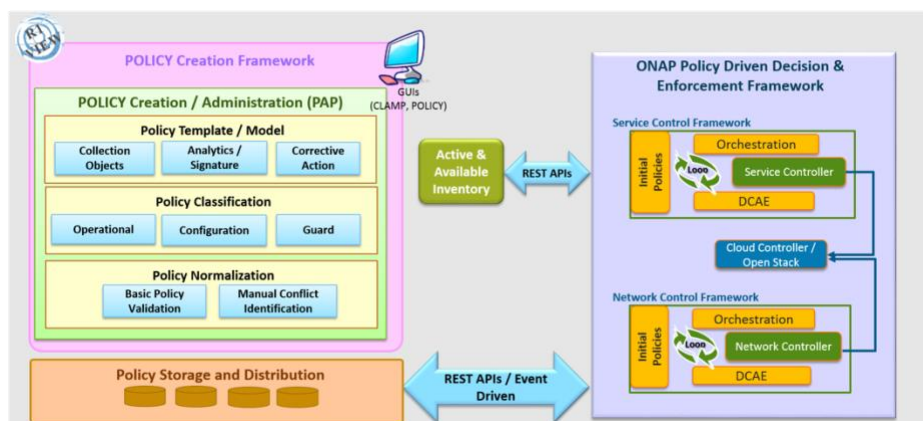


Fig. 10 Policy's architecture

ANNEX II: VIRTUAL FIREWALL IMPLEMENTATION

As it has been commented in the document, we were unable to keep working due to incompatibilities between ONAP's and OpenStack's Keystone versions.

However, in this annex you will find the last steps to take in order to complete the installation, instantiation, and deployment of the virtual firewall use case.

All the information related to this annex, including both the text and the pictures, has been extracted from ONAP's community videos and tutorials that help users understand how to implement ONAP's use cases. We are just referring them.

The creation of this annex has only been possible due to ONAP's community, especially thanks to Marco Palatina's video from which we have taken screenshots in order to illustrate each step [33].

Continuing from where we left of in point 3.2.3, the deployment of both the sink and the packet generator are a success, as it can be seen in Figure 1.

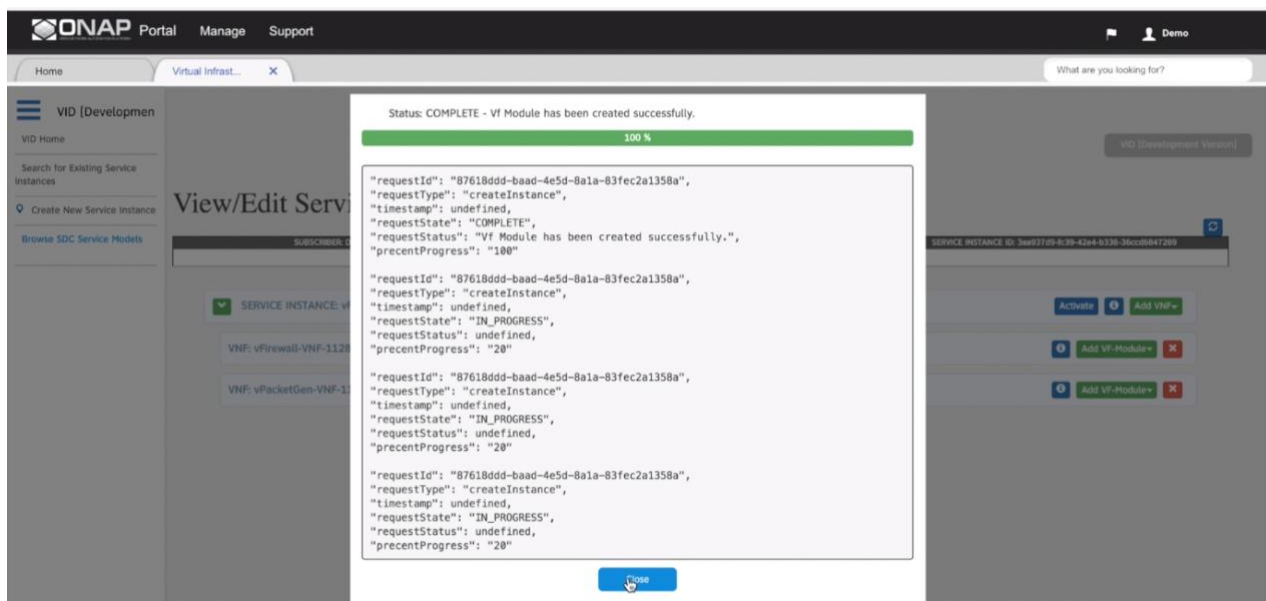


Fig. 1 VID's successful deployment

Moreover, this instantiation motivates the creation of two new virtual machines in OpenStack.

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
<input type="checkbox"/> vPacketGen-vm01-128	ubuntu-14-04-cloud-amd64	external • 10.12.5.214 zdfw1fw01_unprotected • 192.168.10.200 oam_onap_Wov7 • 10.0.80.2	m1.medium	vfw_key_HZnu	Active	nova	None	Running	0 minutes	Create Snapshot
<input type="checkbox"/> vFirewall-VNF-1128-1	ubuntu-14-04-cloud-amd64	external • 10.12.5.248 zdfw1fw01_unprotected • 192.168.10.100 zdfw1fw01_protected • 192.168.20.100 oam_onap_Wov7 • 10.0.80.1	m1.medium	vfw_key_KCT1	Active	nova	None	Running	4 minutes	Create Snapshot
<input type="checkbox"/> zdfw1fw01snk01	ubuntu-14-04-cloud-amd64	external • 10.12.5.222 zdfw1fw01_protected • 192.168.20.250 oam_onap_Wov7 • 10.0.80.3	m1.medium	vfw_key_KCT1	Active	nova	None	Running	4 minutes	Create Snapshot

Fig. 2 New OpenStack's vSNK and vPKG virtual machines

At this point, the vFW service can be considered as installed and deployed.

However, we still have to associate the stack with the service ID, a step we will explain in the next point.

With the packet generator's endpoints up and running and with the virtual sink's graphic interface, we proceed to study the performance of the virtual firewall in order to make sure it works as intended.

Also, we will contemplate the steps to be taken in order to recreate more useful and realistic scenarios with ONAP.

II.1. vSINK and vPKG interaction

At first, there is only one stream of data. That stream is visible both in OpenStack and in the graphic interface. However, the graphic interface gives much more orderly information than OpenStack.

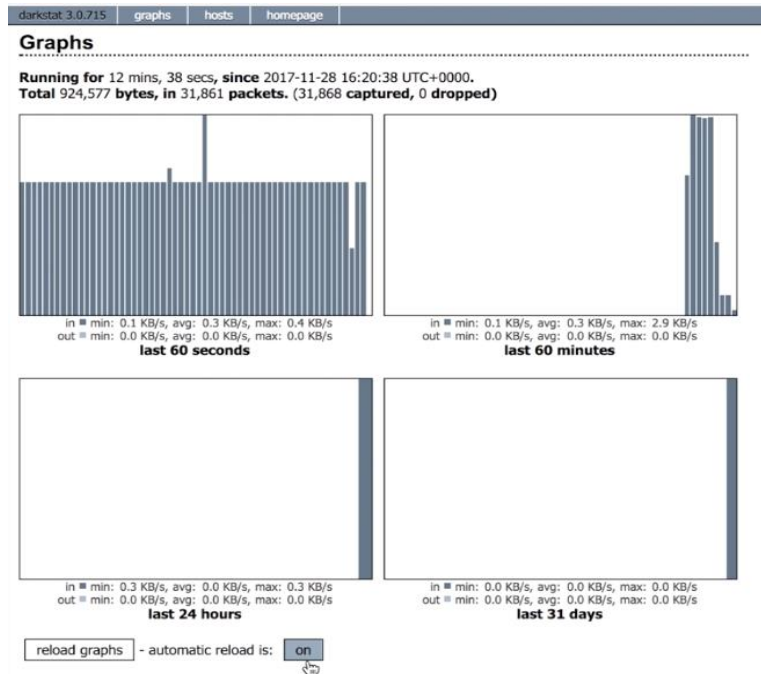


Fig. 3 The stream of data as seen by the graphic interface

The script loaded in the packet generator alternates between low traffic and high traffic, starting, as it's shown, with low traffic.

However, the DCAE module will generate events that the policy will use to trigger closed-loop operations, such as increasing or decreasing the flow of data.

Since the script specified to increase the data generation if a low threshold was crossed, we see how the data suddenly rises.

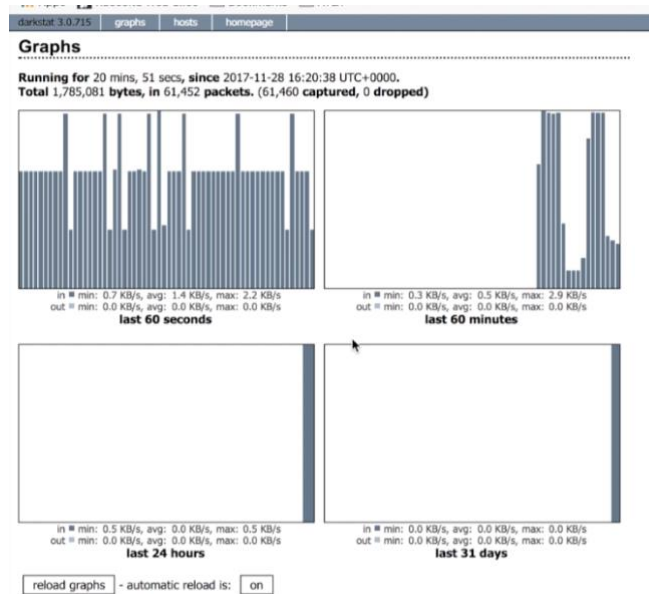


Fig. 4 Sudden increase of data seen in the graphic interface

This happens because Policy detects the event generated by the DCAE module and increases the number of flow from one to five.

To make sure the graphic information shown by the sink is reliable, we check the number of streams by sending an HTTP petition to the APPC module. As it can be seen in Figure 5, the information shown by the sink's interface was correct.

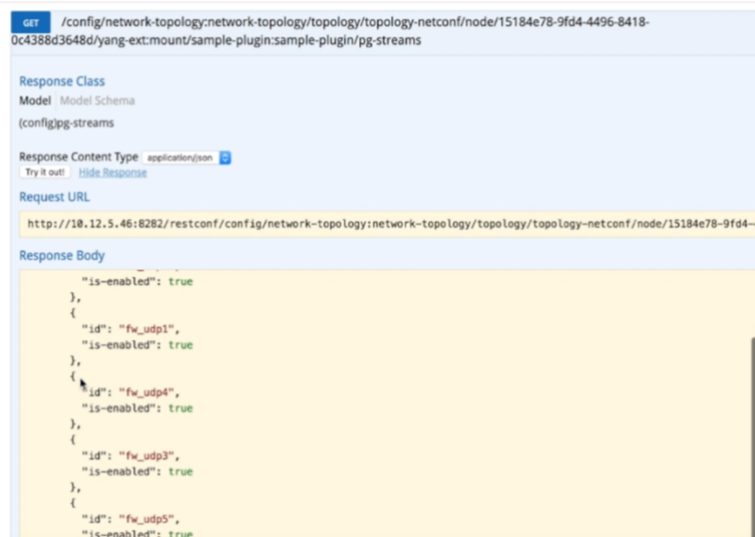


Fig. 5 Number of streams given by the APPC module

After having checked that both the graphic interface and the APPC module give the same information, we can ascertain the correct functioning of ONAP's closed-loop.