



Escola d'Enginyeria de Telecomunicació i  
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# MASTER THESIS

**TITLE:** Implementation of an NFV monitoring system for reactive environments

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

**AUTHOR:** César Cornelio Cajas Parra

**ADVISOR:** Carolina Fernández

**TUTOR:** David Rincón Rivera

**DATE:** July 14th, 2022



**Title: Implementation of an NFV monitoring system for reactive environments.**

**Author: César Cornelio Cajas Parra**

**Advisor: Carolina Fernández**

**Tutor: David Rincón Rivera**

**Date: July 14th, 2022**

## **ABSTRACT**

This work aims at researching the existent solutions of monitoring and alerting techniques, as well as defining a suitable architecture, design and implementation of a complete and customizable monitoring and alerting framework used to inspect and notify specific conditions on dynamically instantiated applications operating in the network. Such Network Services (NS) are used in the Network Function Virtualization (NFV) architecture, allowing rapid instantiation and configuration of virtualized environments that handle network configuration.

This design and implementation seek to provide more flexibility and dynamicity to the network operator to monitor custom or generic metrics and trigger notifications based on custom thresholds, without depending on the Virtual Network Function (VNF) developer to adapt its descriptor and onboard each version into the NFV Orchestrator (NFVO) prior to each usage. The framework here developed follows a modular architecture that separates the monitoring and alerting policies from the onboarding and instantiation process of the Network Functions. The architecture also facilitates the integration with other systems and adapting the functionality of an operational environment thanks to its decoupled and modular approach.

The presented work considers a monitoring and alerting framework that is especially useful for dynamic environments such as those relying in NFV, like those in the EU H2020 PALANTIR project. There, the framework is used to help assessing the correct behavior of the Security NSs that are used to prevent or mitigate security anomalies in the network of each client. If abnormalities are found, remediation measures will take place to replace the potentially compromised NS instances with clean, appropriate ones.

## **ACKNOWLEDGEMENTS**

I would like to thank my tutor Carolina Fernandez for guiding me through all the processes required to accomplish the objectives of this work and to all the Software Networks team (part of the i2CAT foundation) involved in it.

I want to thank also to my parents Cornelio and Cecilia and my sisters Daniela and Valeria, who have supported me in this journey and motivated me to accomplish another milestone in my career.

## LIST OF FIGURES

Figure 1: High Level NFV framework .....	4
Figure 2: ETSI NFV reference architectural framework.....	5
Figure 3: Prometheus Architecture .....	8
Figure 4: Grafana dashboard .....	9
Figure 5: OSM Monitoring structure .....	13
Figure 6: DCAE architecture diagram.....	14
Figure 7: Proposed active monitoring framework .....	15
Figure 8: Message sequence flow for active monitoring.....	16
Figure 9: Monitoring schemes: a) passive/centralized, b) active/centralized....	17
Figure 10: Proposed monitoring and alerting framework, as part of the SCO subcomponent.....	21
Figure 11: Monitoring module architecture .....	23
Figure 12: Policies module architecture .....	23
Figure 13: MongoDB data model for monitoring and alert system.....	24
Figure 14: Data modeling for MON.....	25
Figure 15: Data modeling for POL.....	25
Figure 16: Mean response time for a metric under passive monitoring (Prometheus) .....	32
Figure 17: Mean response time for a metric under active monitoring (SSH Channel).....	32
Figure 18: Mean response time for POL to notify upon detecting a threshold being surpassed.....	33
Figure 19: Mean response time for POL as MON increases its observed metrics .....	34
Figure 20: Performance degradation of POL for 10h.....	36
Figure 21: Performance comparison between active and passive monitoring..	37
Figure 22: VIM account creation.....	48
Figure 23: NS onboarding .....	49
Figure 24: Instantiated VNFs list .....	50
Figure 25: VNF specifications.....	51
Figure 26: Prometheus Server (Targets).....	53
Figure 27: Node Exporter Metrics.....	54
Figure 28: Prometheus Pushgateway dashboard.....	56

## LIST OF TABLES

Table 1: VNFs' data sources, measurements to collect and problems.....	12
Table 2: Differences between active and passive monitoring.....	17
Table 3: Relation between system's scraping time and Prometheus Overload	35
Table 4: Whitelist of allowed monitoring commands.....	40

# CONTENTS

<b>INTRODUCTION .....</b>	<b>1</b>
<b>CHAPTER 1. BACKGROUND .....</b>	<b>3</b>
<b>1.1. Network Function Virtualization (NFV) .....</b>	<b>3</b>
1.1.1 Reference Architecture .....	4
1.1.2 Monitoring framework .....	7
<b>1.2 Monitoring and alerting tools .....</b>	<b>7</b>
1.2.1 Prometheus and Grafana .....	8
1.2.2 PerfSONAR .....	9
1.2.3 Zabbix and Nagios .....	9
1.2.4 Network Management as a Service (NMaaS) .....	10
<b>CHAPTER 2. STATE OF THE ART .....</b>	<b>11</b>
<b>2.1. Monitoring in NFVOs .....</b>	<b>11</b>
2.1.1. OSM .....	12
2.1.2. ONAP .....	13
2.1.3. OPNFV .....	14
<b>2.2. Active monitoring framework .....</b>	<b>14</b>
2.2.1 Workflow definition .....	15
<b>2.3. Passive monitoring framework .....</b>	<b>16</b>
<b>2.4. Summary of monitoring techniques .....</b>	<b>17</b>
<b>CHAPTER 3. PROBLEM STATEMENT AND ARCHITECTURE OF THE SOLUTION .....</b>	<b>19</b>
<b>3.1 Problem Analysis .....</b>	<b>19</b>
3.1.1 Proposed solution .....	20
<b>3.2 Architecture of the monitoring framework .....</b>	<b>21</b>
<b>3.3 Design of the solution .....</b>	<b>22</b>
3.3.1 Guiding principles .....	23
3.3.2 Data modeling .....	24
3.3.3 Interfaces .....	25
<b>CHAPTER 4. IMPLEMENTATION AND DEPLOYMENT .....</b>	<b>27</b>
<b>4.1 Implementation decisions .....</b>	<b>27</b>
4.1.1 Retrieval of metrics .....	23
4.1.2 Data persistence .....	24
4.1.3 Language .....	25
4.1.4 Maintainability .....	23
<b>4.2 Deployment decisions .....</b>	<b>29</b>
4.2.1 Virtualization layer .....	23
4.2.2 Access control .....	24
4.2.3 Configuration .....	25

<b>CHAPTER 5. EXPERIMENTATION AND EVALUATION.....</b>	<b>31</b>
<b>5.1 Mean response time .....</b>	<b>31</b>
5.1.1 Comparison of mean response time to extract a metric between active and passive monitoring .....	31
5.1.2 Mean response time for the alerting subsystem.....	33
<b>5.2 Compromise in selected configuration values.....</b>	<b>34</b>
5.2.1 Selection of the acquisition frequency for metrics .....	34
5.2.2 Selection of the maximum data retention .....	35
<b>5.3 Performance Degradation.....</b>	<b>36</b>
 <b>CHAPTER 6. CONCLUSIONS AND FUTURE WORK .....</b>	 <b>38</b>
<b>6.1 Conclusions of the work .....</b>	<b>38</b>
<b>6.2 Future work .....</b>	<b>39</b>
<b>6.3 Considerations on security, ethics and privacy.....</b>	<b>39</b>
6.3.1 Security .....	39
6.3.2 Potential to detect Indicators of compromise .....	40
6.3.3 Ethics and privacy .....	41
<b>6.4 Considerations on environmental sustainability .....</b>	<b>42</b>
 <b>ACRONYMS .....</b>	 <b>43</b>
 <b>REFERENCES.....</b>	 <b>44</b>
 <b>ANNEX I: DATA MODELS .....</b>	 <b>46</b>
1.1 MongoDB data modeling .....	46
1.2 MON and POL data modeling .....	47
 <b>ANNEX II: OSM .....</b>	 <b>48</b>
1.1 Virtual Infrastructure Manager (VIM) account setup .....	48
1.2 Network Service (NS) onboarding and instantiation .....	49
 <b>ANNEX III: OPENSTACK .....</b>	 <b>50</b>
1.1 Navigating the VMs in use by the VNFs .....	50
 <b>ANNEX IV: PROMETHEUS .....</b>	 <b>52</b>
1.1 Creating the Prometheus Server (e.g., locally).....	52
1.1.1 Prometheus Node Exporter .....	53
1.1.2 Creating the Prometheus Pushgateway .....	54



**ANNEX V: REST CALLS..... 57**

**1.1 Monitoring subsystem MON..... 57**

**1.2 Alerting subsystem POL..... 59**



## INTRODUCTION

The Network Function Virtualization (NFV) paradigm leverages on the benefits and growth of the virtualization technology. NFV offloads Network Functions (NFs) from dedicated hardware to software running on Commercial Off-The-Shelf (COTS) equipment [1]. Several telecommunication operators have worked together to release an Industry Specification Group for NFV in accordance with the European Telecommunication Standard Institute (ETSI) in 2012 [2]. One of these standards is dedicated to active monitoring and failure detection, where the design of the proposed monitoring system is based on the described reference framework.

One way to obtain a good performance of NFs is to monitor them properly. From the point of view of the technology, the measurements provided by the monitoring process heavily rely upon for managing systems, given that such outcomes indicate the state of the system and can expose abnormalities on any of its components.

The monitoring of NFs in the Open-Source MANO (Management and Orchestration) (OSM)<sup>1</sup> NFVO, widely used in the research ecosystem, requires defining specific metrics inside of the VNF descriptor, prior to its packaging and onboarding in OSM. This introduces a dependency from the network operator towards the developer. Instead, a more adequate approach is to avert this need and provide operators with enough flexibility and customization to dynamically monitor the NS instances running in their infrastructure. Furthermore, it should serve as a generic and modular monitoring system that any developer, integrator or operator can use without OSM.

The benefits of the proposed approach are the following:

- Separation between the definition of the VNF and the extracted metrics. Thus, the operator does not need to understand nor update the VNF descriptor, repackage and onboard it again; which hogs the infrastructure and the OSM instance with slightly changed versions, as well as increased management tasks. Another benefit is that the operator does not need to understand the inner behavior of the NF instance.
- Selection of both generic and custom metrics. Generic metrics are extracted by a third-party module called Prometheus Node Exporter; whilst custom metrics are manually defined (as UNIX commands) and fetched directly from the NF instance
- Binding of alerts with previously defined metrics. With this, alerts are generated once a given condition is met.

---

<sup>1</sup> ETSI-hosted project to develop an Open Source NFV for management and orchestration (MANO)

This thesis seeks to compare currently available monitoring solutions, design the architecture of the complete system and of the monitoring and alerting subsystems, implement these and evaluate the overall performance. Therefore, this work focused on developing and implementing a modular NFV monitoring and alerting system that allows any operator to manage a number of NFs instantiated in a Network Function Virtualization Infrastructure (NFVI).

This document is organized as follows:

This document is organized as follows: (1) a first chapter dedicated to the background information about key concepts related to NFV, as well as tools used for monitoring and alerting; (2) a second chapter related to the State of the Art, where both general monitoring techniques are detailed and where the monitoring approaches adopted by NFVOs; (3) a third chapter that explains the problem analysis/statement and proposed solution, design and architecture and resources needed to develop the API; (4) a fourth chapter that contains all the implementation and deployment description of the monitoring and alert system; and (5) a final chapter elaborating on the API's evaluation and performance.

The document ends with a final chapter that contains general conclusions and future work lines to be implemented as well as brief sustainability considerations. There are also several annexes with details that were omitted in the main body of the document for the sake of brevity and clarity of the narration.

## CHAPTER 1. BACKGROUND

This chapter will delve into the concepts of the involved technologies that are used to develop the monitoring and alert system.

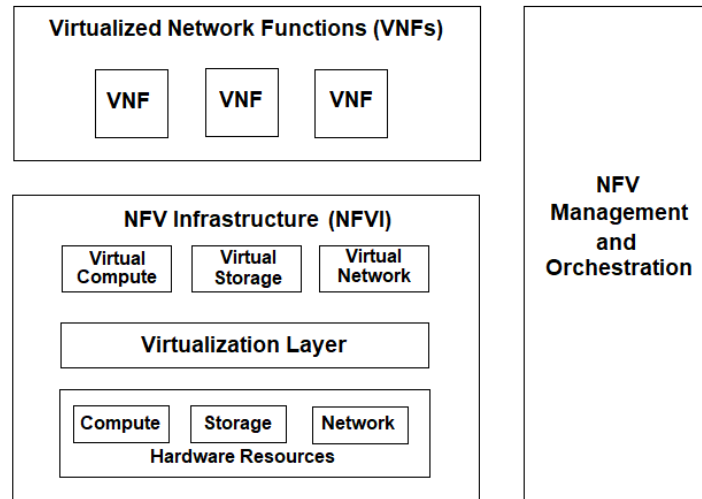
First, the Network Function Virtualization concept is described, along with its reference architecture and the active and passive monitoring framework to be used. In a second part of the chapter, different monitoring and alerting tools are shown to be compared to have a better understanding of the benefits and drawbacks of each of them and look for the implementation of the best fit.

### 1.1. Network Function Virtualization (NFV)

The NFV concept involves the offloading (that is, transferring the logic of the network functions) from dedicated hardware appliances to software-based applications, taking advantage of the rapid evolution of IT virtualization. These applications are executed on standard IT platforms like high-volume servers, switches, and storage. With NFV, the instantiation of Network Functions (NFs) is possible and allow deploying in heterogeneous networks in a transparent manner [3].

NFV provides a good number of benefits to the telecommunication industry, such as the flexibility and scalability, open-source platforms, improvements in operations performance and time development cycle, but most importantly reducing CAPEX and OPEX investments [4].

The high level NFV architectural framework [5] is composed of three important working domains, as shown in (Figure 1). Specifically, these are the Virtualized Network Functions (top), the NFV Infrastructure (bottom) and the NFV Management and Orchestration (right).

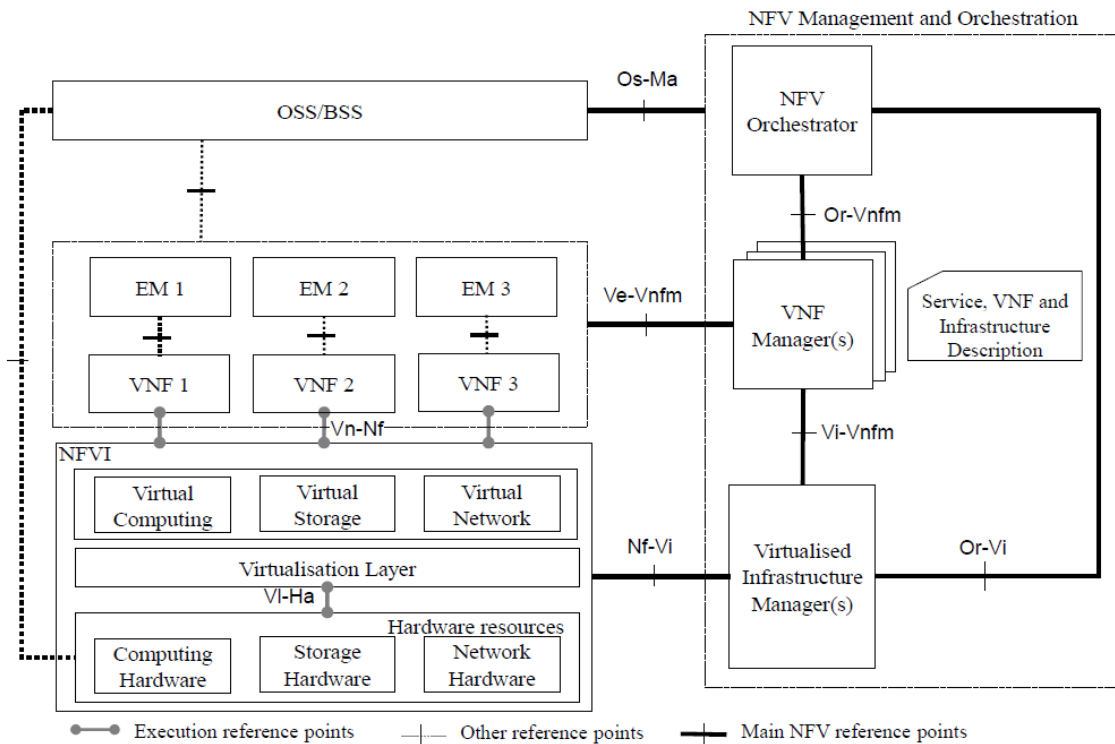


**Figure 1:** High Level NFV framework

Each VNF runs the network-related logic on top of the NFVI. On the other hand, the NFVI conforms the virtualized resources where the VNF is running. Finally, the NFV Management and Orchestration (MANO) oversees the orchestration and lifecycle management of the VNFs; and could partially interact with other physical and/or software resources.

### 1.1.1 Reference Architecture

The NFV architectural framework focal point is on the new functional blocks in the operator's network resulting from the virtualization process. Figure 2 [5] illustrates the NFV architectural framework illustrating the functional blocks and most relevant points in the NFV framework. The most important functional blocks are the VNFs, Element Management (EM), NFV infrastructure, Virtualized Infrastructure Manager (VIM), NFV Orchestrator, VNF Manager, Service, VNF and Infrastructure Description and the Operations and Business Support Systems (OSS/BSS). Solid lines highlight the reference points of NFV, while dotted lines represent available points present in actual deployments that might require adaptations to support NFV. This framework focuses on the functionalities that must be included in a virtualized environment able to operate a network, other than that, the network operator decides which network function should be virtualized.



**Figure 2: ETSI NFV reference architectural framework**

#### 1.1.1.1 Virtualized Network Function (VNF)

VNFs carry out virtualized tasks in the networks, which were previously performed by proprietary dedicated hardware [6]. VNFs can run as a Virtual Machine (VM) or containers (i.e., Docker). These tasks, which are used by both network service providers and enterprises, include functionalities typically present in routers, switches, firewalls and other devices.

#### 1.1.1.2 Element Management (EM)

The Element Management is the responsible of the good functioning of the VNF [7]. The EM manages important functional components of the VNF called FCAPS (Fault, Configuration, Accounting, Performance and Security management). One EM could manage one or several VNFs, additionally, EM could be considered as a VNF itself.

#### 1.1.1.3 VNF Manager

The VNF Manager (VNFM) is in charge principally of VNF lifecycle management [7]. The lifecycle management executes diverse operations over the NSs such as deployment, deletion and lifecycle operations which includes service start/stop, scalability, inner configuration, monitoring, etc. The main difference between the

EM and the VNFM is that the first one is responsible for the management of the functional components of the VNF, meanwhile, the second one manages the virtual components.

As an example, let us assume we have a virtualized mobile core, VNFM could manage issues related to the deployment of the VNF and the EM manages issues related to the inner configuration/functionality of the service.

#### **1.1.1.4 Network Function Virtualization Infrastructure (NFVI)**

NFVI [5], includes the physical and virtual resources (both, hardware and software) that make up the virtualization environment where the VNF is deployed. NFVI could be potentially distributed across multiple locations.

#### **1.1.1.5 Virtualization Layer**

The Virtualization Layer [5], decouples the software of the VNF from the hardware it sits on. The Virtualization Layer guarantees that the VNFs lifecycle is independent of the hardware.

#### **1.1.1.6 Virtualized Infrastructure Manager (VIM)**

The Virtualized Infrastructure Manager [5], is the authority that allows a smooth interaction between the VNF and the network resources, storage and computing. The VIM manages the VNFs virtualization as well.

There are several VIMs that are accepted for the ETSI Open-Source MANO [8]: Openstack, OpenVIM, VMware's vCloud Director, AWS, Microsoft Azure or Google Cloud Platform. For this thesis, we will be using Openstack as our VIM.

#### **1.1.1.7 NFV Orchestrator**

The NFV Orchestrator [5], will be the one that orchestrates and manages the NFVI and software resources. The NFVO acknowledges the network services instantiated in the NFVI. The NFVO interfaces are with the VNFM (Or-Vnfm) and VIM (Or-Vi).

#### **1.1.1.8 Operation Support System/Business Support System (OSS/BSS)**

The OSS and BSS [7], may be directly used by the network operator. OSS deals with network, fault, configuration, and service management. The BSS oversees customer, product, and order management. The OSS/BSS communicates directly to the NFV MANO through the Os-Ma interface.



For this work, the monitoring process is expected to act at the "Ve-Vnfm" interface to passively obtain or actively retrieve metrics. It could also potentially act at the "Or-Vi" interface in case the targeted node to monitor is not a VNF, but part of the VIM. This latter option is open to the operator.

### **1.1.2 Monitoring framework**

Monitoring techniques can be active and passive. The following subsections describes both of them. These techniques, and other aspects related to monitoring are more detailed in Chapter 2.

#### **1.1.2.1 Active monitoring techniques**

Active monitoring acquires specific metric(s) lively from the network in a proactive manner. This is used to monitor live networks. To monitor a network using an active technique, it is necessary to send traffic to the network in order to assess its health properly [9]. The importance of the active monitoring appliance is that it could provide fault detection, performance degradation and configuration issues.

#### **1.1.2.2 Passive monitoring techniques**

This monitoring technique, called passive monitoring [10], also monitors live networks. It focuses on the analysis of the user's real data. One benefit of using passive monitoring is the response time, as it usually collects metrics faster than the active monitoring. The disadvantage of these techniques is that the measurements can be only analyzed off-line. This creates two problems: delay in actionable results and big amount of data processing at once.

Chapter 2 elaborates on the active and passive monitoring framework techniques applied to NFV, as well as its workflow definition.

## **1.2 Monitoring and alerting tools**

This section is dedicated to elaborating on the most used and common monitoring tools that address traditional monitoring and alerting problems. The ones that are considered to be described are (i) Prometheus [11] and (ii) Grafana, (iii) perfSONAR [12], (iv) Zabbix and Nagios [13], (v) Icinga [14] and (vi) Network Management as a Service (NMaaS).

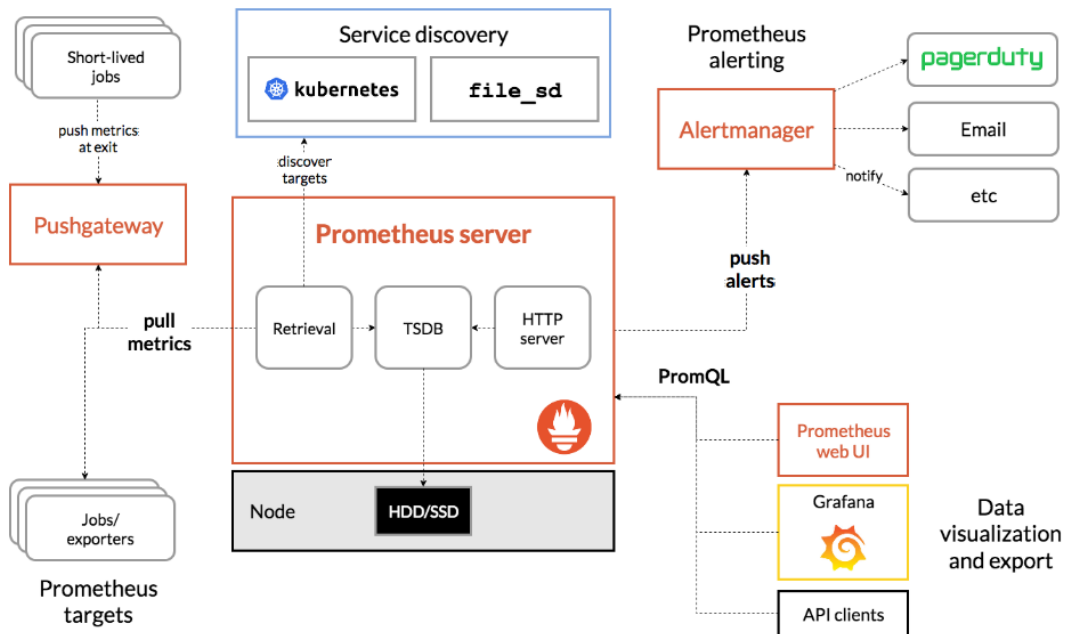
## 1.2.1 Prometheus and Grafana

### 1.2.1.1 Prometheus

Prometheus is an open-source monitoring and alerting system toolkit. Prometheus collects and stores its metrics as time series data. That is, values from metrics are stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels.

Prometheus's main features provide a multi-dimensional data model with time series data identified by metric name and key/value pairs, a flexible query language to leverage this dimensionality, autonomous single server nodes, time series collection via a pull model over HTTP, time series pushing supported for a gateway, targets discovery via static configuration, multiple modes of graphing and support for dashboards.

Figure 3 [11] shows the Prometheus server architecture, depicting the most relevant components: the main Prometheus Server, which scrapes and stores time series data; client libraries for instrumenting application code; a Pushgateway for supporting short-lived jobs; and Grafana, the data visualization tool.



**Figure 3:** Prometheus Architecture

Prometheus scrapes metrics from instrumented jobs, either directly or via Prometheus Pushgateway for short-lived jobs. It stores all scraped samples locally and runs rules over this data to either aggregate and record new time series from existing data or generate alerts. Grafana or other API consumers can be used to visualize the collected data.

Prometheus works well for recording any purely numeric time series. It fits both machine-centric monitoring as well as monitoring of highly dynamic service-oriented architectures. In a world of micro-services, its support for multi-dimensional data collection and querying is a particular strength.

### 1.2.1.2 Grafana

Grafana is a fully manageable observability platform for different applications and infrastructures, its benefits can be leveraged by connecting it to other tools like Prometheus, Loki and Tempo. This platform provides a graphical interface and a fully functional dashboard as shown in Figure 4.

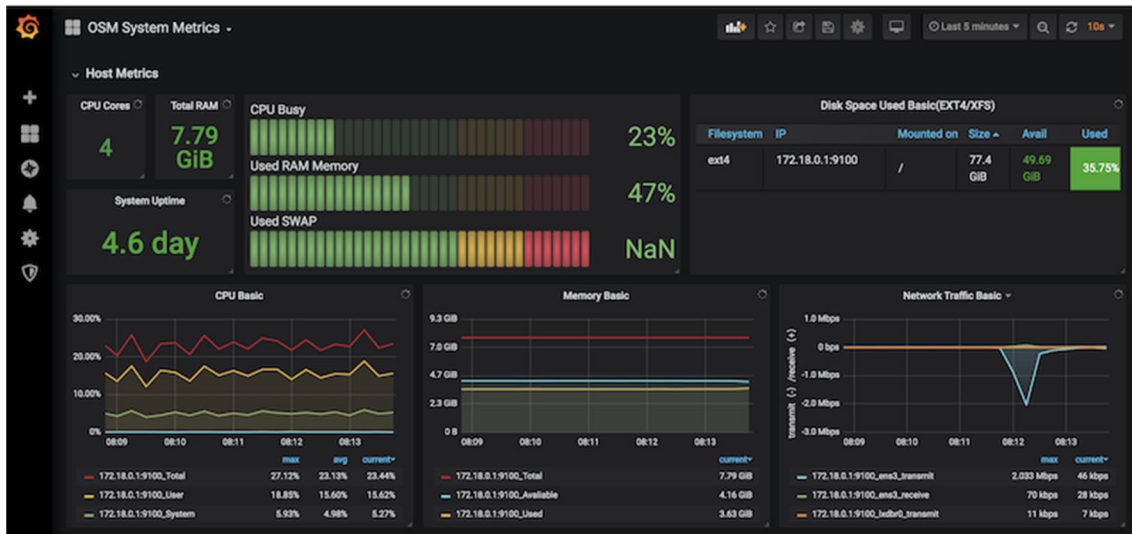


Figure 4: Grafana dashboard

Other benefits of this tool are the collection, storage, visualization and alerting on data running in a specific environment.

### 1.2.2 PerfSONAR

perfSONAR is a network measurement toolkit. perfSONAR principal aim is to transmit and share network measurements information/data with an end-to-end connection. The amount of tools that perfSONAR provides is extensive. This might seem a benefit, but it would increase the difficulty of the implementation for our monitoring and alerting subsystem.

### 1.2.3 Zabbix and Nagios

The Zabbix tool can be categorized as a monitoring application which is used to monitor the performance of VMs, networks and servers and is available as open source which can monitor CPU usage, network usage and disk consumption on machines. The monitoring tool can be configured with an API through a specific

interface. This tool can be a good candidate to use for the monitoring and alerting system but its approach is to fit big enterprises requirements.

Nagios is an open source continuous monitoring tool used to monitor the health of system machines, network infrastructure and architectures. The Nagios tool immediately sends an alert to the technician if something goes wrong with the system. A principal disadvantage of the use of Nagios is the lack of easy configurations access through a web interface, this is only permitted in a paid version of the tool. The time that requires configure the monitoring tool is around 15 minutes, but this process requires turn down the Nagios server and edit the configuration files. For this reason, it is not suitable for our monitoring and alerting system.

#### **1.2.3.1 Icinga**

Icinga is a monitoring system that checks the availability and performance of network resources, notifies problems to the users through its alert system, and creates data reports. Scalable and extensible, Icinga can monitor large, complex environments across multiple locations. Icinga was built as a fork of Nagios monitoring system. Its new release is Icinga 2 which contains new features that can be deployed on top of the Icinga stack. The main disadvantage of Icinga is that the architecture does not use the benefits of the time series databases and this will be necessary to use it next with Prometheus, therefore, is not a good selection.

#### **1.2.4 Network Management as a Service (NMaaS)**

NMaaS is a set of tools comprising network management applications that run in a secure network monitoring infrastructure. NMaaS user targets are the big organizations that are willing to outsource network management. GÉANT's NMaaS provides several services such as the management of the network infrastructure and support for both system and tools utilization.

Since the core service of NMaaS is outsourcing the network management, it is not suitable for our monitoring and alert system. This is because we could lose control and customization on the system being dependent on a third entity, which is not the objective of this work.

## CHAPTER 2. STATE OF THE ART

This chapter provides an overview of the current monitoring techniques used by different widely adopted NFVOs to monitor VNFs, describing the followed methodology for each and the advantages or drawbacks of each approach. Besides this, active and passive monitoring techniques are discussed in the context of NFV.

### 2.1. Monitoring in NFVOs

Orchestration is key when it comes to deploying different NSs, each made up of one or more NFs running in the same common NFV Infrastructure (NFVI) [15]. In order to provide the end users with their expected performance, different segments and nodes in the network must be monitored and potentially notified in a seamless way, making it possible to detect anomalies (e.g., security issues or potential service breaches) as soon as they occur.

The inherent multi-layer infrastructure implementing the NFV architecture requires an evolved monitoring approach relying on the following key aspects:

**Real-time analytics capability** i.e., the monitoring system should be able to correlate data from all the layers of the NFV architecture. This is important to realize the root of the problem. The correlation should be analyzed from top (service or function-related) and from bottom (infrastructure-related) layers.

**End-to-end active service monitoring:** i.e., it should be able to implement an active monitoring to test continuously end users service layers. This process should be automated.

Table 1 [15] resumes the data sources, measurements to collect and detect possible problems to consider for the VNFs and the virtualization layer.

In this thesis, it is considered collecting different metrics from the NFs (here, VNFs) that provide data for virtual system resources usage (CPU, Disk, RAM) by VNF application process; virtual system level KPIs; alarms; events and configuration.

**Table 1:** VNFs' data sources, measurements to collect and problems.

<b>Virtual Network Functions &amp; Virtualization layer</b>		
<b>Data sources</b>	<b>Measurements to collect</b>	<b>Problems for VNFs</b>
<ul style="list-style-type: none"> <li>•Application and OS data.</li> <li>•VNF Agents/Proxies or CLI scripts at VM level (for system resources)</li> </ul>	<ul style="list-style-type: none"> <li>•Virtual System resources usage (CPU, Disk, RAM) by VNF application process</li> <li>•Virtual System level KPIs, Alarms, Events, Configuration</li> </ul>	<ul style="list-style-type: none"> <li>•System blackout</li> <li>•Application Process blackout</li> <li>•Resource Usage Thresholds Alarms,</li> <li>•Unexpected events</li> <li>•Inconsistent VNF configuration</li> </ul>
Infrastructure orchestration VNF Managers	<ul style="list-style-type: none"> <li>•Virtual Infrastructure lifecycle events and triggered actions</li> <li>•Virtual Infrastructure inventory</li> <li>•VNF lifecycle events and triggered actions</li> <li>•VNF inventory</li> </ul>	Infrastructure and VNF Alarms (Resources unavailable, recovery action failures)
Compute workload active probes	Processing KPIs, RAM R/W KPIs I/O KPIs, Disk R/W KPIs	Percentage of operations above correctness and timing thresholds
VIM telemetry data (e.g. Openstack Telemetry) collected by agents/ proxies	<ul style="list-style-type: none"> <li>•Virtual resources consumption per VM</li> <li>•Cloud system resources usage, system level and application level KPIs, alarms, events, configuration</li> <li>•VM and VN inventory</li> </ul>	<ul style="list-style-type: none"> <li>•Node blackout</li> <li>•Application process blackout</li> <li>•Resource Usage Thresholds Alarms,</li> <li>•Unexpected events</li> <li>•Inconsistent SDN configuration (model, policies.)</li> </ul>

### 2.1.1. OSM

Open Source MANO (OSM) [16], implements an Open-Source Management and Orchestration (MANO) stack that is aligned with ETSI NFV standards (e.g., information models and interfaces). This project is led by a world-wide community that helps to deliver a production-quality MANO stack used for NFV deployments that meets requirements from the telecommunication operators.

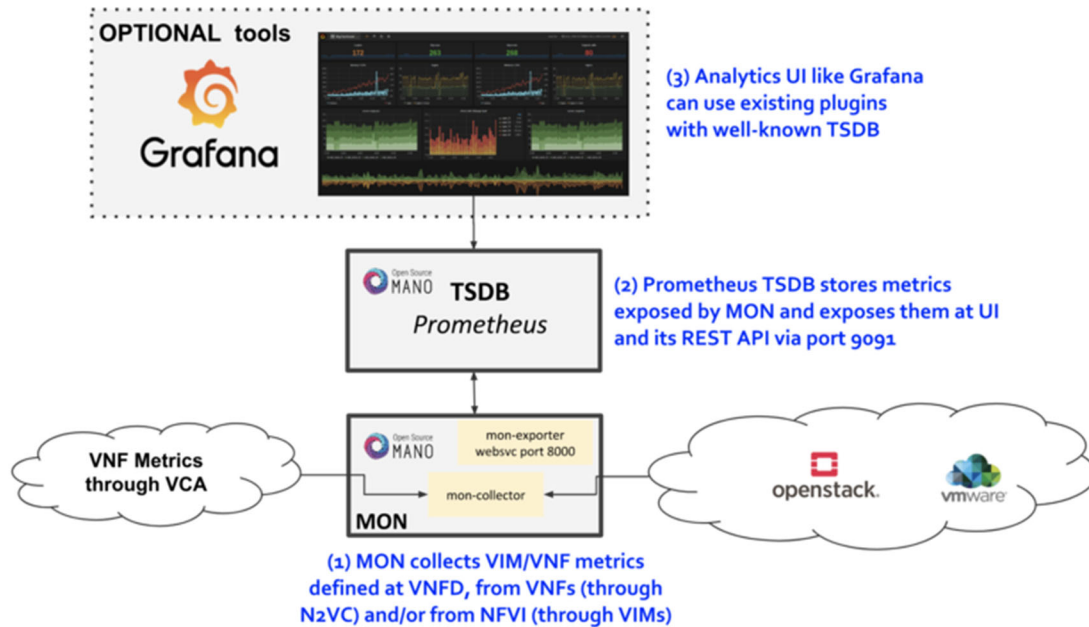
The documentation of this NFVO delivers a complete and up-to-date user guide, detailing every step from the first setup and configuration to NS development and instantiation, among other key functionalities. One of the guide's sections comprehends the monitoring and autoscaling features, in which VNFs metrics collection processes are described in the Virtual Network Function Descriptor (VNFD), and specific life-cycle management actions can be triggered upon metrics.

In order to collect metrics from a VNF, the OSM monitoring subsystem features a "mon-collector" module to extract metrics specified at the VNFD [17]. The metrics will be collected only if these exist at any of these two levels:

- NFVI, which can be made available by the VIM's built-in telemetry system.

- VNF, which are made available by OSM VCA (using Juju Metrics) and fetched from the VNF instances.

Figure 5 [17] describes graphically the OSM monitoring overall structure and its relation with ancillary tools for fetching and persisting data (Prometheus) and rendering it (Grafana).



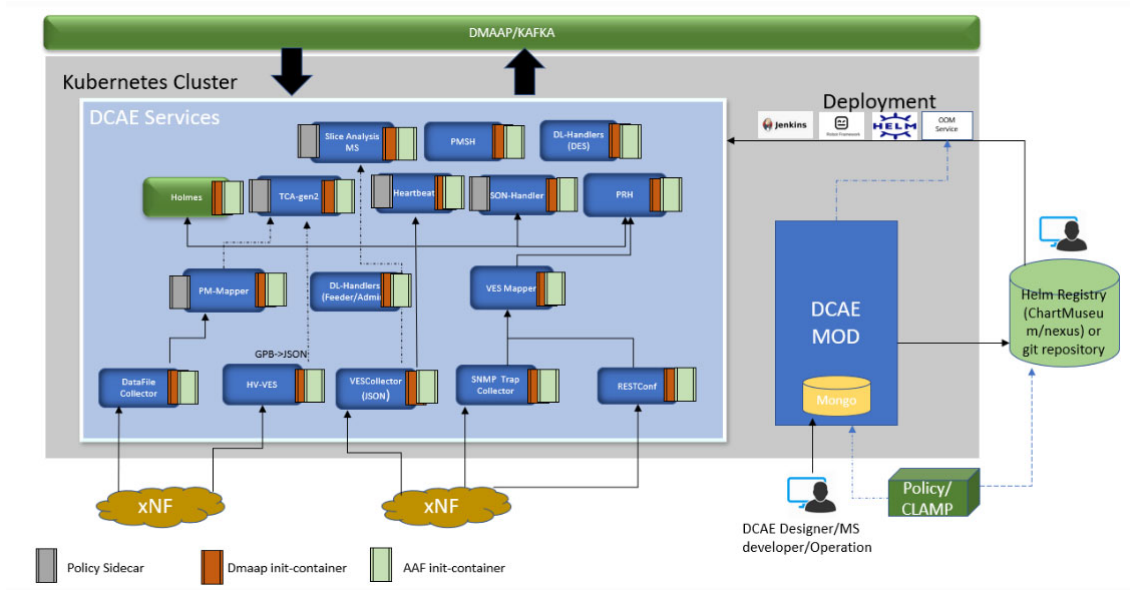
**Figure 5:** OSM Monitoring structure

### 2.1.2. ONAP

Open Network Automation Platform (ONAP) [18], is the orchestration framework in which the Data Collection Analytics and Events (DCAE) project is developed.

DCAE provides intelligence for ONAP to support automation by performing network data collections, analytics & correlation and trigger actionable root-cause events. To that end, it collects, receives and analyses monitoring data (e.g., health, performance, operational status) from VNFs. Active monitoring is used when polling from the VNF, but DCAE also uses passive monitoring, where VNFs directly register this information. The DCAE services components includes all the micro services - collectors, analytics and event processor which supports active data flow and processing as required by ONAP use cases.

Figure 6 [18] shows the DCAE architecture and how components work with each other. The components on the right constitute the platform controller which are statically deployed. The components on the left represent the services which can be both deployed statically or dynamically.



**Figure 6:** DCAE architecture diagram.

All services included in DCAE are offered as Docker containers and can be deployed as well as Kubernetes Deployments and Services.

The DCAE Services are: (1) collectors, such as Virtual Event Streaming (VES) collector or RESTConf collector; (2) analytics, such as Docker based Threshold Crossing Analytics and (3) event processors.

### 2.1.3. OPNFV

Open Platform for NFV (OPNFV) [19] has projects like “doctor” for the operators to detect faults in the systems and perform maintenance. They monitor with Nagios or Zabbix and is relying on the virtual resource manager (e.g., any resource provider) of OpenStack, the VIM in use. Another project is “barometer”, used to monitor performance to detect violations in the Service Level Agreements SLAs, as well as degradation in the performance of the resources in the NFVI, among others.

## 2.2. Active monitoring framework

The active monitoring framework proposed for NFV [20] consists of three core modules: test controller, Virtual Test Agent (VTA) and Test Result Analysis Module (TRAM). Each of these modules have a specific responsibilities and roles into the monitoring framework:

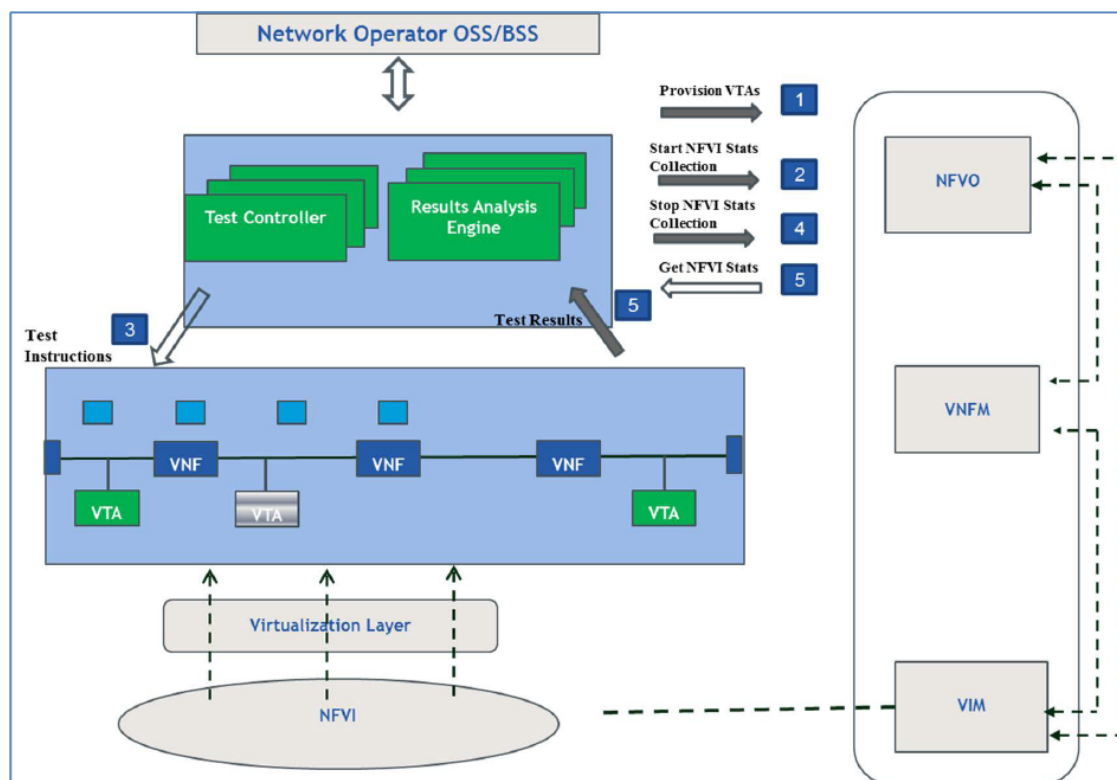
**Virtual Test Agent:** virtual entity that keeps the monitoring running in a VNF even within VNF migration scenarios.

**Test Controller:** maintains the configuration and catalogue of the Test Agent, tracks active tests, and other responsibilities.



**Test Result Analysis Module:** provides a report of scalability status, pulls information from the VTA and sends it to the OSS/BSS via NFVO and provides different test results.

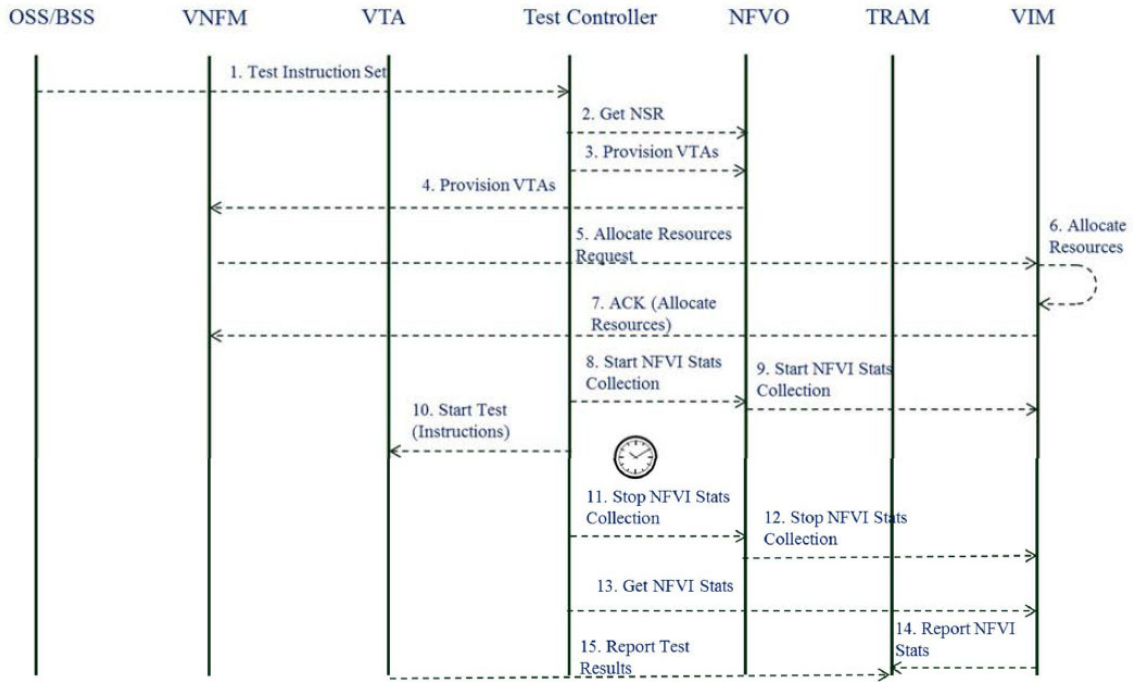
Figure 7 [20] illustrates the proposed active monitoring framework.



**Figure 7:** Proposed active monitoring framework

### 2.2.1 Workflow definition

Figure 8 [20] illustrates the messages exchanged between the active monitoring entities and the NFV entities for the provisioning of VTAs and the collection of NFVI statistics [21].



**Figure 8:** Message sequence flow for active monitoring

In the development of this thesis, VTAs are provisioned too. This process is done by directly placing the monitoring agent in the targeted VNF, in this case the VTA will be the Prometheus Node Exporter which uses passive monitoring. The active monitoring is a “built in” feature in the used Virtual Machines (VMs) since they have the Open SSH installed in their system to perform the SSH tunneling. The exporters will be monitoring metrics from inside of each VNF's system.

### 2.3. Passive monitoring framework

As the name suggests, passive monitoring does not require the involvement, or even awareness, of another site on the network. In its simplest form, passive monitoring may be nothing more than a periodic collection of port statistics such as bytes and packet sending and receiving counts.

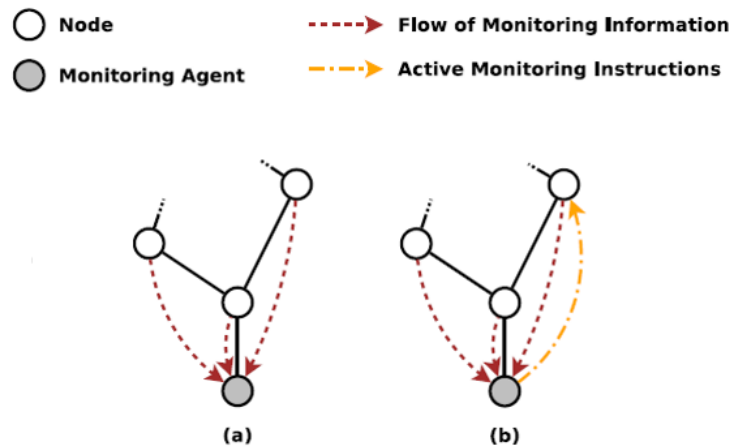
Table 2 presents the main differences between active and passive monitoring.

**Table 2:** Differences between active and passive monitoring.

	Active Monitoring	Passive Monitoring
<b>Quality</b>	Quality of service	Quality of experience (QoE)
<b>Analysis</b>	Direct, end-to-end	Correlated end-to-end
<b>Type of monitoring results</b>	Deterministic and predictive	Detailed traces for diagnosis
<b>Type of monitoring</b>	Real-time, end-to-end transport, network and service performance	Non-real-time, in-depth network, service and subscriber experience

## 2.4. Summary of monitoring techniques

The Orchestration solution design shall take into account several monitoring aspects. The first aspect to be considered is how the monitoring agents collect the metrics measurements. This enables a classification of monitoring as passive of active (Figure 9 (a) and Figure 9 (b), respectively).

**Figure 9:** Monitoring schemes: a) passive/centralized, b) active/centralized

According to the related specification proposed by ETSI [20], monitoring tasks may be passive, active, or hybrid.

In passive monitoring it is assumed that the monitoring server does not use a proactive approach to obtain metrics. Instead, it gathers unmodified metrics provided by the VNFs monitoring elements. Passive monitoring can analyze, for instance, the user traffic in real time and assume that the metrics obtained will be collected and processed off-line, this could provoke substantial delays between the events and the corresponding actions.

Active monitoring will use proactive fault detection in this case. The server waits for the metrics sent by the monitoring agents. It also runs autonomous actions, like tests or new flows generations for specific monitoring situations. The use of active monitoring techniques enables an iterative approach to analyze VNFs or NFVI resources without the need of user traffic [22].

## CHAPTER 3. PROBLEM STATEMENT AND ARCHITECTURE OF THE SOLUTION

In this chapter, the problem analysis that led to this proposal is discussed. The architecture and design of the monitoring subsystem MON and the alerting subsystem POL (also Policies Management Module) are also described, indicating the role that each module take and how the communications between the NFs and the monitoring subsystem MON are established

### 3.1 Problem Analysis

The NFV monitoring tools provided by OSM use the VNFD to configure and setup internal monitoring metrics for the VNF or VIM. The monitoring metrics must be defined in the descriptor file, in a way that fulfils the needs of the network operator. The monitoring definition in the descriptor is limited to some predefined reacting operations, like autoscaling. Therefore, this scenario presents a higher difficulty to accommodate the operator's needs that relate to the configuration and management of the monitoring services.

The main problem to tackle in this project is to remove the need of the operator to frequently modify the VNFD to be able to monitor the NSs by creating an independent and customizable metrics monitoring module. Currently, the monitoring in OSM is dependent on the VNFD. This reduces the capacity of the operator of stablishing new monitoring processes in a fast way. There are several limitations at the descriptor level of the OSM monitoring. These pitfalls are mostly related to the descriptor participation as a central point in the process of setting a monitoring and alerting configuration. A better approach to improve monitoring services is to decouple the VNF metrics from the descriptor to enable a better and more manageable system.

The VNFD is composed of a list of instructions that builds the VNF properties. These properties define the VNF resources such as virtual compute, storage and networking. Other properties to define are the type of connectivity, flavor, lifecycle management, autoscaling and monitoring.

To implement a new metric into the VNFD it is necessary to describe it at both the VDU and VNF levels. The description requires providing a metric ID and a normalized metric name (e.g., id: disk\_used, nfvi\_metric: cpu\_utilization). This is an example of how monitoring metrics are detailed in the VNFD. Therefore, the addition of a new metric represents the creation and instantiation of a new VNFD for the NS and the VNF, affecting the memory and space disk resources used in the NFVO (OSM). This, along with the predefined reacting operations, limits the ability of the operator to extend the life-cycle management and tailor it to its needs

This monitoring approach is not convenient since it does not provide enough flexibility to define customized metrics to be monitored, to configure these easily

and independently of both the NFVO in place and the logic of any given VNF in their VNFD; as well as not providing easy ways to bind with other reaction operations (e.g., though the emission of alerts to specific endpoints, which execute specific actions).

### **3.1.1 Proposed solution**

The devised solution averts the dependency between the definition of the logic for the NF instances that will operate the network and the network operation lifecycle itself. This provides the infrastructure operator with enough flexibility and customization to programmatically monitor the NS instances running in their infrastructure without the need to understand the logic of the NS and NF, nor of their descriptors. In general, with the abstraction of the VNFD dependency, the system becomes more flexible and enables the opportunity of dedicating more resources to other functionalities and services of the NFs. This is the reason why it is important to remove this characteristic of the OSM approach. Besides this, the proposed monitoring and alerting subsystem can be used by developers, integrators or operators in a modular fashion, making it possible to export to different environments, platforms and MANO tooling (e.g., to use with other NFVO other than OSM).

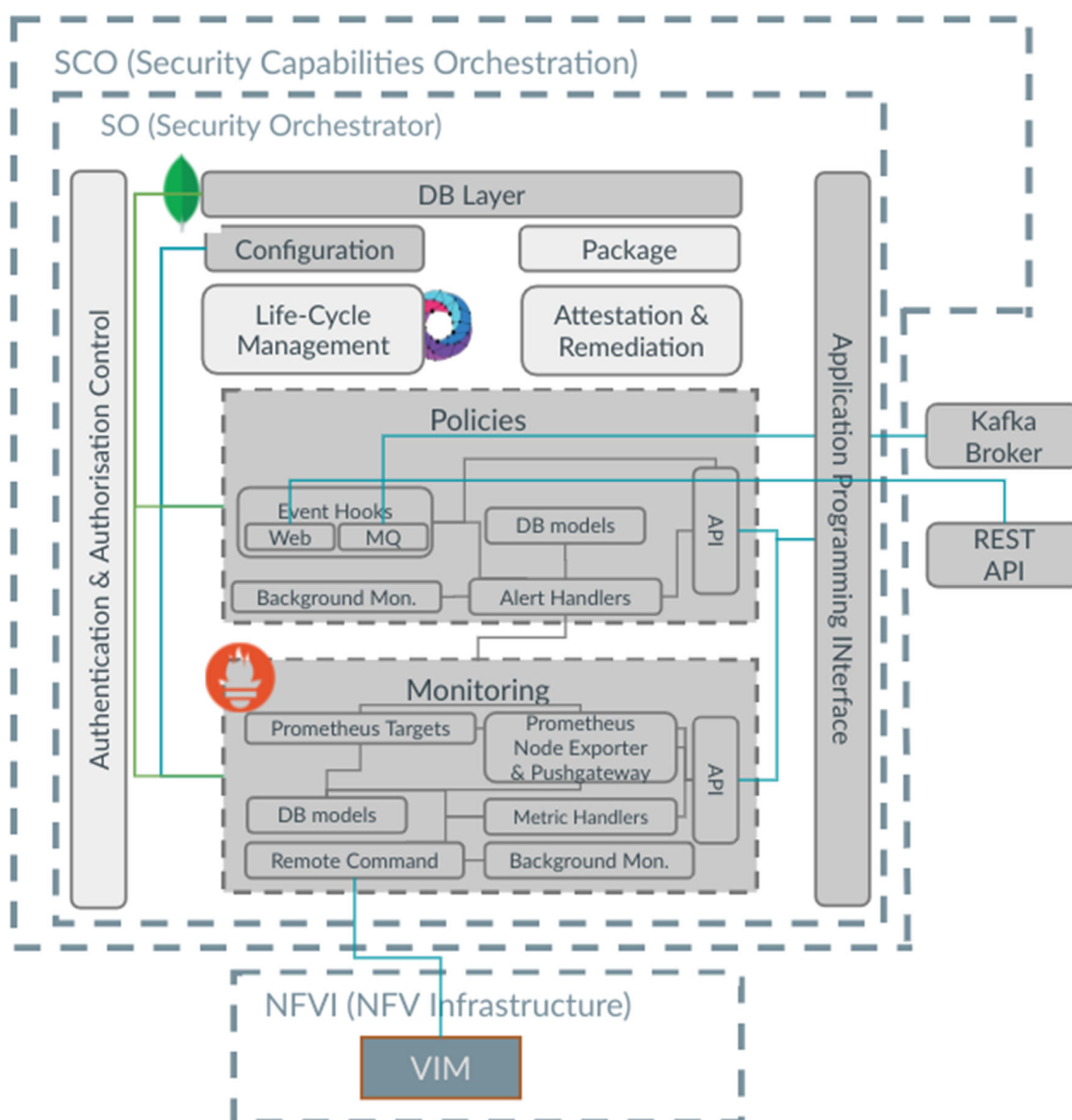
The benefits of the proposed approach are the following: (1) separation between the NF and the metrics extracted from it, so there is no need to understand nor update the VNF descriptor, repackage and on-board again – which hogs the infra and OSM with slightly changed versions or requiring running instances to be removed beforehand; (2) selection of generic metrics definition or custom metrics, where the generic metrics are defined by Prometheus Node Exporter and custom metrics are manually defined, as UNIX commands, and subject to filters; (3) binding of alerts with previously defined (generic or custom) metrics and where alerts are emitted upon meeting a given condition or threshold; and (4) easy integration with other systems and interfaces, given its modular approach.

The expected workflow followed by the proposed solution performs the first onboarding of the NS and associated NF(s) packages through OSM, with a basic descriptor configuration and allocates enough resources to the NF. The second step instantiates the NS into OpenStack as a VM. The third step is to set up a communication channel (here, SSH) to the running VM of the VNF instance and it must be accessible from the API. In the fourth step, the MON module can register the VM's IP into MongoDB and then, MON is ready to use it to send commands and receive data in return. Here, metrics will be obtained through custom or generic commands and a background monitoring feature will be running for each metric. Finally, the policies subsystem will be able to register new custom alerts and compare the obtained VNF's metrics with the alert's condition in order to enable the notification subsystem when the condition is met.

See Annexes II and III to have a bigger picture of the instantiation processes of NS and VNFs, respectively.

## 3.2 Architecture of the monitoring framework

The PALANTIR project is focused on the mitigation of cybersecurity threats, leveraging the NFV architecture. To that end, the Security Orchestrator (SO) subcomponent within the Security Capabilities Orchestration (SCO) component is used. The SO contains the monitoring (MON) and alerting (POL) modules or subsystems. Figure 10 displays the design of SO, where the internal design for the MON and POL modules are highlighted. Other supporting modules are shown, as well as transversal elements, like the database and the Kafka broker, a message/notification bus for notifications.



**Figure 10:** Proposed monitoring and alerting framework, as part of the SCO subcomponent

### 3.3 Design of the solution

To provide the monitoring and alerting functionalities it is necessary to develop APIs, background process and data modelling, among others. Both MON and POL modules must characterize the data in use by each of them, implement the interfaces to be exposed and to interconnect across modules, deploy third-party tools to extract measurement data or trigger notifications, etc.

The functions provided by MON are listed below:

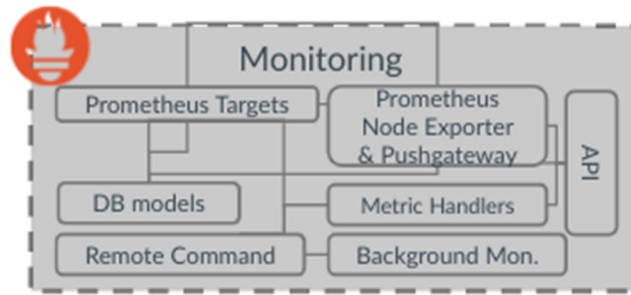
- Registration of NFs to be monitored. The client is able to add, delete and replace the IP or ID of the target NF.
- Listing of the already registered NFs to be monitored, extracted from the database.
- Remote setup of the Prometheus Node Exporter. This can be done in an automatic or manual fashion, installing or uninstalling the exporter into the target NF. The installation process is automatically performed every time a new NF is registered, yet manual installation is also permitted.
- Registration of custom metric to monitor in a VNF. The client (typically the network operator) can request the monitoring of a customized metric to be monitored periodically and registered to the database.
- Trigger of background monitoring on customized metrics. This ad-hoc request starts the background monitoring of a previously registered NF and the metric.
- Listing of all metrics from monitored NFs. This retrieves all metrics from all target NFs that are persisted in the database. While both kind of metrics are stored in the database, it is easy to differentiate the generic metrics from these that are customized.
- Listing of metrics from the Prometheus Node Exporter. This retrieves specifically the metrics of the exporter registered in the database, given a specific target NF.
- Listing of alerts from metrics. This retrieves a list of the registered metrics alerts from the database.

Figure 11 depicts the design for MON, which shows the internal elements and how they relate to each other.

To extract generic metrics from the targets it is possible to use the passive monitoring techniques, using the standard Prometheus Node Exporter which scraps certain metric from a pool of metrics already obtained from the NF running this exporter. To fetch this data from MON, the name of the metric and the ID of the NF is provided. On the other hand, customized metrics can be fetched from a target NF by using remote commands via SSH (using the Paramiko library in Python).

All the data is stored into the MongoDB database and is made available at any time so the network operator can query it. In this database coexist the measurements of the metrics/alerts previously obtained as well as the list of registered NFs.



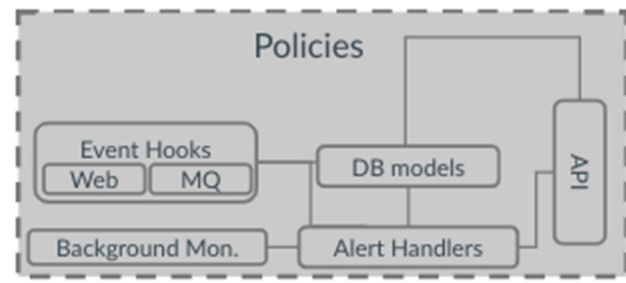


**Figure 11:** Monitoring module architecture

The second part of the monitoring and alerting subsystem is realized by the Policies Management (POL) module, providing the following features:

- Registration of custom alerts into the shared database.
- Listing of all custom alerts. This returns a list of all customized alerts that are registered into the database.
- Listing of metrics related to a given alert.
- Evaluation of the alert and notification triggering.

Figure 12 presents the design of POL's where the background monitoring is activated to obtain the metrics from the target. In this design, the "Event Hooks" element compares the value of the obtained metric with the condition that is already configured and registered in one of the customized alert fields and activates the event if that specific condition is met during a given period.



**Figure 12:** Policies module architecture

### 3.3.1 Guiding principles

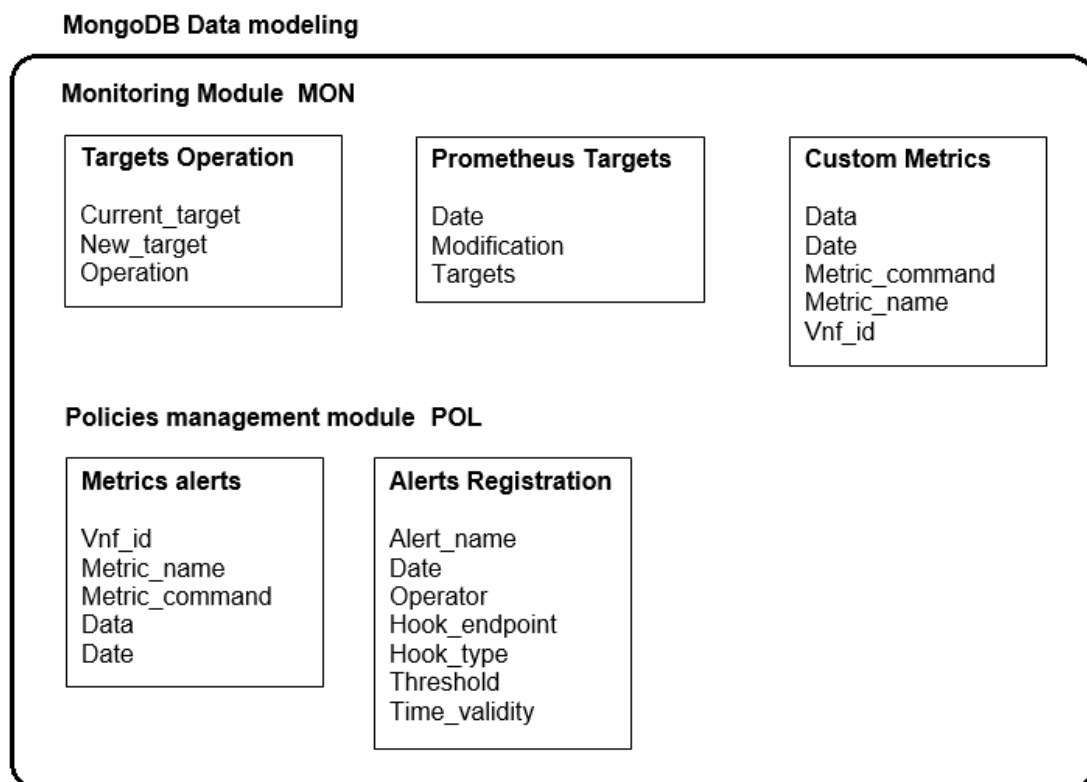
The principles being considered for the design of the solution relate to the flexibility and modularity, as well as the independence between the network operator and the NF developer and between the operator and third-party tools that are not built-in in the encompassing PALANTIR framework. This results in a monitoring and alerting subsystem that can monitor any simple command without imposing specific templating for that, and which is designed to integrate easily into different environments and frameworks, such as other networking stacks and even different MANO solutions.

### 3.3.2 Data modeling

The MON and POL modules work each with a specific JSON data model. This model is used to register or to retrieve data from the database. The data is shown as a document with different fields. To obtain or register any information it is necessary to use REST calls (Annex V elaborates on a complete REST calls guide used in this work) to request the function from the web application. The data modeling for the database entries used by MON and POL is structured as depicted in Figure 13 and described in detail in the Annex I.

There are 3 different data modelling for MON when it comes to store the data in MongoDB: (1) “Targets operation” registers the data that the user wants to modify or add. (2) The user can register a new target, eliminate it, or override it. This model is called “Prometheus Targets”. Finally, (3) the user can register a custom metric “Custom Metrics”.

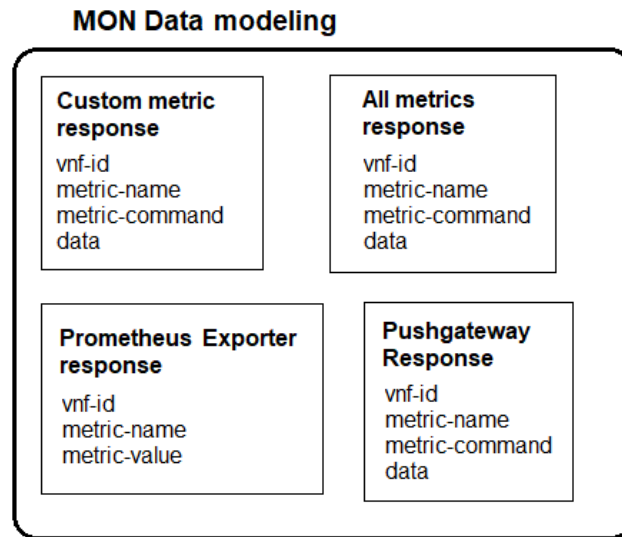
In POL, there are 2 registration data modelling: (1) The registration of an alert related to a specific metric. This model is called “Metrics alerts”. (2) The registration of new alerts with specific conditions and threshold. This model is called “Alerts Registration”.



**Figure 13:** MongoDB data model for monitoring and alert system

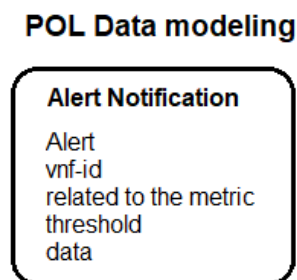
The data modeling for the monitoring and alerting subsystems MON and POL are as shown in Figure 14 and Figure 15 respectively, Annex I also elaborates on these too.

Data modeling for MON basically responds the user with the important data being requested. There are 4 types of requests the user can make: (1) customized metric from a target (VNF), where the response is a data modelling named “Custom metric response”; (2) all metrics from all registered targets, with the response named “All metrics response”; (3) target scraped from Node Exporter, with a response named “Prometheus Exporter response”; and (4) target that is scraped via the SSH channel, the same as for a custom metric request, but with the metric is going through the Pushgateway.



**Figure 14:** Data modeling for MON

The data modelling for POL is dedicated to informing the user about a metric value surpassing the established threshold, previously indicated by the network operator. Once the value of the metric is surpassed, the alert notification is sent and displayed. This response is called “Alert Notification”.



**Figure 15:** Data modeling for POL

### 3.3.3 Interfaces

The interfaces for MON are:

- Prometheus with NF: communicates the NF with the Prometheus Node exporter to obtain the metrics from the NS through the Node Exporter instance and store it in Prometheus.

- SSH channel communicates NF and API to send requests commands and receive the metric.
- MON with database: to retrieve and store data.

The interfaces for POL are:

- POL with database: to retrieve and store data.

Interfaces between MON and POL:

- Background monitoring: communicates MON and POL to obtain registered metric requests and starts the background process.
- Database: both POL and MON share the same database engine (MongoDB) and the virtual network to access it, but the data is properly allocated and organized in different databases and collections inside MongoDB to differentiate them.

## CHAPTER 4. IMPLEMENTATION AND DEPLOYMENT

This chapter describes the implementation and deployment decisions of the MON and POL modules to give the reader a better understanding of how the API works and the reason of the selected tools and elements used for the development of this project.

### 4.1 Implementation decisions

The MON and POL modules dictate a proper implementation of different tools that help the retrieval of metrics and data persistence. It is important to select also an adequate programming language and establish the correct structure for maintainability purposes. The following subsections elaborate on the decisions we have made to ensure the quality of the API and the overall project.

#### 4.1.1 Retrieval of metrics

The following subsections describe the tools used for the retrieval of metrics. Specifically, it elaborates on what these tools are and the use of them into the project.

##### SSH channel

In order to establish a communication channel between the VNF and the application, the Paramiko<sup>2</sup> library is used [23]. This permits the client used by the operator to remotely reach the NF, for this, the key pair has been proportioned – which is configured initially by the operator and is automatically passed during the instantiation to the VIM. Paramiko will be used to execute metric commands to the registered targets in the API, and then receive the metric data to be stored in the database.

##### Prometheus Pushgateway

The Prometheus Pushgateway [24] is a tool that helps to push time series data from live services batch jobs to another intermediary job that Prometheus can easily scrape due to its simple format that contains only relevant information. This tool is used as an alternative path to scrape metrics from the VNF, being useful to obtain only integer values. (Pushgateway deployment is explained in Annex IV)

##### Prometheus Node Exporter

Node Exporter [25] is an important tool of Prometheus, with this we can export metrics from the VNF to the Prometheus Server. It has configurable metric

---

<sup>2</sup> Full python implementation of the SSHv2 protocol with Client and Server functionality

collectors and can measure a variety of server resources such as RAM, CPU utilization or disk space. It can be deployed with Docker into the VNF in an easy way to obtain a set of metrics from it. (Annex IV explains the deployment of the Node Exporter)

#### **4.1.2 Data persistence**

The following subsections explain the implementation decisions in regard of the data persistence processes. For this, the MongoDB and Prometheus were selected.

##### **MongoDB**

MongoDB is a NoSQL database program that has a document-oriented and JSON-like approach [26]. It is useful to store the metrics in JSON format and organizing it in different collections. The access to the data is simple and has libraries that can be added to the MON and POL modules to manage the retrieval and registration of the data.

##### **Prometheus**

Prometheus can store the data from the exporters. When the Prometheus Node Exporter is deployed in the VNF, the Prometheus server scrapes the set of metrics from it so the information can be available to be shown when the user requests any of those metrics.

To acquire more information about the Prometheus deployment, see Annex IV.

#### **4.1.3 Language**

Python is the ideal selection for this project. The principal reasons for this decision are explained next.

##### **Python**

Python is a computer programming language with a simple scripting system that allows the developer to code and build prototypes in a faster way. The evaluation of the code is dynamic, and it can benefit to the maintainability of the MON and POL modules.

#### **4.1.4 Maintainability**

Having a maintainable software is very important to this project due to the possible interchange of client or operator of the system, for this, it is crucial to have the benefits of an easy maintained modules.

Maintainable software allows to quickly and easily fix bugs and add new features preventing the introduction of new bugs, improve usability, increase the performance, fixing structures or code to prevent bugs occurring in the future, make changes to support new environments, tools or operating systems and can easily introduce new developers on board the project [27].

The modularity makes the API easy to use and adapt into other systems. Specifically, the development follows a hybrid approach, combining micro-services and micro-apps. That is, on the one hand, each service (understood as a group of similar functionalities) is provided in its own container, as the micro-service paradigm dictates. On the other hand, shared functionalities (e.g., access to data persistence layer, data modelling, authentication and authorization control) are not replicated per micro service, but deployed as micro-apps, transversal and accessible to the services. While this imposes dependencies across them, it also increases maintainability.

Configuration files and key pairs must be updated regularly in order to improve the security.

## **4.2 Deployment decisions**

The next subsections are related to an important aspect for the development of this project. Deployment decisions were taken to accomplish an efficient use of the resources, future implementations and optimal functionalities. The details of each decision are described below.

### **Usage of Docker for virtualization**

Docker containers were selected for its fast deployment, their independence with the underlying environment (so MON and POL can run not only at development and testing environments, but also to production environments), as well as for its ease of use to define new instances, the possibility of either creating images locally or exporting them to reuse later with a reduced time to deployment (e.g., in production environments), for its easy-to-set virtualized networks on top (which are useful to complement access control between containers), among others.

The Docker-compose tool offers features such as allowing multiple isolated environments on one host. This feature permits the MON and POL modules to consist of more than one service, as well as interconnecting them in a straightforward manner.

### **Coexistence of Docker and Virtualenv deployments**

Typically, Docker is used for production-like deployments, while virtualenv provides a development-oriented environment that accounts for easier modification and quicker redeployment of the different modules. Both MON and POL modules use bash scripts to deploy Docker and virtualenv environments.

## **MongoDB as the principal database**

MongoDB is used for its JSON-type collections data model. This helps to the development of the modules since Python uses JSON-convertible dictionaries. Also, MongoDB deployment helps on reducing resources consumption and avoiding duplicated instances.

## **Prometheus Server and monitoring module MON deployment**

MON is deployed along with the Prometheus Server instance, instead of referencing to an external Prometheus instance. This decision is made to simplify the deployment process and provide a self-hosted environment.

### **4.2.1 Virtualization layer**

As indicated previously, MON and POL shall use Docker to deploy instances and facilitate the transparent deployment disregarding the environment. Kubernetes is also considered for the future in order to take advantage of the same image generation that Docker offers, besides the HA and orchestration features provided by the latter.

### **4.2.2 Access control**

To ensure the minimum access to MON and POL, it is necessary to use proper access control and setup adequate connectivity to them. This control is based on a group of Docker virtual networks that expose proper services and ports to other containers in the same network. In this regard, MON and POL are in a shared network, along with the global API subsystem that receives requests from the users and steers these internally to each relevant subsystem.

### **4.2.3 Configuration**

The configuration of the overall system is centralized. This general configuration will dictate the ports in which the different containers with its services will expose and the access to the database. There are other specific configurations in the subsystem level (i.e., "mon.yaml") where we can configure the instances, scrape intervals, whitelists of commands, etc. This hierarchical structure is made to enable the deployment and to fill data in memory when we initiate the services.



## CHAPTER 5. EXPERIMENTATION AND EVALUATION

This chapter is dedicated to the experimentation and evaluation of the MON and POL modules. In order to analyze the application properly and prepare it for successfully accomplishing the service requirements, it is necessary to subject it to extreme conditions.

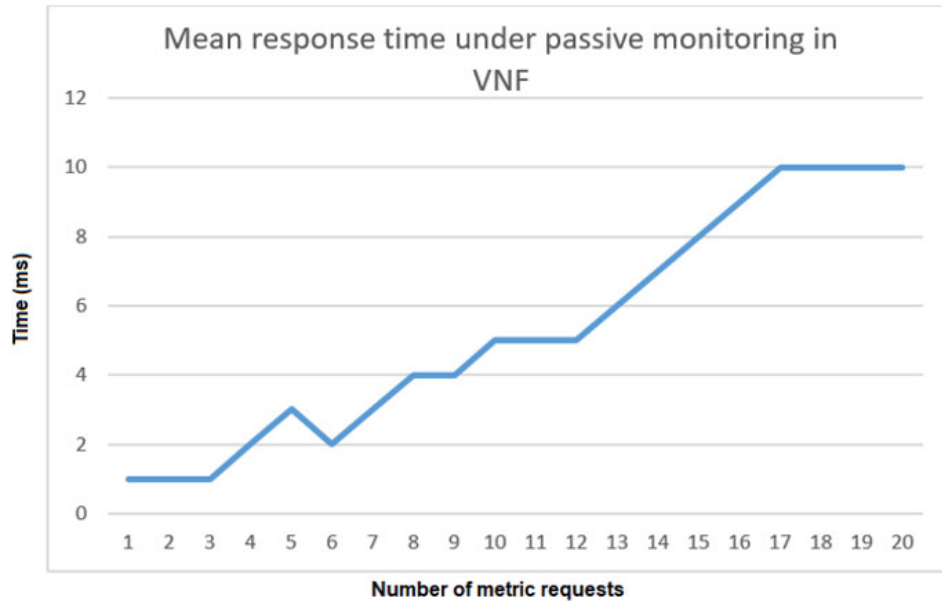
To do so, it is worth mentioning that different factors may affect the results; such as the computer architecture, system's software/hardware components and microprocessor, among compilers, libraries or other tools more subject to the specifics of their implementation. In such cases, the presented results may vary from others performed under different conditions. For the following tests, the MON and POL modules were tested on top of the Linux Ubuntu Operating System, with the 20.04 LTS release. The hardware used featured an x64\_86 Intel core i7-4700MQ @ 2.40GHz with 12GB RAM.

### 5.1 Mean response time

One important indicator of the overall performance of MON and POL is the time taken to fetch and store the data internally, as well as to return it back to the requesting user. To that end, several measurements have been done into the system.

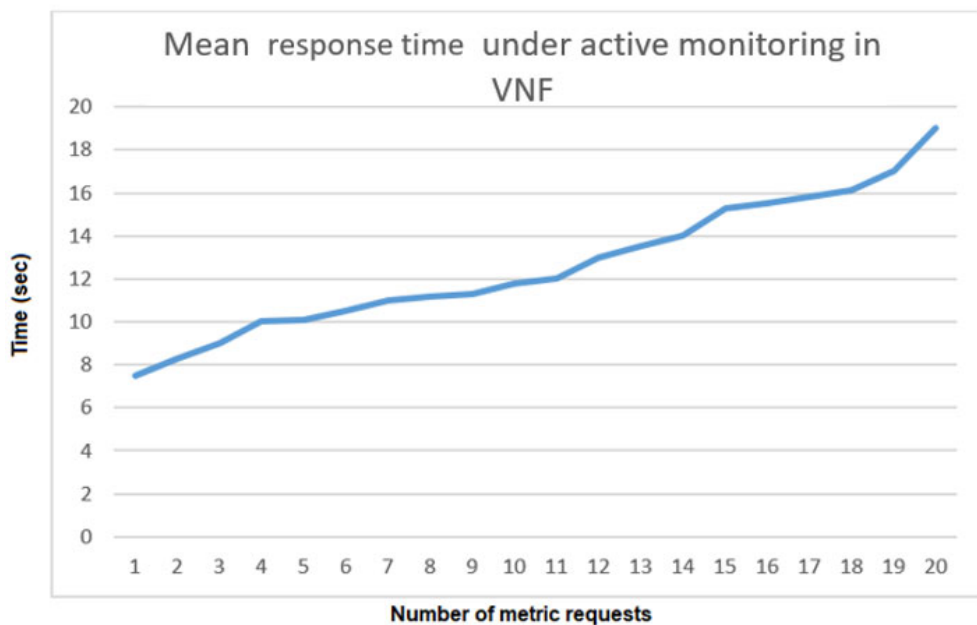
#### 5.1.1 Comparison of mean response time to extract a metric between active and passive monitoring

The first test uses the Prometheus Node Exporter to extract the metric data. This experiment has considered a varying number of metrics (from 1 to 20) requested on a single VNF. Figure 16 shows the comparison between the number of metric requests for active monitoring to the Prometheus node exporter and the time taken, in milliseconds. Each point in the chart corresponds to the average response time taken by five different attempts with the same number of metrics.



**Figure 16:** Mean response time for a metric under passive monitoring (Prometheus)

The second test relies on the SSH channel to execute custom command on the NF and extract its data. Figure 17 depicts the number of custom metrics requested by the operator and the time taken by the system to retrieve such data from each NF from the SSH channel established for the active monitoring. It is important to remark that the metrics requested to the NFs perform different operations and therefore result in different response times across them.



**Figure 17:** Mean response time for a metric under active monitoring (SSH Channel)

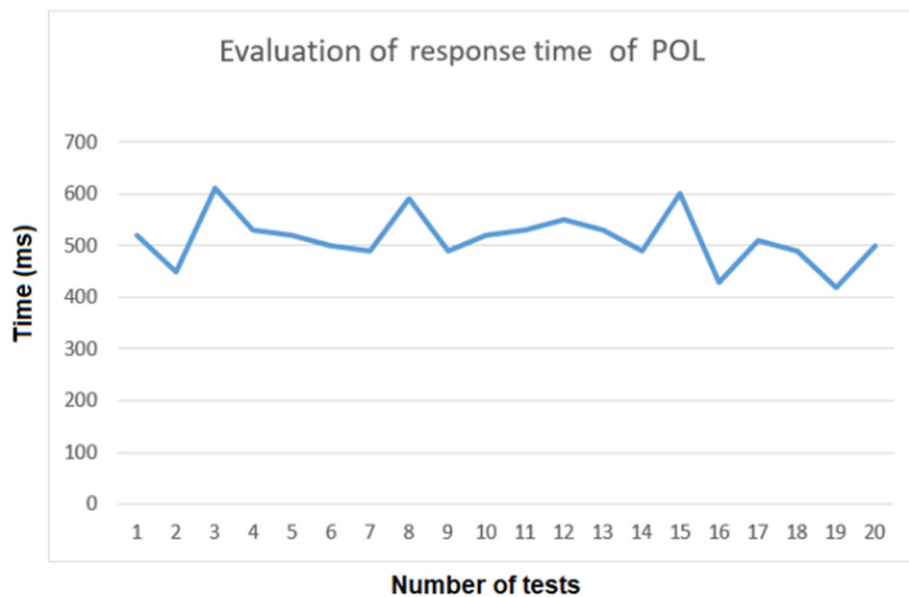
This assessment validates the basic premise that the response time grows proportionally to the number of metrics requested to the monitoring subsystem. The test with passive monitoring, based on Prometheus Node Exporters exposing HTTP endpoints that are polled by the Prometheus Server operate in

the order of milliseconds. This provides a significant reduction in time when compared to the use of ad-hoc, custom metric requests; which run in the order of seconds, as this active monitoring approach requires establishing the SSH channel every time towards each NF.

A possible direction for improvement in this regard relates to the exposure of a lightweight HTTP service within the NF, so that it can extract requested measurements at a system-wide level.

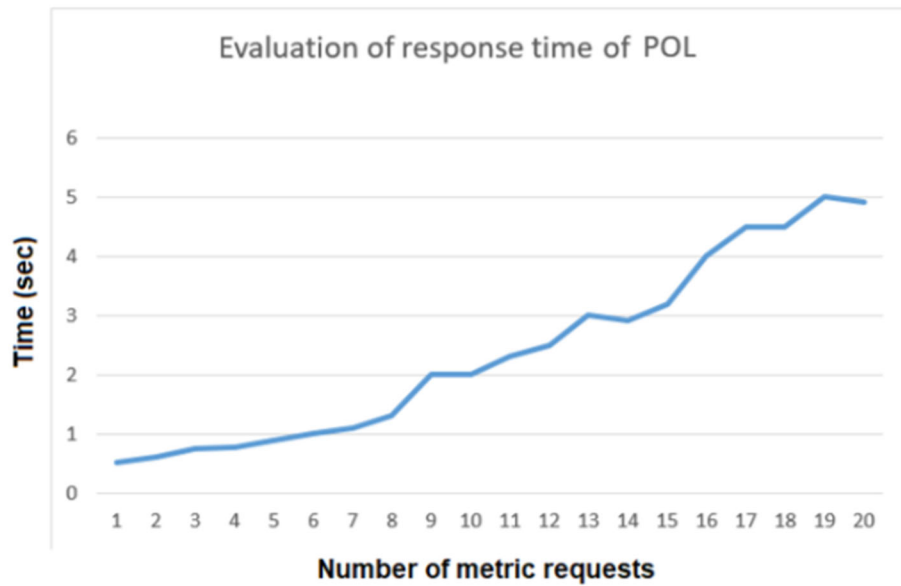
### 5.1.2 Mean response time for the alerting subsystem

To evaluate the performance of the alerting subsystem (POL), the considered time between the detection of an alert (once the metric's value surpasses that defined by the threshold) and the time of the notification's emission is taken. Figure 18 lays out the time it takes for the POL subsystem to send the notification after detecting that a given metric has already surpassed the previously indicated threshold. This test has been executed 20 times with one metric request for one VNF, giving an average time response of 513.5 milliseconds.



**Figure 18:** Mean response time for POL to notify upon detecting a threshold being surpassed

Given that both subsystems are intertwined, the background evaluation process from MON that continuously extracts the metrics can affect that used in POL to evaluate the metrics' status against the alerts' thresholds. Thus, delays in the former have a direct impact in the latter. To measure this, another test analyzed the time it takes for the alerting subsystem to evaluate the metrics in these occasions where the monitoring subsystem has a big amount of metric requests. Figure 19 indicates how the response time increases in proportion to the number of requested metrics that are being monitored.



**Figure 19:** Mean response time for POL as MON increases its observed metrics

As a future work, an improvement in response time of POL should be considered. In this thesis, the communication between the VNF and the API is established through SSH channel tunneling. Another methodology could be implemented to reduce the time issue to the minimum.

## 5.2 Compromise in selected configuration values

There are important configurations to take into account in order to obtain a good performance of the monitoring process and the proper use of the resources. The frequency of metrics acquisition and maximum data retention selections are analyzed next.

### 5.2.1 Selection of the acquisition frequency for metrics

Any monitoring system needs to establish a frequency for the acquisition of measurements ("data scraping"). Such frequency should be the result of a good trade-off between the data scraping time and the system overload. Likewise, the metrics' scrape frequency may also depend on other factors, such as the relevance and how important it is for the operator in a certain time.

The proposed MON and POL modules adopt the scraping time as a configurable value, in the same fashion as Prometheus does. The "MON" configuration file ("mon.yaml") allows setting such frequency as a general value to all monitored targets (i.e., VNFs). This value will be taken as the frequency used by the background monitoring process, in charge of updating the values for the customized measurements requested by the operator.

Table 4 shows the result of different tests taken in one VNF to observe the relation between the scraping time and the Prometheus Overload (% of CPU utilization).

Here, the observed metric requests in MON are increasing. The tests are made with 1, 5, 10 and 20 metric requests at a time to see the consumption of resources on the Prometheus instance (which store the data). To evaluate this, four scraping times were tested with values less or equal to the default scraping time (15 seconds). Specifically, frequencies of 5, 10, and 15 seconds were considered. This table shows that the scraping time has an impact in the resource consumption in the monitoring subsystem.

**Table 3:** Relation between system's scraping time and Prometheus Overload

Scraping time (sec)	Prometheus Overload (CPU Usage (%))			
	1 metric requests	5 metric requests	10 metric requests	20 metric requests
5	1	10	17	28
10	1	8	15	22
15	1	7	14	20

The best result to minimize the CPU usage in the Prometheus instance is achieved when using the default scraping time (15 seconds). The suggested scraping time for MON and POL varies between 5 and 15 seconds, as this is an adequate range to monitor metrics in the VNFs. However, since this is applied to a cybersecurity framework, scraping times should be as low as supported by the network bandwidth, the available disk and the processing capacity to minimize the loss of measurements.

It is worth mentioning that Prometheus has an imposed time limit of 5 minutes before declaring a time series as "stale" (meaning that the data can be discarded or considered as old by that time).

### 5.2.2 Selection of the maximum data retention

Prometheus stores an on-disk time series data under specific directories. These directories can be configured and allows the modification of the retention time to persist the data. It is recommended to configure this retention time based on the available disk space in the system.

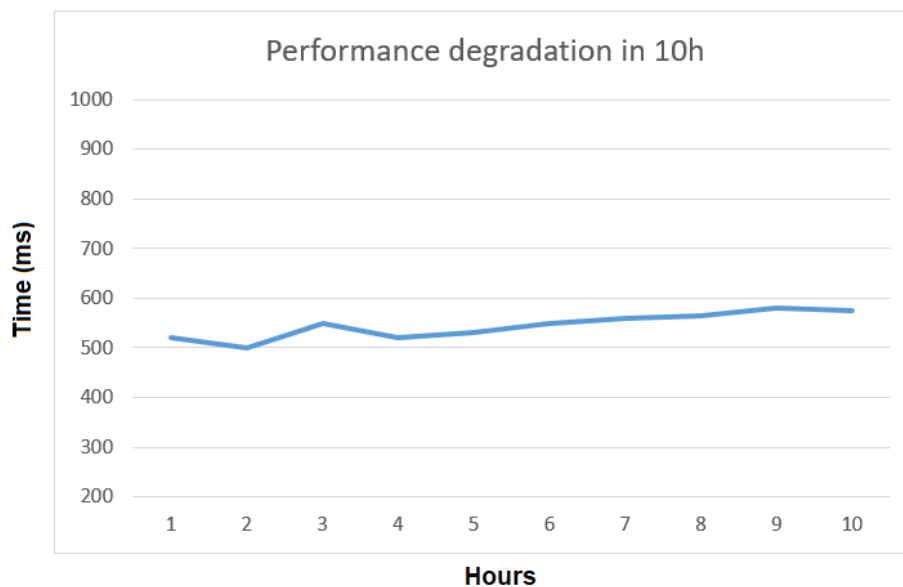
The maximum data retention considered for the monitoring subsystem (MON) depends on the requirements of the client service. The proposed system is expected to be able of protecting the network of micro (fewer than 10 employees) and small companies (10 to 49 employees) considering, on average, up to 10 security-related NFs deployed in their network

In this scenario, local storage is preferred over remote storage due to the extra costs incurred by the public cloud, which small organizations would not be able or willing to pay. On the other hand, local storage imposes some limitations; such as data not being replicated and/or configured in HA mode or even with data loss of overwriting, even if relying on some write-ahead techniques to minimize the potential data loss.

### 5.3 Performance Degradation

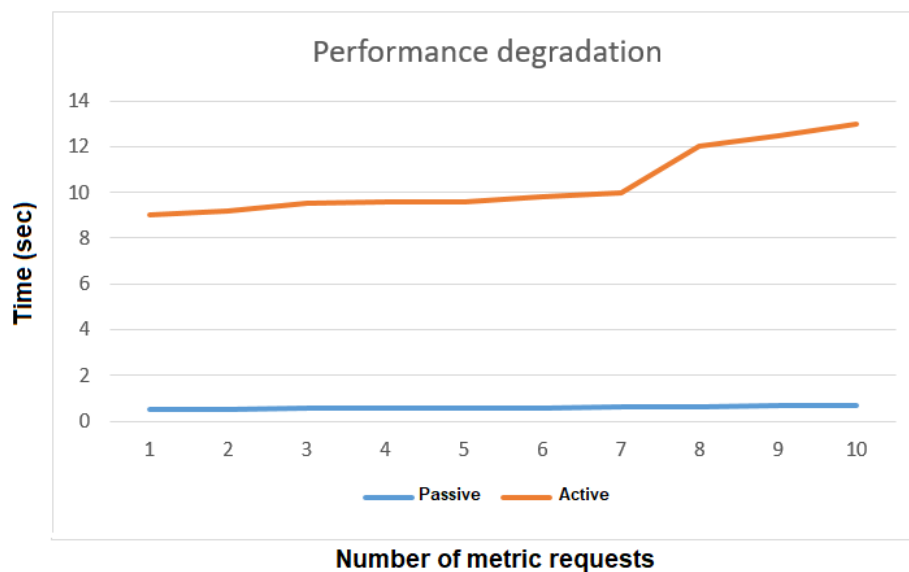
Now, the performance degradation of the MON and POL modules is assessed. This is evaluated when the background monitoring process is running under different circumstances. The first test accounts for the degradation of POL in a range of 10 hours, and the second one considers the continued requests of new metrics to the background process.

Figure 20 shows the performance degradation of POL for 10 hours. The default scraping time (15 seconds) is used, considering one metric requested per VNF. The response time for POL is the average in each hour, which means that the measurements were taken in ranges of one hour and then we obtained the average value. The test indicates that the performance for POL slightly deteriorates throughout this time.



**Figure 20:** Performance degradation of POL for 10h

Figure 21 shows the performance degradation of MON. This experiment has considered a varying number of metric requests (from 1 to 10), which were requested on a single NF with the default scraping time of 15 seconds. The time response of the subsystem is based on a one-time measurement since the background process is running and just one new metric request is added every time. We can observe that the large difference in time remains between the passive monitoring using Prometheus Node Exporter (blue line) and the active monitoring using SSH channel (orange line).



**Figure 21:** Performance comparison between active and passive monitoring

A possible improvement to this loss of performance must focus on the implementation of more efficient and faster communications between the targeted VNF and MON.

## CHAPTER 6. CONCLUSIONS AND FUTURE WORK

The last chapter is dedicated to summarizing the extracted conclusions throughout the document, as well as sharing the upcoming lines of work that were identified to improve the behavior and set of features, impacting the design, implementation and deployment of both MON and POL. Finally, considerations are given on the security, ethics and privacy.

### 6.1 Conclusions of the work

The provided work conforms an adaptable monitoring and alerting subsystem that can be deployed as a separate, modular entity for basic monitoring and notification needs in virtualized environments. Its key objective is to provide a generic and dynamic modelling of metrics and alerting that decouples the monitoring from the service definition.

Active and passive monitoring within the proposed monitoring subsystem shows significant differences and a high variability in the response time. As seen in Figure 17, the time taken to acquire the value of a metric from the NF instance is almost negligible when using passive monitoring (Prometheus), yet its performance degrades significantly when using active monitoring (SSH channel). In the latter case, the time response takes 7 seconds or more for the background monitoring process when the system is stressed (Figure 18); compared to the order of milliseconds incurred by Prometheus, which relies on passive monitoring techniques.

It can be concluded that the usage of passive monitoring, as implemented by the Prometheus Node Exporters exposing HTTP endpoints, which are polled by the Prometheus Server, is a much more efficient option to ensure fast monitoring time responses. The use of ad-hoc requests with customized metrics in the monitoring subsystem, on the other hand, is deemed a more independent and generic monitoring technique that leverages on extremely common tools in the remote servers (i.e., the SSH channel) and account for better flexibility and ease of deployment when it comes down to request customized metrics.

The current implementation comes at the cost of hindering the system performance and must be addressed before moving to a production environment. The performance degradation of the monitoring subsystem is more visible as the background monitoring process is stressed in a continuously fashion, increasing the number of metric requests to server (Figure 21). In the current implementation, the system could have an acceptable performance until certain number of monitored metrics per NF (roughly from 3 to 8), which could be imposed by the operator as a compromise between its monitoring needs and the performance of the monitoring subsystem. The network operator could also tailor the scraping time according to the best compromise of data freshness and network saturation.



## 6.2 Future work

While the main benefit of using the active monitoring in this system is the generalization of the metric retrieval process and customization of the requested metric command, improvements must be applied to enhance the reliability, throughput (e.g., minimizing the response time of MON) and further generalization; as well as the set of provided features.

By observing the results of chapter 5, the throughput and reliability are key aspects to improve. There, the throughput of the metric acquisition (especially that related to custom metrics, obtained through active monitoring means) must be improved to reduce the time to milliseconds, ideally. This could be achieved by following a similar approach to the Prometheus' deployment of internal HTTP servers per NF or even defining and deploying custom Prometheus Node Exporters that encompass the set of required custom metrics defined at any time. The reliability must be also improved, assessing scalability and increasing the availability of the system, both in terms of High Availability (HA) of instances and disk space. This could end up materializing in long-term and/or remote storage and with a hierarchical and distributed monitoring setup to aggregate the data in each network segment.

On the other hand, the monitoring subsystem must be abstracted to allow transparent monitoring in any kind of virtualized nodes, such as Containerized Network Functions (CNFs) which run in Docker or Kubernetes, possibly with an appropriate deployment of the Prometheus Node Exporter and accompanied by system and network adaptations. Requesting commands in non-UNIX virtualized nodes is also a feature to implement for the system to run in e.g., Windows-like virtualized images. The data model shall also be refined for more advanced modelling of custom metrics in the monitoring subsystem, as well as enable the composition of complex thresholds to be met, as an aggregation of different metrics (whether these are generic or customized). Finally, the alerting subsystem should integrate with other traditional communication interfaces for notification, as well as consider integrating with components featuring reaction capabilities, e.g., for automated mitigation response.

## 6.3 Considerations on security, ethics and privacy

The following subsections narrate about the considerations taken on security ethics and privacy.

### 6.3.1 Security

Given the application of the MON and POL modules for cybersecurity purposes, two main aspects are considered: the potentially malicious usage of the

commands to monitor and its applicability to complement the detection of abnormal conditions.

### 6.3.1.1 Potentially malicious custom metrics in active monitoring

The active monitoring process is used to directly fetch user-defined metrics in the form of UNIX commands. As with any user-based input, the possibility of malicious intents cannot be disregarded; and thus, some sanitization or filtering process must take place beforehand executing it in the Operating System.

In order to prevent malicious actions that could put in danger the performance, security or configuration of a VNF, a filtering method is triggered before executing the command given by any metric request. The chosen method is a simple whitelist of allowed commands that are configurable by the network operator in the configuration file for the monitoring subsystem ("mon.yaml").

The list of the allowed commands provided by default in such configuration, as deemed useful for the monitoring of a VNF, are listed in Table 4:

**Table 4:** Whitelist of allowed monitoring commands

Commands for monitoring	Function
acct	User activity monitoring
arpwatch	Ethernet activity monitoring
df	Free disk space available
env	Available environments
free, vmstat	RAM and virtual memory statistics
iftop	Network Bandwidth Monitoring
iostat	Time device activity
iptraf	Real-time IP LAN monitoring
lstat	Input/output statistics
iotop	Monitor Linux disk I/O
ls	List elements in the file system
lsdf	List open files for a process
htop, top	Monitoring of Linux processes
monit	Linux process and service monitoring
monitorix	System and network monitoring
nethogs, netstat	Network statistics
ps	Status of active processes
tcpdump	Network packet analyzer
acct	User activity monitoring

### 6.3.2 Potential to detect Indicators of compromise

Indicators of compromise or IoCs [28] provide evidence of data breach and help determining basic information on how a cybersecurity attack has been done (e.g.,

involved suspicious domains and tools). These can be obtained during the analysis of a cyberattack and then are subject to sharing with other security response teams. This information can be collected from a specific software such as antivirus or antimalware systems.

Some common IoCs that look at the behavior of the system or network are (1) unusual outbound network traffic; (2) activity from unusual geographic areas; (3) unexplained activity by privileged users; (3) substantial increase in read operations on the DB; (4) large and increasing number of authentication failures; (5) increase of requests of critical core filesystem or application's configuration; as well as (6) unexplained and suspicious configuration changes.

The detection of several of these anomalies can be complemented by the monitoring subsystem here presented. For instance, the network or system operator could define specific commands to assess the mean consumption of network data from a given NF, as high ingress traffic could be related with a DoS attack and high egress traffic could be indicative of connection to malicious servers outside. The network operator could, for instance, also help evaluating the trustworthiness of the VM by setting controls that observe the amount, date and type of changes on the core files and emit notifications on suspicious conditions, etc.

In the PALANTIR project, the monitoring subsystem can integrate with other systems to enhance their detection capabilities. One of them is the threat detection system (the Threat Intelligence component), where the monitoring complements its signature and Machine-Learning based detection with other custom measurements. Another one is the SLA detection systems, which can benefit from the identification of specific breaches on the assurance level on the deployed security NS.

### **6.3.3 Ethics and privacy**

Although MON and POL are not devised nor used in a manner that could raise ethic issues (given that the network and security logic processing does not typically need to access any personal data, or even identifiable information); there could be privacy issues to be considered if that was not the case. Specifically, privacy issues could arise if any of these two conditions coexist: (1) any given NF performs behavior inspection from identifiable users; or (2) personal data is available within the NF instance. If so, and assuming the operator knew what to extract, data leaks of personal information could occur.

To mitigate such threats to privacy and honor the main tenets and articles set in the General Data Protection Regulation (GDPR), the NF developer should take extra care not to process or persist personal data unless absolutely needed (fulfilling the data minimization principle) – and if so, adequate security measures must be in place to secure the data. On the infrastructure side, the operator should comply with the principle of least privilege to access and process data, as well as providing similar measures to avoid unauthorized access or usage of personal data.

## **6.4 Considerations on environmental sustainability**

The monitoring and alerting modules designed and developed in this thesis are based on the virtualization concept of NFs over the NFV architecture framework. Such an implementation, heavily relying on virtualization, provides environmental benefits since the underlying system makes it easier to minimize energy consumption by co-locating virtual resources in the same machine and scheduling the usage of resources in the system adequately to maximize performance and minimize consumption. Additionally, NFV aims to minimize the usage of dedicated hardware, which also reduces dedicated computing power.

## ACRONYMS

<b>API</b>	Application Programming Interface
<b>BSS</b>	Business Support System
<b>CAPEX</b>	Capital Expenditure
<b>CNF</b>	Cloud-Native Network Function
<b>COTS</b>	Commercial Off-The-Shelf
<b>DCAE</b>	Data Collection Analytics and Events
<b>EM</b>	Element Manager
<b>ETSI</b>	European Telecommunication Standard Institute
<b>FCAPS</b>	Fault Configuration Accounting Performance Security
<b>IT</b>	Information and Technology
<b>MANO</b>	Management and Orchestration
<b>MON</b>	Monitoring Module
<b>NF</b>	Network Function
<b>NFV</b>	Network Function Virtualization
<b>NFVI</b>	NFV Infrastructure
<b>NFVO</b>	Network Function Virtualization Orchestrator
<b>NMaaS</b>	Network Management as a Service
<b>NS</b>	Network Service
<b>ONAP</b>	Open Network Automation Platform
<b>OPEX</b>	Operational Expenditure
<b>OPNFV</b>	Open Platform for NFV
<b>OSM</b>	Open Source MANO
<b>OSS</b>	Operation Support System
<b>POL</b>	Policies Management Module
<b>QoE</b>	Quality of Experience
<b>QoS</b>	Quality of Service
<b>REST</b>	Representational State Transfer
<b>SLA</b>	Service Level Agreements
<b>SO</b>	Security Orchestrator
<b>TC</b>	Test Controller
<b>TRAM</b>	Test Result Analysis Module
<b>VCA</b>	VNF Configuration and Abstraction
<b>VIM</b>	Virtual Infrastructure Manager
<b>VM</b>	Virtual Machine
<b>VNF</b>	Virtual Network Function
<b>VNFD</b>	Virtual Network Function Descriptor
<b>VNFM</b>	Virtual Network Function Manager
<b>VTA</b>	Virtual Test Agent

## REFERENCES

- [1] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: State of the art, challenges, and implementation in next generation mobile networks (vEPC)," *IEEE Netw.*, vol. 28, no. 6, pp. 18–26, 2014, doi: 10.1109/MNET.2014.6963800.
- [2] P. Greendyk, A. Parikh, and S. Tripathi, "Service platforms," *Build. Netw. Futur. Get. Smarter, Faster, More Flex. with a Softw. Centric Approach*, no. 1, pp. 293–329, 2017, doi: 10.1201/9781315208787.
- [3] V. Sciancalepore, F. Z. Yousaf, and X. Costa-Perez, "Z-TORCH: An Automated NFV Orchestration and Monitoring Solution," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, no. 4, pp. 1292–1306, 2018, doi: 10.1109/TNSM.2018.2867827.
- [4] ETSI, "Network Function Virtualization: An Introduction, Benefits, Enablers, Challenges, & Call for Action," 2012. [Online]. Available: [portal.etsi.org/NFV/NFV\\_White\\_Paper.pdf](http://portal.etsi.org/NFV/NFV_White_Paper.pdf)
- [5] ETSI, "Network Function Virtualization: Architectural Framework," 2013. [Online]. Available: [http://www.etsi.org/deliver/etsi\\_gs/NFV/001\\_099/002/01.01.01\\_60/%0Ags\\_NFV002v010101p.pdf%0A](http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/%0Ags_NFV002v010101p.pdf%0A)
- [6] A. Froehlich, "virtual network functions (VNFs)," 2022. <https://www.techtarget.com/searchnetworking/definition/virtual-network-functions-VNF>
- [7] Faisal, "A Cheat Sheet for Understanding 'NFV Architecture,'" 2015. [https://telcocloudbridge.com/blog/a-cheat-sheet-for-understanding-nfv-architecture/#:~:text=EM \(Element Management \)%3A&text=This is responsible for the,itself can be a VNF.](https://telcocloudbridge.com/blog/a-cheat-sheet-for-understanding-nfv-architecture/#:~:text=EM%20(Element%20Management),%3A&text=This%20is%20responsible%20for%20the%20itself%20can%20be%20a%20VNF.)
- [8] OSM ETSI, "How to Set Up Virtual Infrastructure Managers (VIMs)," 2022. <https://osm.etsi.org/docs/user-guide/latest/04-vim-setup.html>
- [9] R. Hohemberger, A. F. Lorenzon, F. Rossi, and M. C. Luizelli, "Optimizing Distributed Network Monitoring for NFV Service Chains," *IEEE Commun. Lett.*, vol. 23, no. 8, pp. 1332–1336, 2019, doi: 10.1109/lcomm.2019.2922184.
- [10] G. Gardikis *et al.*, "An integrating framework for efficient NFV monitoring," *IEEE NETSOFT 2016 - 2016 IEEE NetSoft Conf. Work. Software-Defined Infrastruct. Networks, Clouds, IoT Serv.*, pp. 1–5, 2016, doi: 10.1109/NETSOFT.2016.7502431.
- [11] Prometheus, "Overview of Prometheus Server," 2022. <https://prometheus.io/docs/introduction/overview/>
- [12] perfSONAR, "perfSONAR," 2022. [https://www.perfsonar.net/gtk\\_what.html](https://www.perfsonar.net/gtk_what.html)
- [13] EDUCBA, "Key differences between Zabbix and Nagios," 2022. <https://www.educba.com/zabbix-vs-nagios/>
- [14] Icinga GmbH, "Icinga 2," 2022. <https://icinga.com/docs/icinga-2/latest/doc/01-about/>
- [15] Altran, "Mano and Monitoring : Two Bumps on the Road To Sdn / Nfv," *White Pap.*, 2018.
- [16] ETSI, "Open Source MANO," 2022. <https://osm.etsi.org/>

- 
- [17] ETSI, “Monitoring and Autoscaling,” 2022, [Online]. Available: <https://osm.etsi.org/docs/user-guide/latest/05-osm-usage.html#monitoring-and-autoscaling>
  - [18] ONAP, “DCAE Architecture,” 2022. <https://docs.onap.org/projects/onap-dcaegen2/en/latest/sections/architecture.html>
  - [19] OPNFV, “Understanding NFV and OPNFV,” 2022. <https://www.opnfv.org/resources/download-understanding-opnfv-ebook>
  - [20] ETSI, “Network Functions Virtualisation (NFV); Assurance; Report on Active Monitoring and Failure Detection,” 2016.
  - [21] W. Jardine, S. Frey, B. Green, and A. Rashid, “Senami,” pp. 23–34, 2016, doi: 10.1145/2994487.2994496.
  - [22] A. J. Gonzalez, G. Nencioni, A. Kamisinski, B. E. Helvik, and P. E. Heegaard, “Dependability of the NFV orchestrator: State of the art and research challenges,” *IEEE Commun. Surv. Tutorials*, vol. 20, no. 4, pp. 3307–3329, 2018, doi: 10.1109/COMST.2018.2830648.
  - [23] J. Forcier, “Paramiko,” 2022. <https://www.paramiko.org/>
  - [24] Prometheus, “Prometheus Pushgateway,” 2022. <https://prometheus.io/docs/practices/pushing/>
  - [25] Prometheus, “Prometheus Node Exporter,” 2022. <https://prometheus.io/docs/guides/node-exporter/>
  - [26] MongoDB, “MongoDB,” 2022. [https://www.mongodb.com/cloud/atlas/lp/try2?utm\\_source=google&utm\\_campaign=gs\\_emea\\_spain\\_search\\_core\\_brand\\_atlas\\_desktop&utm\\_term=mongodb&utm\\_medium=cpc\\_paid\\_search&utm\\_ad=e&utm\\_ad\\_campaign\\_id=12212624563&adgroup=115749706983&gclid=CjwKCAjwquWVBhBrEiwAt1K](https://www.mongodb.com/cloud/atlas/lp/try2?utm_source=google&utm_campaign=gs_emea_spain_search_core_brand_atlas_desktop&utm_term=mongodb&utm_medium=cpc_paid_search&utm_ad=e&utm_ad_campaign_id=12212624563&adgroup=115749706983&gclid=CjwKCAjwquWVBhBrEiwAt1K)
  - [27] S. Crouch, “Developing maintainable software,” 2022. <https://software.ac.uk/resources/guides/developing-maintainable-software#:~:text=More formally%2C the IEEE Standard,adapt to a changed environment.%22>
  - [28] A. Popa, “8 types of Indicators of Compromise (IoCs) and how to recognize them,” 2021. <https://attacksimulator.com/blog/how-to-recognize-indicators-of-compromise/>

## ANNEX I: Data Models

### 1.1 MongoDB data modeling

#### 1.1.1 Monitoring Module MON

**Targets Operation:** The “Current\_target” value indicates the id of the already registered VNF. “New\_target” value indicates the id of the new target to be registered. “Operation” value can have 3 options: (1) add, which adds a new target to the database; (2) replace, which replace an existent target id; and (3) delete, which deletes the target from the database.

**Prometheus Targets:** “Date” is the value of the date used for chronological reference. “Modification” value indicates the type of modification the user wants to apply. “Targets” value indicates the id of the registered targets.

**Custom Metrics:** “Data” indicates the value of the requested metric. “Date” is the value of the date used for chronological reference. “Metric\_command” is the value that indicates the metric command requested to the VNF. “Metric\_name” indicates the name of the requested metric. “Vnf\_id” is the id value of the target.

#### 1.1.2 Policies management module POL

**Metrics alerts:** “Vnf\_id” is the id of the VNF target. “Metric\_name” is the name of the metric related with the alert. “Metric\_command” is the command the user will send to obtain the metric. “Data” is the value of the requested metric. “Date” is the value of the date used for chronological reference.

**Alerts Registration:** “Alert\_name” indicates the name of the new alert registration. “Date” is the value of the date used for chronological reference. The “Operator” could be a simple comparison sign such as >, <, >=, <=, ==. “Hook\_endpoint” is the url where the alert notification will be sent when the threshold is surpassed. “Threshold” is the value that the operator defines as a limit, this will be compared later with the metric value. “Time\_validity” indicates the limit time the alert system should wait before sending the alert notification. If the metric value keeps surpassing the threshold after this time, the alert notification is sent.



## 1.2 MON and POL data modeling

### 1.2.1 MON data modeling

Custom metric response, all metrics response and Pushgateway response has the same data modeling: “vnf-id” indicates the id of the target. “metric-name” indicates the name of the requested metric. “metric-command” indicates the command needed to obtain the metric. “data” indicates the metric value obtained.

**Prometheus exporter response:** “vnf-id”: indicates the id of the target. “metric-name” indicates the name of the requested metric. “metric-value”: indicates the metric value obtained.

### 1.2.2 POL data modeling

**Alert Notification:** “Alert” indicates the alert name. “vnf-id” indicates the id of the target. “related to the metric” indicates which metric is related to this specific alert. “threshold” indicates the value that has been surpassed for the metric value. “data” indicates the value of the metric.

## ANNEX II: OSM

Open-Source MANO (OSM) is the NFVO in use, a framework that oversees the life-cycle management of the NS instances in the NFV Infrastructure (NFVI). The NFVOs work hand in hand with a Virtual Infrastructure Manager (VIM), in charge of managing the virtual resources offered for management to the NFVO. Once the VIM is successfully registered, the NS instances can be deployed with the requested configuration.

The following subsections explain the details and steps for configuration and deployment of the required elements and tools needed by the OSM NFVO in order to manage the lifecycle of the NS instances.

### 1.1 Virtual Infrastructure Manager (VIM) account setup

There are two ways to register a new VIM account: via the (1) OSM CLI and the (2) OSM GUI. The first option is used for this exercise.

The following command registers a new VIM in OSM (with type OpenStack, although other are available). This command provides the parameters that indicate the authentication URL and username (or tenant) and password to access the VIM and access its virtual resources.

```
$ osm vim-create --name <infra_name> --user <infra_tenant> --password ***** --auth_url <infra_url>:5000/v3/ --tenant <infra_tenant> --account_type openstack --config='{security_groups: <allow_from_network_x>, keypair: <mon_keypair>}'
```

In the Figure 22, the new VIM account details are shown:

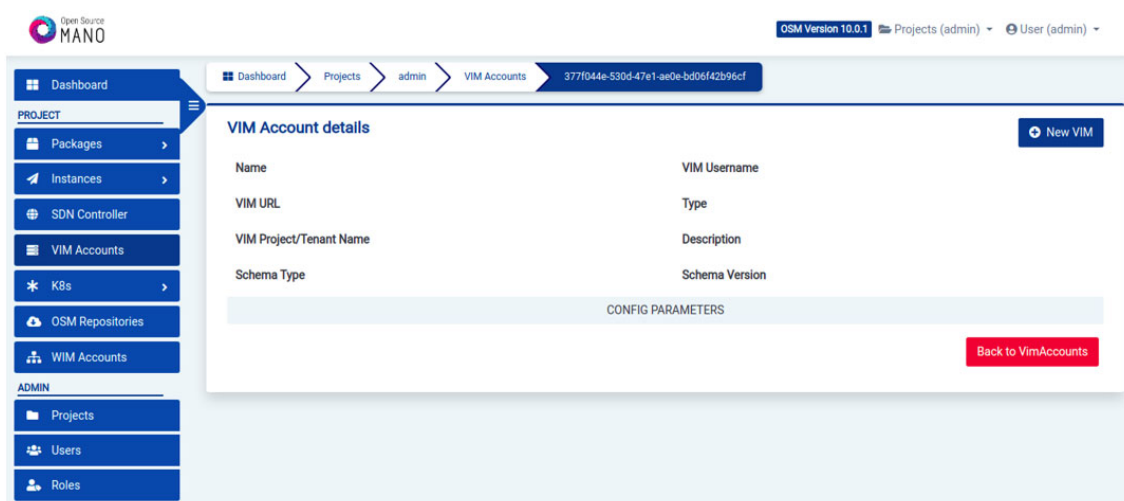


Figure 22: VIM account creation

## 1.2 Network Service (NS) onboarding and instantiation

To instantiate a new Network Service:

i) Create the NS and VNF packages from the set of files that conform each of their descriptors (that is, the logic that indicates OSM how to load and behave when interacting with the packages that model a given service).

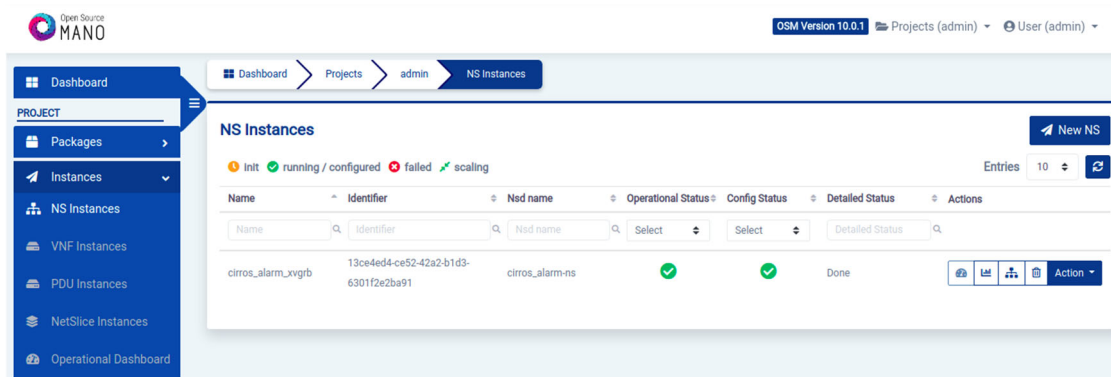
ii) Onboard the VNF first, and then the NS. This is so because the VNF is referenced by the NS and must exist beforehand.

```
$ osm vnfd-create descriptors/vnfs/ping.tar.gz
$ osm nsd-create descriptors /ns/cirros_alarm-ns.tar.gz
```

iii) Instantiate the NS. This step communicates with the VIM to deploy the VNF as a VM in the selected OpenStack VIM.

```
$ osm ns-create --nsd_name cirros_alarm-ns --ns_name cirros_alarm-xvgrb --vim_account test
```

After the onboarding, a successful message will be seen in the OSM dashboard like in Figure 23:



**Figure 23:** NS onboarding

# ANNEX III: OpenStack

OpenStack allows deploying virtual machines easily. It can be linked from the OSM NFVO to deploy the NS instances, as well as deploying any test VM manually, whether using the GUI or the CLI.

## 1.1 Navigating the VMs in use by the VNFs

Once the VM used by the VNF is up and running, the dashboard will show the list of instantiated VNFs in a similar fashion as in Figure 24.

<input type="checkbox"/>	<a href="#">prueba_moin2</a>	Ubuntu 18.04 LTS	mgmtnet 10.10.100.176 OPENVERSO-access1 172.28.2.192	<a href="#">C1_R1G</a>	openverso_keypair	Activo		nova
<input type="checkbox"/>	<a href="#">Test2Instance</a>	Ubuntu 20.04 LTS	mgmtnet 10.10.100.108 OPENVERSO-access1 172.28.2.142	<a href="#">C2_R4G</a>	openverso_keypair	Activo		nova
<input type="checkbox"/>	<a href="#">prueba_moin</a>	Ubuntu 18.04 LTS	OPENVERSO-access1 172.28.2.146 mgmtnet 10.10.100.46	<a href="#">C1_R1G</a>	openverso_keypair	Activo		nova

Figure 24: Instantiated VNFs list

Each row provides information on the VM’s name, image with the Operating System in use and its release, network interfaces along with IPs, the OpenStack flavor (i.e., specifications of CPUs, RAM, disk, etc), the SSH keypair in use, its status and so on. When clicking for details, an exhaustive list is provided, as depicted in Figure 25.

## prueba\_mon2

[Visión general](#)
[Interfaces](#)
[Log](#)
[Consola](#)
[Registro de acciones](#)

<b>Nombre</b>	prueba_mon2
<b>ID</b>	ed996d21-a69f-474d-9cf9-f08a63ac3cf4
<b>Descripción</b>	monitoreo
<b>ID del proyecto</b>	523adf6066804f8e93cfc0369ebdf53a
<b>Estado</b>	Activo
<b>Bloqueada</b>	False
<b>Zona de Disponibilidad</b>	nova
<b>Creado</b>	20 de Agosto de 2021 a las 08:47
<b>Age</b>	6 meses

### Especificaciones

<b>Nombre del sabor</b>	C1_R1G
<b>ID del sabor</b>	b1b5a233-b2df-400e-82cf-d9385270f4fc
<b>RAM</b>	1GB
<b>VCPU</b>	1 VCPU
<b>Disco</b>	0GB

### Direcciones IP

<b>mgmtnet</b>	10.10.100.176
<b>OPENVERSO-access1</b>	172.28.2.192

### Grupos de seguridad

<b>ASIS-AllowfromVPN</b>	PERMITIR IPv6 to ::/0 PERMITIR IPv4 udp from 10.0.0.0/16 PERMITIR IPv4 tcp from 10.0.0.0/16 PERMITIR IPv4 to 0.0.0.0/0 PERMITIR IPv4 udp from 84.88.40.250/32 PERMITIR IPv4 tcp from 84.88.40.250/32
<b>default</b>	PERMITIR IPv4 to 0.0.0.0/0 PERMITIR IPv4 from default PERMITIR IPv6 from default PERMITIR IPv6 to ::/0

### Metadatos

<b>Nombre de la clave</b>	openverso_keypair
<b>Nombre de la imagen</b>	Ubuntu 18.04 LTS
<b>ID de imagen</b>	52319b0e-d72a-446b-b315-cfe7f581b8b0

### Volúmenes asociados

<b>Asociado a</b>	e93774c8-1bbf-4457-8b1d-3d3037b2585d en /dev/vda
-------------------	--

**Figure 25:** VNF specifications

## ANNEX IV: Prometheus

### 1.1 Creating the Prometheus Server (e.g., locally)

1. Locally pull the Docker image of the Prometheus server

Create network to place Prometheus and the monitoring API altogether, so that the latter can access the former

```
$ docker network create palantir-monitoring-network --driver bridge
```

Pull image

```
$ docker pull prom/prometheus
```

2. Create the prometheus.yml file

```
$ cat <<EOF>>prometheus.yml
global:
  external_labels:
monitor: codelab-monitor
scrape_interval: 15s
scrape_configs:
- job_name: prometheus
  scrape_interval: 5s
  static_configs:
  - targets:
  - "localhost:9090"
- job_name: prometheus-exporter
  scrape_interval: 90s
  scrape_timeout: 45s
  static_configs:
  - targets:
  - "${VNF_IP}:9100"
- job_name: prometheus-pushgateway
  scrape_interval: 20s
  scrape_timeout: 10s
  honor_labels: true
  static_configs:
  - targets:
  - "palantir-prometheus-pushgateway:9091"
EOF
```

Right afterwards do change the “targets” under “static\_configs” to replace `${VNF_IP}` with the IP of the VNF you want to read from.

Important: (apparently) when adding/editing/removing new “scrape jobs” (to obtain data from exporters), the Prometheus server instance must be reloaded

Important: when using docker do not use 127.0.0.1, instead use the FQDN/DNS name (the name of the container)

### 3. Run the Prometheus server instance:

Create instance:

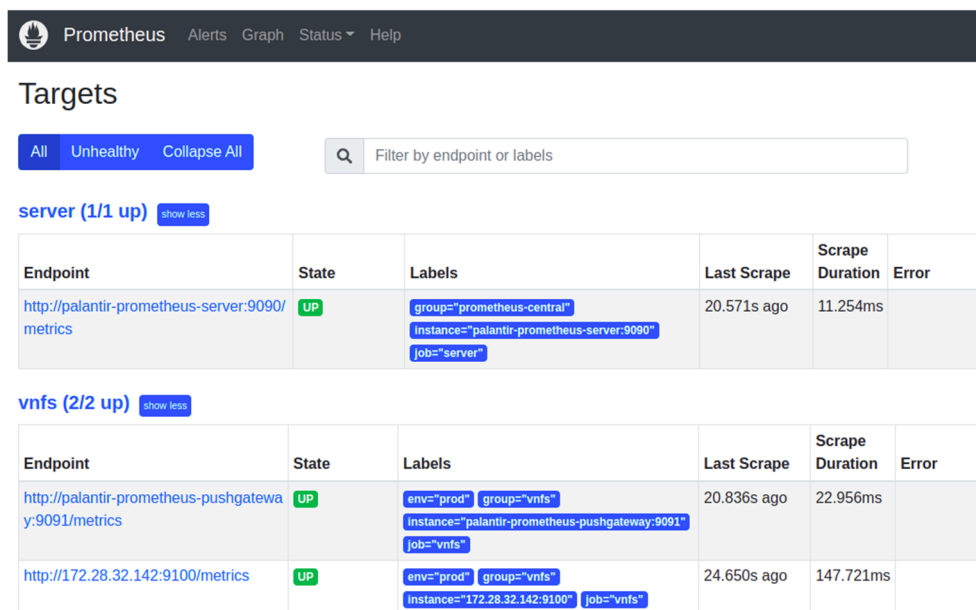
```
$ docker run --name palantir-prometheus-server -itd \
--network "palantir-monitoring-network" \
-p 9090:9090 \
-v ${PWD}/prometheus.yaml:/etc/prometheus/prometheus.yml \
-v ${PWD}/prometheus-targets.json:/etc/prometheus/prometheus-targets.json \
prom/prometheus
```

Important: the /etc/prometheus/prometheus.yml must be overwritten (volume or copied), otherwise any change in the configuration will not work

The Prometheus server has the following endpoints:

- Root UI: <http://127.0.0.1:9090>
- Metrics: <http://127.0.0.1:9090/metrics>

Example with an exporter and a Pushgateway in Figure 26:



Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<b>server (1/1 up)</b> <a href="#">show less</a>					
<a href="http://palantir-prometheus-server:9090/metrics">http://palantir-prometheus-server:9090/metrics</a>	UP	group="prometheus-central" instance="palantir-prometheus-server:9090" job="server"	20.571s ago	11.254ms	
<b>vnfs (2/2 up)</b> <a href="#">show less</a>					
<a href="http://palantir-prometheus-pushgateway:9091/metrics">http://palantir-prometheus-pushgateway:9091/metrics</a>	UP	env="prod" group="vnfs" instance="palantir-prometheus-pushgateway:9091" job="vnfs"	20.836s ago	22.956ms	
<a href="http://172.28.32.142:9100/metrics">http://172.28.32.142:9100/metrics</a>	UP	env="prod" group="vnfs" instance="172.28.32.142:9100" job="vnfs"	24.650s ago	147.721ms	

**Figure 26:** Prometheus Server (Targets)

#### 1.1.1 Prometheus Node exporter

##### 1. Locally pull the Docker image of the Prometheus Node exporter

```
$ docker pull prom/node-exporter
```

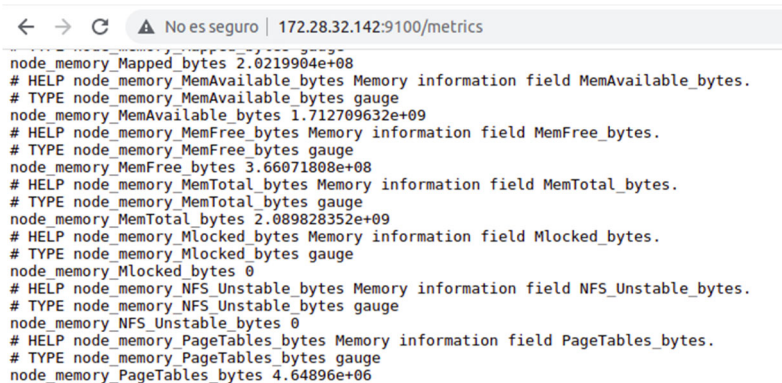
##### 2. Run the Prometheus Node exporter instance:

Create instance:

```
$ docker run --name palantir-prometheus-node-exporter -itd \
-p 9100:9100 \
prom/node-exporter
```

The Prometheus Node exporter has the following endpoints (NB: change IP to the one of the VNF):

- Metrics: <http://127.0.0.1:9100/metrics>



```
node_memory_Mapped_bytes 2.0219904e+08
# HELP node_memory_MemAvailable_bytes Memory information field MemAvailable_bytes.
# TYPE node_memory_MemAvailable_bytes gauge
node_memory_MemAvailable_bytes 1.712709632e+09
# HELP node_memory_MemFree_bytes Memory information field MemFree_bytes.
# TYPE node_memory_MemFree_bytes gauge
node_memory_MemFree_bytes 3.66071808e+08
# HELP node_memory_MemTotal_bytes Memory information field MemTotal_bytes.
# TYPE node_memory_MemTotal_bytes gauge
node_memory_MemTotal_bytes 2.089828352e+09
# HELP node_memory_Mlocked_bytes Memory information field Mlocked_bytes.
# TYPE node_memory_Mlocked_bytes gauge
node_memory_Mlocked_bytes 0
# HELP node_memory_NFS_Unstable_bytes Memory information field NFS_Unstable_bytes.
# TYPE node_memory_NFS_Unstable_bytes gauge
node_memory_NFS_Unstable_bytes 0
# HELP node_memory_PageTables_bytes Memory information field PageTables_bytes.
# TYPE node_memory_PageTables_bytes gauge
node_memory_PageTables_bytes 4.64896e+06
```

Figure 27: Node Exporter Metrics

### 1.1.2 Creating the Prometheus Pushgateway

1. Locally pull the Docker image of the Prometheus server

Create network to place Prometheus Pushgateway, the Prometheus Server and the monitoring API altogether, so that the two latter can access the former

```
$ docker network create palantir-monitoring-network --driver bridge
```

Pull image:

```
$ docker pull prom/pushgateway
```

Run instance:

```
$ docker run --name palantir-prometheus-pushgateway -itd \
--network "palantir-monitoring-network" \
-p 9091:9091 \
prom/pushgateway
```

2. Consider the following to be added later on, in prometheus.yaml from the Prometheus Server instance (under “scrape\_configs”):

```
$ vim prometheus.yaml
- job_name: prometheus-pushgateway
```



```

scrape_interval: 20s
scrape_timeout: 10s
honor_labels: true
static_configs:
- targets:
- "palantir-prometheus-pushgateway:9091"

```

Important: note that the target is pointed using the name of the container. This is required since Prometheus Server will run in another Docker container and these can easily reach others based on their names

3. Install the prometheus-client with pip (pip3 for python3):

```
$ pip3 install prometheus-client
```

4. Code the metric as gauge/histogram/etc values and insert the monitored value into the gateway

```

from prometheus_client import CollectorRegistry, Gauge, push_to_gateway
registry = CollectorRegistry()
g = Gauge("memory_free", "Free memory (MB)", registry=registry)
g.set(2000)
h = Histogram('resource_192_168_1_3', 'Description of histogram for node X',
registry=registry)
h.observe(14.7)
push_to_gateway("localhost:9091", job="monitor_vim_palantir-1", registry=registry)

```

Important: note that the code uses "localhost:9091" instead of the container's name. This is because this code is ran from the local host where Docker containers are running, but outside the Docker networks (which are available to containers only, not to other processes)

5. Metrics can also be inserted with cURL:

Simple (no type defined):

```
$ echo "cpu_utilization 20.25" | curl --data-binary @-
http://localhost:9091/metrics/job/monitor_vim_palantir-
2/instance/172.168.100.1:9000/provider/vim-2
```

Complex (define type, help, etc):

```

$ cat <<EOF | curl --data-binary @-
http://localhost:9091/metrics/job/monitor_vim_palantir-
2/instance/172.168.100.1:9000/provider/vim-2
# TYPE memory_utilization gauge
# HELP memory_utilization Information on how much memory is in use (in MB)
memory_utilization{label="os"} 2000
EOF

```

```
$ curl -L http://localhost:9091/metrics/
```

The Prometheus Pushgateway has the following endpoints:

- Root UI: <http://127.0.0.1:9091>
- Metrics: <http://127.0.0.1:9091/metrics>

The Root UI served by the Pushgateway running dashboard is shown in Figure 28

Pushgateway

Metrics

Status

Help

Runtime Information

Started

2022-04-15 15:48:54.228772036 +0000 UTC m=+0.827618688

Build Information

branch

HEAD

buildDate

20211011-17:51:55

buildUser

root@f68dbd4cbcd

goVersion

go1.16.9

revision

99981d7be923ab18d45873e9eaa3d2c77477b1ef

version

1.4.2

Startup Flags

log.format

logfmt

**Figure 28:** Prometheus Pushgateway dashboard

## ANNEX V: REST calls

### 1.1 Monitoring subsystem MON

#### 1.1.1 Registering VNFs to be monitored

Registration, edition or deletion of one or more targets (i.e., VNFs to be monitored). Requires: target IP/FQDN+port, target port (for Prometheus Node Exporter).

**cURL:**

```
$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X POST
http://127.0.0.1:50106/mon/targets -d '{"url": "target-ip-or-fqdn:9090"}'

$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X PUT
http://127.0.0.1:50106/mon/targets -d '{"current-url": "target-ip-or-fqdn:9090", "new-url":
"10.10.10.11:9090"}'

$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X DELETE
http://127.0.0.1:50106/mon/targets -d '{"url": "target-ip-or-fqdn:9090"}'
```

**Obtained structure:**

Http code response

#### 1.1.2 Listing VNFs to be monitored

Retrieval of the complete list of targets previously registered.

**cURL:**

```
$ curl http://127.0.0.1:50106/mon/targets
```

**Obtained structure:**

```
Http code response
{"results": {"targets": ["target-ip-or-fqdn:9090", "second-target:9100", "third-target:9091", ...]}}
```

#### 1.1.3 Remotely setup the Prometheus exporter (manual)

Install or uninstall Prometheus Node Exporter on given [list of] target[s]. Requires: target IP/FQDN+port.

```
$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X POST
http://127.0.0.1:50106/mon/metrics/node -d '{"vnf-ip": ["target-ip", "second-target-ip", ...]}'

$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X DELETE
http://127.0.0.1:50106/mon/metrics/node -d '{"vnf-ip": ["target-ip", "second-target-ip", ...]}'
```

**Obtained structure:**

```
Http code response
{"results": "Node Exporter Installation status on VNF(s) ['target-ip']: [installed/deleted,
sha256:f2269e73124dd0f60a7d19a2ce1264d33d08a985aed0ee6b0b89d0be470592cd, prom/node-exporter]"}
```

### 1.1.4 Registering custom metric to monitor in a VNF

A customised UNIX-like command can be executed into the target to return its data. Requires: target IP/FQDN+port, custom metric name, custom metric command.

**cURL:**

```
$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X POST
http://127.0.0.1:50106/mon/metrics/vnf -d '{
  "vnf-id": "target-ip-or-fqdn:9100",
  "metric-name": "metric-name",
  "metric-command": "metric-command"
}'
```

**Obtained structure:**

Http code response

```
{ "vnf-id": "target-ip-or-fqdn:9100", "metric-name": "metric_name", "metric-command": "metric_command",
  "data": "data" }
```

### 1.1.5 Triggering background monitoring on new custom metric

Starts a background monitoring process of a custom metric in a target and returns its data. Requires: target IP/FQDN+port, custom metric name, custom metric command.

**cURL:**

```
$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X POST
http://127.0.0.1:50106/mon/metrics/background -d '{
  "vnf-id": "target-ip-or-fqdn:9100",
  "metric-name": "metric-name",
  "metric-command": "metric-command"
}'
```

**Obtained structure:**

Http code response

```
{ "vnf-id": "target-ip-or-fqdn:9100", "metric-name": "__so_pol__metric", "metric-command": "command",
  "data": "data" }, ...
```

### 1.1.6 Listing all metrics from monitored VNFs

Obtains the (filtered) list of all metrics in all targets. Requires: target IP/FQDN+port and/or metric name.

**cURL:**

```
$ curl http://127.0.0.1:50106/mon/targets/metrics?vnf-id=target-ip-or-fqdn:9090&metric-name=metric-name
```

**Obtained structure:**

Http code response

```
{ "results": [
  { "prometheus-node-exporter-metrics": {
    "vnf-id": "target-ip-or-fqdn:9090", "metric-name": "metric-name", "metric-command": "command",
    "data": "data" }
  }
]
```

### 1.1.7 Listing Prometheus Node Exporter's metrics

Obtains one metric of the Prometheus Node Exporter. Required: metric name, target IP/FQDN+port.

**cURL:**

```
$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://127.0.0.1:50106/mon/metrics -d '{
  "vnf-id": "target-ip-or-fqdn:9100",
  "metric-name": "node_metric_name"
}'
```

**Obtained structure:**

Http code response

```
{"vnf-id": "target-ip-or-fqdn:9100", "metric-name": "node_metric_name", "metric-value": "metric-value"}
```

## 1.2 Alerting subsystem POL

### 1.2.1 Listing metrics' alerts

Lists (filtered) alerts registered from the POL module. Requires: target IP/FQDN+port and/or metric name.

**cURL:**

```
$ curl http://127.0.0.1:50106/mon/metrics/alerts?vnf-id=target-ip-or-fqdn:9100\&metric-
name=__so_pol_metric
```

**Obtained structure:**

Http code response

```
{"results": [{"vnf-id": "target-ip-or-fqdn:9100", "metric-name": "__so_pol_metric", "metric-command":
"command", "data": "data"}, ...]}
```

### 1.2.2 Registration of custom alert

Registration of customised alerts. Requires: alert name, type of threshold and comparison operator, validity (seconds) to fulfil the condition before triggering the alert, hook type and details (to send the notification).

**cURL:**

```
$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X POST
http://127.0.0.1:50108/pol/alerts -d '{
  "alert-name": "so_pol_alert-name",
  "threshold": "threshold-quantity",
  "operator": "=", "<=", ">=", "<", ">",
  "time-validity": "time-in-sec",
  "hook-type": "webhook",
  "hook-endpoint": "http://127.0.0.1:50108/pol/notification"
}'
```

**Obtained structure:**

Http code response

```
{
  "alert-name": "so pol alert-name",
  "threshold": "threshold-quantity",
  "operator": "==, <=, >=, <, >",
  "time-validity": "time-in-sec",
  "hook-type": "webhook",
  "hook-endpoint": "http://127.0.0.1:50108/pol/notification"
}
```

### 1.2.3 Listing all custom alerts

Lists all customised alerts registered.

**cURL:**

```
$ curl http://127.0.0.1:50108/pol/alerts
```

**Obtained structure:**

Http code response

```
{
  "results": [
    {
      "alert-name": "so pol alert-name",
      "threshold": "threshold-quantity",
      "operator": "==, <=, >=, <, >",
      "time-validity": "time-in-sec",
      "hook-type": "webhook",
      "hook-endpoint": "http://127.0.0.1:50108/pol/notification"
    }
  ],
  ...
}
```

### 1.2.4 Listing metrics related to a given alert

Retrieves a list of (filtered) metrics related to registered alerts. Requires: target IP/FQDN+port and/or metric name.

**cURL:**

```
$ curl http://127.0.0.1:50108/pol/metrics?vnf-id=target-ip-or-fqdn:9100&metric-name=__so_pol__metric-name
```

**Obtained structure:**

Http code response

```
{
  "results": [
    {
      "vnf-id": "target-ip-or-fqdn:9100",
      "metric-name": "__so_pol__metric",
      "metric-command": "command",
      "data": "data"
    }
  ],
  ...
}
```

### 1.2.5 Triggering background monitoring on new custom metric

Triggers a background monitoring process in MON. Requires: target ID (in OSM), target IP/FQDN+port, custom metric name, custom metric command.

**cURL:**

```
$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X POST http://127.0.0.1:50108/pol/metrics -d '{
  "vnf-id": "target-ip-or-fqdn:9100",
  "metric-name": "__so_pol__metric-name",
  "metric-command": "metric-command"
}'
```

**Obtained structure:**

Http code response

```
{
  "vnf-id": "target-ip-or-fqdn:9100",
  "metric-name": "__so_pol__metric",
  "metric-command": "command",
  "data": "data"
}
```

## 1.2.6 Evaluate alert and trigger notification

Triggers a comparison between the monitored metric data and alerts' threshold/condition, then sends a notification when the conditions are met. Requires: custom alert name, custom metric name.

### cURL:

```
$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X POST
http://127.0.0.1:50108/pol/events -d '{
  "alert-name": "__so_pol_alert-name",
  "metric-name": "__so_pol_metric-name"
}'
```

### Obtained structure:

Http code response

When conditions are met: "target-ip-or-fqdn:9100. Sending alarm to webhook"

**Alert:** \_\_so\_pol\_date1, **vnf-id:** target-ip-or-fqdn:9100, related to the **metric:** \_\_so\_pol\_metric-name, **threshold:** threshold, **data:** data"

When conditions are not met: "Monitoring target-ip-or-fqdn:9100"