Miguel dos Santos Lopes

Garantia de Serviço em Redes 5G

Service Assurance in 5G Networks

Miguel dos Santos Lopes

Garantia de Serviço em Redes 5G

Service Assurance in 5G Networks

"*The greatest challenge to any thinker is stating the problem in a way that will allow a solution*"

— Bertrand Russell

Universidade de Aveiro
2021

Miguel dos Santos Lopes

Garantia de Serviço em Redes 5G

Service Assurance in 5G Networks

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Daniel Nunes Corujo, Professor investigador doutourado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Marco Paulo Viegas Araújo, Investigador Principal na Capgemini Engineering.

o júri / the jury

presidente / president

Prof. Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Bruno Miguel de Oliveira Sousa
Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Prof. Doutor Daniel Nunes Corujo
Professor Auxiliar da Universidade de Aveiro

agradecimentos / acknowledgements

A conclusão da dissertação marca o fechar de um ciclo académico. Ciclo este que, apesar de por vezes ter sido penoso, a gratificação que advém da sua conclusão é o sentimento que prevalece.

Quero desta forma, primeiro, agradecer especialmente a quem tudo devo, à minha mãe, que sempre me deu a mão e apoiou durante todo este caminho, lidou com todos os dissabores durante o meu percurso académico, sem nunca baixar os braços e a mantendo sempre uma atitude positiva. Não deixo de referir também a minha irmã, que também sempre me apoiou nesta viagem.

De seguida, agradecer à Universidade de Aveiro, o Instituto de Telecomunicações e à Capgemini Engineering por terem permitido a realização do presente trabalho.

Por todo o apoio prestado durante esta dissertação, queria deixar o meu agradecimento, ao Professor Doutor Daniel Nunes Corujo por todo o acompanhamento prestado, na qualidade de orientador.

Do lado empresarial, o meu especial agradecimento ao orientador Doutor Marco Viegas Araújo pela constante presença e crítica construtiva que potenciou imensamente o meu crescimento pessoal e profissional.

À restante equipa de Research & Innovation da Capgemini Engineering, por toda a hospitalidade e amizade com que me acolheram, dirigindo também um especial agradecimento ao Mestre Paulo Duarte por ter sido incansável na ajuda que me proporcinou ao longo da dissertação.

À minha namorada e melhor amiga por toda a sua compaixão, carinho e amizade partilhada ao longo de todo o percurso.

Por fim, mas não menos importante, todos os amigos que ganhei e com que partilhei os altos e baixos desta caminhada.

Não havendo mais palavras para expressar a minha gratidão, resta-me dizer, muito obrigado a todos.

Palavras Chave 5G, NFV, SDN, vCDN, Cloud Computing, MEC, 4G, Orquestração

Resumo

A projecção das redes 5G provocaram uma mudança no paradigma das tecnologias que são utilizadas para suportar as redes celulares antecedentes. A quinta geração de redes móveis foi fortemente movida pelos desenvolvimentos tecnológicos na área das redes baseadas em software, tais como, Funções de Rede Virtualizadas (NFV) e Redes Definidas por Software (SDN), sendo estas impulsionadoras na mudança da gestão de insfraestruturas de telecomunicações. Face aos exigentes casos de uso a nível de Qualidade de Serviço (QoS) e Qualidade de Experiência (QoE), é necessária uma adaptação da infrastrutura , descentralizando e colocando esta infrastrutura na extremidade. Aliada ao Cloud Computing, a possibilidade de existência de poder de computação na extremidade, é denominado por Multi-Access Edge Computing (MEC). Utilizando o poder conjunto destas tecnologias para a criação de uma nova geração de redes móveis, é possível desenvolver sistemas adaptados a uma crescente demanda por consumo de conteúdo multimédia na rede, como é o caso de Redes de Entrega de Conteúdo Virtuais (vCDN). As vCDN são redes distribuidas geograficamente com o objectivo de os servidores de acesso às mesmas se situem mais próximos do utilizador, com o objectivo de fornecer uma alta disponibilidade de conteúdos com reduzida latência. Também se diferenciam pela sua capacidade de escalabilidade e flexibilidade, impossível de atingir em sistemas precedentes. Contudo, estes avanços introduzem novos desafios, nomeadamente na automação da rede, gestão e orquestração. Para mitigar e resolver estes problemas, várias plataformas especializadas estão a ser constantemente desenvolvidas e utilizadas, como o Open Network Automation Platform (ONAP), sendo esta de código fonte que providencia suporte de gestão e orquestração de serviços e infraestruturas extremo a extremo. A presente dissertação visa abordar os problemas de QoS de um serviço vCDN, nomeadamente, recolha de dados a nível de hardware e aplicacionais, para que, com recurso ao ONAP, seja possível criar politicas baseadas nos dados, por forma a escalar o serviço de acordo com as necessidades da rede e formar um ciclo de automação fechado.

Keywords 5G, NFV, SDN, vCDN, Cloud Computing, MEC, 4G, Orchestration

Abstract

The projection of 5G networks have caused a paradigm shift in the technologies that are used to support the preceding cellular networks. The fifth generation of mobile networks has been strongly driven by technological developments in the area of software-based networks, such as Virtualized Network Functions (NFV) and Software Defined Networks (SDN), which are drivers in changing the management of telecom infrastructures. In the face of demanding Quality of Service (QoS) and Quality of Experience (QoE) use cases, it is necessary to adapt the infrastructure, decentralizing and placing it at the edge. Allied to Cloud Computing, the possibility of having computing power at the edge, is called Multi-Access Edge Computing (MEC). Using the combined power of these technologies to create a new generation of mobile networks, it is possible to develop systems adapted to a growing demand for multimedia content consumption on the network, such as Virtual Content Delivery Networks (vCDN). The vCDNs are geographically distributed networks with the goal of having the access servers closer to the end user, in order to provide high availability of content with reduced latency. They are also differentiated by their scalability and flexibility, unattainable in previous systems. However, these advances introduce new challenges, namely in network automation, management and orchestration. To mitigate and solve these problems, several specialized platforms are constantly being developed and used, such as the Open Network Automation Platform (ONAP), an open source project that provides management and orchestration support for end-to-end services and infrastructures. This dissertation aims to address the QoS problems of a vCDN service, namely, hardware and application level data collection, so that, using ONAP, it is possible to create policies based on the data, in order to scale the service according to the network needs and form a closed automation loop.

Contents

List of Figures

List of Tables

Glossary

3GPP	3rd Generation Partnership Project
5GS	5G System
A&AI	Active and Available Inventory
APPC	Application Controller
BSS	Business Support System
CAPEX	Capital Expenditure
CC	Critical Communications
CDN	Content Delivery Networks
CLAMP	Control Loop Automation Management Platform
CNFs	Containerized Network Functions
DCAE	Data Collection Analytics and Events
DMaaP	Data Movement as a Platform
E2E	End-to-End
eMBB	Enhanced mobile broadband
ETSI	European Telecommunications Standards Institute
KPI	Key Performance Indicators
MANO	Management and Orchestration
MEC	Multi-Access Edge Computing
MM	Monitoring Manager
MR	Message Router
NFs	Network Functions
NFV	Network Functions Virtualization
NFVI	Network Funtions Virtualisation Infrastucture
NFVO	Network Funtions Virtualisation Orchestrator
ONAP	Open Network Automation Platform
OPEX	Operational Expenditure
oreos	Orchestration and Resource Optimization for rEliable and lOw-latency Services
OSM	Open Source MANO
OSS	Operating Support System
PDP	Policy Decision Points
PNFs	Physical Network Functions
QoS	Quality of Service
SBA	Service-based Architecture
SDC	Service Design & Creation
SDN	Software Defined Networks
SLA	Service Level Agreement
SO	Service Orchestration
TCA	Threshold Crossing Analytics
TOSCA	Topology and Orchestration Specification for Cloud Applications
UE	User Equipment
URLLC	Ultra Reliable and Low Latency Communications
vCDN	Virtual Content Delivery Network
VES	VNF Event Streaming Collector
VFs	Virtual Functions
VF	Virtual Function
VID	Virtual Infrastructure Deployment
VNFs	Virtual Network Functions
VNF	Virtual Network Function

CHAPTER 1

Introduction

1.1 Context

Mobile traffic has suffered a huge evolution over the years and has seen a major increase due to all the new emerging applications and various use cases. In particular, Cisco predicts a growth in volume in video and social networking traffic, which are the already predominant traffic in smartphones and tablets[1]. In line with Cisco findings, Ericsson also predicts that by 2026, 5G networks will carry more than half of the world's smartphone traffic and of which 77% will be from video traffic alone. According to these predictions, it is of utmost importance to carry research in the 5G Network Domain and develop solutions to the 5G use cases[2].

New paradigms in networking, such as SDN, NFV and Multi-Access Edge Computing (MEC) have raised the expectations on future systems built by the Telecom Industry, with the potential offered by each. Concretely, potential Capital Expenditure (CAPEX) reduction enabled by the usage of generic equipment, greater scalability, high flexibility unlocked through network programmability, Operational Expenditure (OPEX)) reduction enabled by automated operation, quick service deployment and update, and even new monetization sources by means of disruptive and attractive business models, are just a shortlist of the opportunities provided.

These concepts became of extreme importance for the industry because of their versatility to help configure and manage networks, which are usually hard tasks to perform due to the high complexity and rigidity of the underlying infrastructure.

Network operators required simpler ways to configure and manage their networks, since networks typically involve integration and interconnection of many proprietary integrated devices, thus causing increased difficulty for operators to specify high-level network-wide policies using current technologies.

SDN, advocates separating the data plane and the control plane, making network switches in the data plane simple packet forwarding devices, and leaving a logically centralized software

program to control the behavior of the entire network, introducing new possibilities for network management and configuration methods.

Compared to the legacy methods, the benefits are clear. First, it is much easier to introduce new ideas in the network through a software program, as it is easier to change and manipulate than using a fixed set of commands in proprietary network devices. Second, SDN introduces the benefits of a centralized approach to network configuration, opposed to distributed management: operators do not have to configure all network devices individually to make changes in network behavior, but instead make network-wide traffic forwarding decisions in a logically single location, the controller, with a global knowledge of the network state[3].

On the other end, NFV decouples the network hardware and software to allow network services to run on commodity cloud computing style platforms. Much in the same way it did for traditional IT, the hope is that virtualization spurs innovation in the network infrastructure industry by enabling faster deployment of new services with less risk, allowing iterative improvement of existing services, broadening the developer ecosystem to include new entrants, and reducing network cost structure through infrastructure sharing and automation[4].

Seeking to maximize the alignment with SDN and NFV, 5G Core follows a Cloud-native design and introduces a service-based model, where network functions interact with other network functions as required through the exposure of service-based interfaces, therefore, shifting from the monolithic architecture models into a microservices architecture[5].

1.2 Motivation

The future success of mobile networks, namely the presently starting fifth-generation, heavily depends on the research and development made towards the objective of satisfying the use cases established by 3rd Generation Partnership Project (3GPP) and European Telecommunications Standards Institute (ETSI).

With this in mind, network orchestration is one of the fundamental pillars for the enablement of the new generation of mobile networks, and Open Network Automation Platform (ONAP) is a key open-source tool, which relies on a community with members from around the world, collaborating to refine the platform, to allow for the orchestration and other functions. As a result, many developments are being constantly made in ONAP, with an ever-growing community, to satisfy the use cases envisioned by the 5G community, and continuous research with this tool is required to truly bring to life the full potential of 5G.

Every day, developers are working to provide more ideas for the successful deployment of 5G network use cases, however, there are still many areas that are not fully explored and that still have a lot of research to do, such as the case of scaling the resources allocated to services that are in use in the network.

The scaling of resources, as a rule, has the objective of improving the service, in the face of possible difficulties that it may be encountering, due to a lack of resources resulting from an overload in the use of the service. This type of strategy is already widely used for traditional IT systems and is now also an objective for network services, thanks to advances in the areas of SDN, NFV and MEC.

To this date, 5G is already beginning to emerge in the commercial field. Nevertheless, there is still a large room for improvement for the solutions that are currently being deployed and research must be conducted in order to improve the network.

In the case of this dissertation, the goal is, therefore, the development of a scaling solution for a service that is operating on the 5G mobile network, the Virtual Content Delivery Network (vCDN).

Given the increasing use of mobile traffic for the consumption of multimedia content, and with an increase in consumer demand for efficient delivery of this content, it is necessary to implement these mechanisms to ensure Quality of Service (QoS) to the customer.

The vCDN service was developed to provide its users with multimedia content for a scenario where the User Equipment (UE) is in motion, more precisely, a moving train.

Since vCDN already makes use of the Open Network Automation Platform, we can then continue to use this framework and explore its features even further. As already mentioned, many developers continue to develop new enhancements for this platform, and one of the expected use cases is precisely the scaling of resources of a running service.

The system, as a proof of concept, was successfully implemented, but the service scaling functionality was not foreseen.

The implementation of this type of functionality in the vCDN service, is based on a monitoring system that, after evaluating the workload of the resources being used by the

system it will, subsequently, possibly detect an overload. It is here that ONAP has an important role because it will be according to this evaluation of the system that it decides if the system needs to be scaled or not.

Thus, the presented solution aims at bridging the gap that is left by the absence of this functionality in the system and use the gathered information, collected with the monitoring system, to capitalize on ONAP orchestration functionality and improve upon the existing vCDN service.

1.3 Objectives

The main focus of this thesis is to expand a virtual Content Delivery Network system, leveraging the power of the ONAP, with a solution that can contribute to maintaining the Quality of Service when it is deployed on the network.

The end goal is to provide a system capable of monitoring a vCDN service and enable to scale-out the system, to satisfy the required load needs. The starting point for this thesis objective is the system that the authors of [6] and [7] developed, where they built a vCDN capable of instantiating nodes along a train track to serve clients that are using the train. Although the system already has some mechanisms for quality assurance implemented, it lacks the ability to scale in computational resources.

To envision and design a proper solution to the proposed problem, first and foremost, an assessment of the State of the Art in the 5G networks and its enablers, followed by a study on the monitoring solutions and the available orchestration platforms, must be made, to have a deep understanding of the technologies used.

The next step will be to study the monitoring framework chosen and ONAP. Once completed, develop a solution that accomplishes the end goal requirements and evaluate the results.

1.4 Contributions

The present work explores the possibilities introduced with the advance towards the fifth generation of mobile networks, more specifically in terms of expanding an existing Virtual Content Delivery Network system, using the ONAP Framework as the orchestrator, to achieve a reliable service assurance through the monitoring of metrics from the system and executing the necessary actions to complete the objective.

From an academic perspective, this work contributes to research that is still fairly new and unexplored, with a lot of work to be done on the orchestration side for 5G networks. It provides insight into how to regulate automatically the systems that are embedded into the networks, using the available features provided by the most recent orchestrators in use. Althought the practical part of the thesis is more pointed towards the demonstration of the orchestration capabilities in a system deployed in the network, this document also provides a theoretical background of the current state of the art on the in topics such as 5G network, NFV, SDN, MEC, orchestration platforms and monitoring platforms.

As a consequence of the work developed, there is also a valuable contribution to the telecommunications industry, in the sense that, the system developed for the execution of the use cases discussed in this thesis can be considered valid conceptual proofs, in accordance with the conclusions drawn, and in this way, the concepts can be executed in real world systems, with the necessary adaptations.

During the development of the practical part of the research, there was also a contribution towards ONAP, by reporting a bug[1] in the Threshold Crossing Analytics (TCA) microservice that caused a critical malfunction at the time.

The resulting work contributes to a successful deployment of a proof of concept system capable of scaling up the vCDN based on the hardware load. Finally, this work also contributed to the Orchestration and Resource Optimization for rEliable and lOw-latency Services (oreos) Project[2].

1.5 Document Structure

The remainder of this document is structured as follows: Chapter 2 presents the State of the Art and the key enabling technologies of the 5G network and a study on the monitoring and orchestration frameworks.

Chapter 3 consists of the research methodology used for designing the developed system architecture, which requirements were necessary for the proposed solution, and its implementation. A more in-depth assessment of the components at work for the considered use case is provided in Chapter 4 Chapter 5 explains the system implementation and the configurations that were use to test it. Chapter 6 presents how the tests were executed and a detailed analysis of the obtained results.

Finally, the showcase of the conclusions of this master thesis and proposed future work are in Chapter 7.

[1] `https://lists.onap.org/g/onap-discuss/topic/84607999#23466`

[2] `https://www.ipn.pt/laboratorio/LIS/projecto/142`

CHAPTER 2

State of the Art and Key Enablers

The following chapter presents the key technologies and concepts deemed essential for the execution of this thesis. It is not a comprehensive explanation of all the details involved in each concept presented but a compilation of all the relevant points that remain inside the scope of the developed work.

2.1 5G Network

The 5G networks aim to revolutionize mobile networks, providing a highly moving and fully connected society. Within the 5G System (5GS), End-to-End (E2E) network slicing, service-based architecture, SDN, and NFV are seen as fundamental pillars to support the heterogeneous Key Performance Indicators (KPI) of the new use cases in a cost-efficient way[8].

2.1.1 High-level Requirements and Use Cases

The 5G use cases demand very diverse and sometimes extreme requirements. 5G promises high speeds and low latency, propelling societies into a new age. The International Telecommunication Union (ITU) is one of the entities responsible for standardizing the fifth generation of mobile networks. The industry stakeholders identified several use cases, and ITU-R has defined three primary use cases and their respective requirements categories: Enhanced mobile broadband (eMBB), Massive machine-type communications (mMTC) and Ultra-Reliable and Low Latency Communications (URLLC) illustrated in Figure 2.1[9].

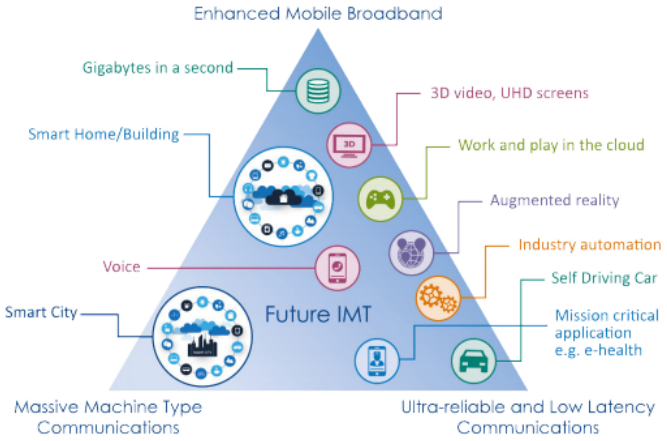


Figure 2.1: 5G Usage Scenarios [9]

In [10], 3GPP defined the 5G requirements in terms of new services and markets, depending on the 5G usage scenario:

- **Enhanced Mobile Broadband (Enhanced mobile broadband (eMBB)):** Requirements specified for data rates, traffic density, user mobility, etc. Various deployment and coverage scenarios are considered, addressing different service areas, such as indoor/outdoor, urban/rural areas, massive gatherings, high user mobility, and many other scenarios.

- **Critical Communications (CC) and Ultra Reliable and Low Latency Communications (URLLC):** These are scenarios driven mainly by industrial automation and require the support of very low latency and very high communications service availability.

- **Massive Internet of Things (mIoT):** Scenarios where the 5G system is required to support very high traffic densities of devices. The Massive Internet of Things requirements includes the operational aspects that apply to the wide range of IoT devices and services anticipated in the 5G timeframe.

- **Flexible network operations:** These are a set of specificities offered by the 5G system. It covers network slicing, network capability exposure, scalability, diverse mobility, security, efficient content delivery, and migration and interworking.

For a more in-depth understanding of all the scenarios envisioned and the required network performance to achieve them, refer to [11].

This diversity of requirements, associated with the different categories of usage described above, enables the use of the 5G System by other industry sectors, referred to as "verticals".

2.1.2 5G Architecture

To support the diversity of use cases and requirements in a cost effective manner, the system design should move away from the 4G monolithic design optimized for mobile broadband and embrace a structural separation of hardware and software. As such, the 5G Architecture is a native SDN/NFV architecture covering aspects from devices, infrastructure, network functions, value enabling capabilities, and all the management functions to orchestrate the 5G System[12].

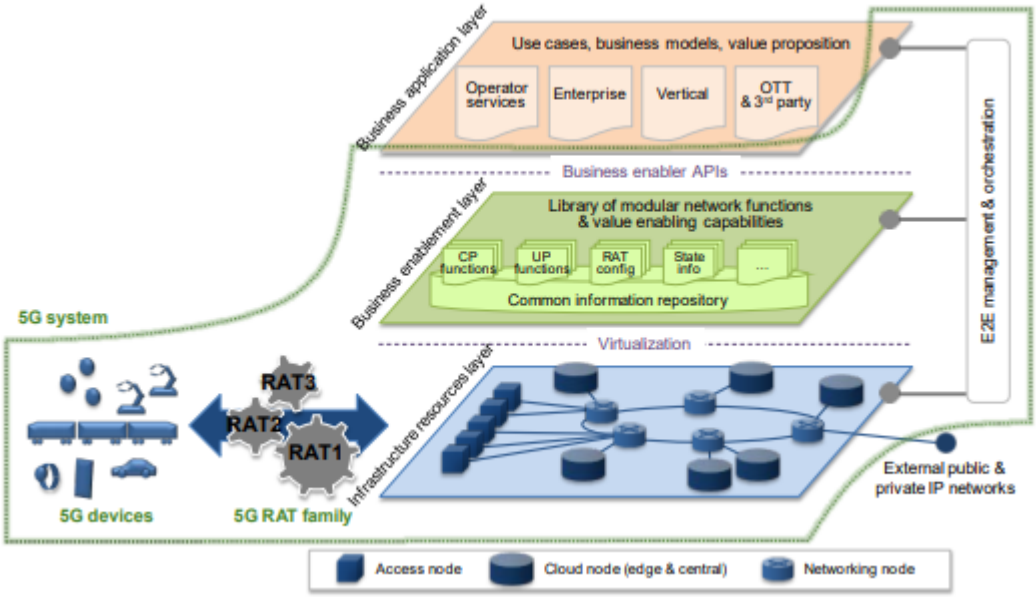


Figure 2.2: 5G Architecture [12]

As illustrated in figure 2.2, four primary components describe the high-level architecture of 5G [12]:

- **Infrastructure resource layer:** consists of the physical resources of a fixed-mobile converged network, comprising access nodes, cloud nodes, 5G devices, wearables, CPEs, machine type modules, networking nodes, and associated links. The resources are exposed to higher layers and the end-to-end management and orchestration entity through relevant APIs. Performance and status monitoring, as well as configurations, are an intrinsic part of such an API.

- **Business enablement layer:** Library of all functions required within a converged network in the form of modular architecture building blocks, including functions realized by software modules that can be retrieved from the repository to the desired location and a set of configuration parameters for certain parts of the network. The orchestration entity calls the functions and capabilities upon request through relevant APIs.

- **Business application layer:** contains specific applications and services of the operator, Enterprise, verticals, or third parties that utilize the 5G network. The end-to-end

management and orchestration entity interface allows, for example, to build dedicated network slices for an application or map an application to existing network slices.

- **E2E management and orchestration entity:** is the contact point to translate the use cases and business models into actual network functions and slices. It defines the network slices for a given application scenario, chains the relevant modular network functions, assigns the appropriate performance configurations, and finally maps all of this onto the infrastructure resources.

2.1.2.1 Service-based Architecture

The 5G network has some fundamental requirements, as seen in section 2.1.2. Based on those conditions, virtualization, and service-based mechanisms are of utmost importance for the 5G ecosystem.

Service-based Architecture (SBA) is the natural step towards empowering 5G network functionality to become more granular and decoupled. Adopting automation and agile operational processes translates into improvements in system integration, reduction in delivery and deployment time, and enhanced operational efficiencies. A service is anatomized capability in the network, with high cohesion, loose coupling, and independent management from other services allowing individual services to be updated independently with minimal impact to other services and deployed on demand [13].

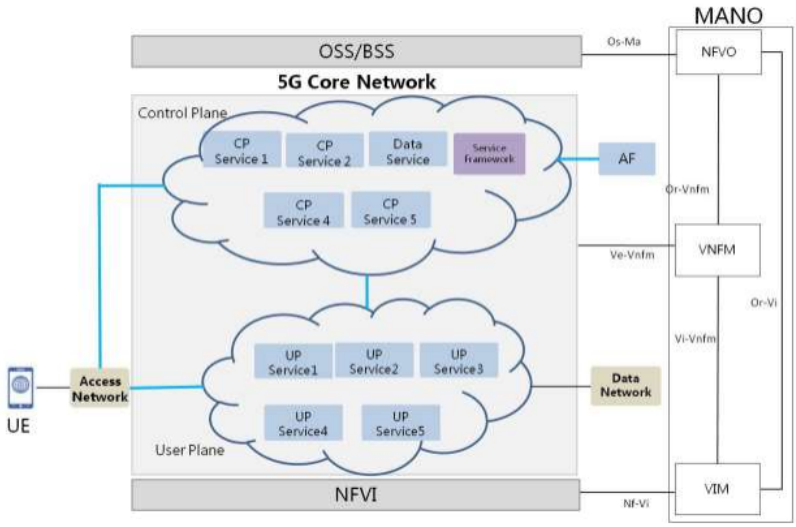


Figure 2.3: Service-based Architecture for 5G Network [13]

The Management and Orchestration (MANO) component, illustrated in 2.3, s the primary focus for the scope of this thesis, which section 2.2.2 explains in more detail. The implementation of MANO may leverage some project or organization, for example, the ONAP.

Thus, SBA enables a virtualized deployment, where the elements that constitute the network will be Network Functions (NFs) rather than monolithic network entities.

2.1.3 Management Control Loops

The management control loop, in figure 2.4, consists on the following steps: Monitoring, Analysis, Decision and Execution. Continuous iteration of the steps in a management control loop completes the adaptation of the resources used for the service[14].

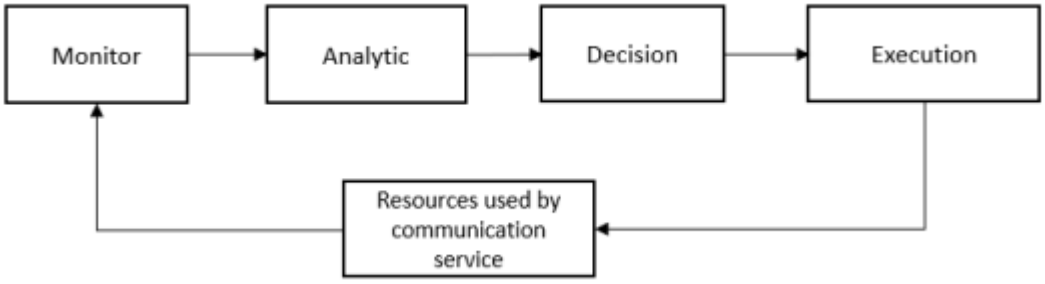


Figure 2.4: Management Control Loop

A control loop is a building block for managing networks and services. There are two types of control loops - closed and open. In the scope of the current work, only the closed control loops are relevant since the decision process will be automated and executed without human operator interference.

In a closed control loop, there is no human factor associated or other management entity in the control loop. Thus the process is completely automated.

Illustrated in 2.5, human interaction is not required to control or manage the process execution inside the control loop itself. Still, it can provide information outside of it, such as configurations for the control loop goals and autonomous decisions. The human factor or any management entity is only required to provide an input to the control loop. The output status also gets forwarded to any other entity.

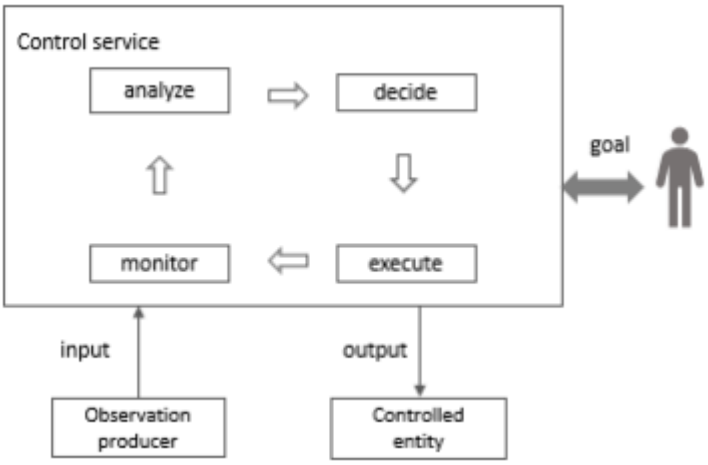


Figure 2.5: Closed loop entities

2.2.1 Software Defined Networks

SDN is an innovative approach to design, implement, and manage networks that separate the network control (control plane) and the forwarding process (data plane) for a better user experience[15], allowing to apply the flexibility of software to the entirety of the networking space. The goal is to enable the development of software, control the connectivity provided by a set of network resources and the flow of the network traffic, offering real-time performance and response to high availability requirements, for example, performing load balancing or fault management for a given system.

It provides limitless options for business relationships, geography spanning the world, and everything from end-user service negotiation and delivery to planning, installing, provisioning, and maintaining the network infrastructure. As well as forwarding traffic, an SDN may process traffic, either as part of added-value services or for service and network maintenance purposes[16][17].

SDN is based on three architectural principles:

- Decoupling traffic forwarding and processing from control
- Logically Centralized Control
- Programmability of network services

In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications.

The SDN Architecture document [17] provides the architecture derived from this vision and established principles, illustrated in 2.6.

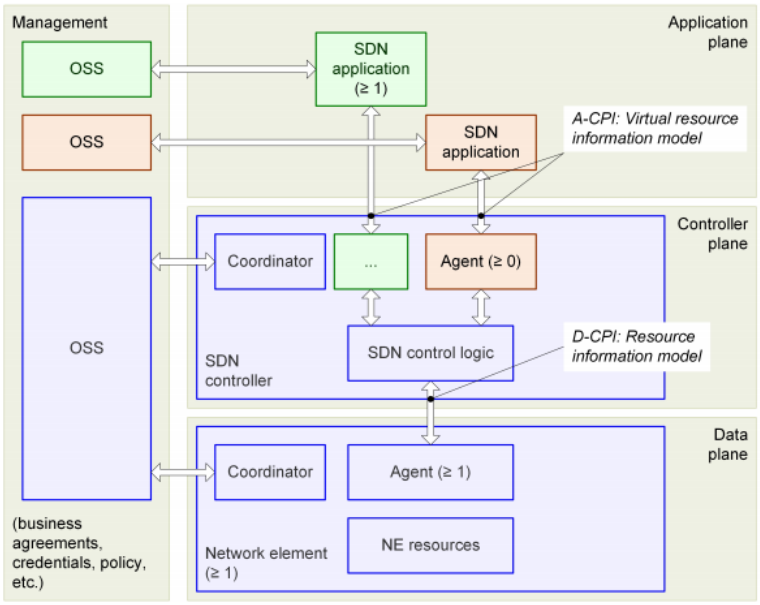


Figure 2.6: SDN Architecture Overview
[16]

Three layers constitute the architecture[16]:

- **Data Plane:** orchestrates the network traffic per the established configuration in the control plane.

- **Control Plane:** Contains the SDN Controller that translates the application's requirements and controls the network elements while providing relevant information up to the SDN applications.

- **Application Plane:** Stores the SDN applications and communicates their network requirements toward the Controller Plane via the A-CPI.

The Management layer also plays a vital role in the architecture. Although many management functions may be bypassed, some are still essential. Management is required to set up the initial network elements and assign resources to the respective SDN controller. n the Controller Plane, it is required to configure the controller and the policies. Lastly, on the application plane configures the contracts and Service Level Agreement (SLA)s, enforced by the Controller Plane.

2.2.2 Network Functions Virtualization

NFV aims to transform how network operators architect networks by evolving standard IT virtualization technology to consolidate many network equipment types onto industry-standard high volume servers, switches, and storage, which could be located in Datacentres, Network Nodes, and in the end-user premises[18].

ETSI NFV Industry Specification Group (ETSI ISG) vision, illustrated through figure 2.7, is that network operators recognize NFV as a critical technology enabler for 5G. The evolved

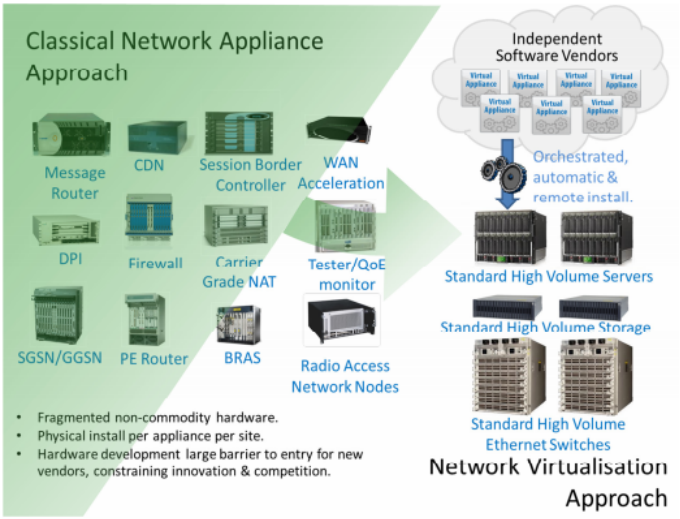


Figure 2.7: Network Functions Virtualisation Vision [18]

5G network will be characterized by agile resilient converged fixed/mobile networks based on NFV and SDN technologies and capable of supporting end-to-end service management across heterogeneous environments [19].

NFV introduces a number of differences, comparing to the current practices [20]:

- **Decoupling software from hardware:** The network element is a decoupled collection of the integrated hardware and software entities, allowing for a separate progression of the two domains.

- **Flexible network function deployment:** The detachment of the software and hardware domains enables the reassignment and sharing of infrastructure resources, thus, allowing software and hardware to perform different tasks at various times. The network function software can become more automated, leveraging the different cloud and network technologies available.

- **Dynamic operation:** Insatiable software components provide greater flexibility to scale the Virtual Network Function (VNF) performance more dynamically.

Regarding the high-level architecture of NFV, three main working domains, illustrated in 2.8, can be identified:

- **Network Funtions Virtualisation Infrastucture (NFVI):** Range of different physical resources supporting the execution of the Virtual Network Functions (VNFs).
- **Virtualised Network Function:** Software implementation of a network function over NFVI.
- **NFV Managment and Orchestration (MANO):** Addresses the orchestration and lifecycle management of physical and software resources that support the infrastructure virtualization and lifecycle management of VNFs.

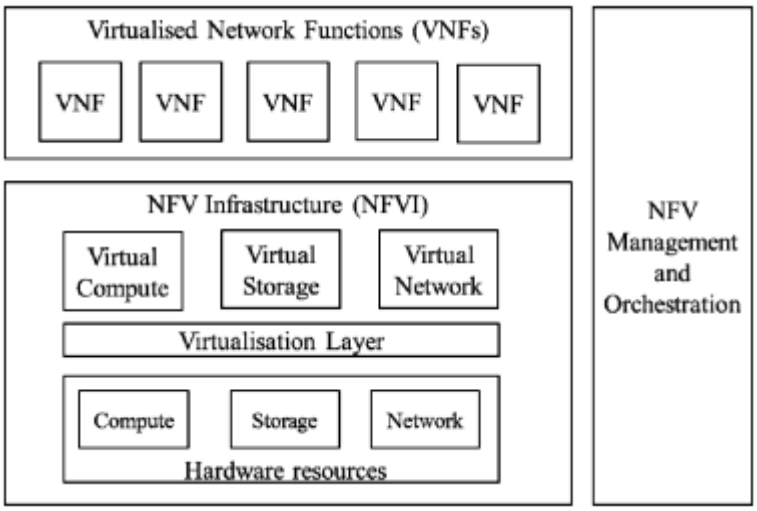


Figure 2.8: High-level NFV framework
[20]

Figure 2.9 represents the three domains that constitute the architectural framework, but also the functional blocks that each domain comprises.

2.2.2.1 NFV Management and Orchestration (MANO)

The Network Functions Virtualisation Management and Orchestration (NFV-MANO) manages the NFVI and orchestrates the resources needed by the Network Services and VNFs. Such coordination is necessary now because of the decoupling of the Network Functions software from the NFVI [21], playing an important role considering service assurance on the 5G Networks.

Three essential functional blocks form NFV Management and Orchestration domain:

- **Virtualised Infrastructure Manager (VIM):** Comprises the functionalities used to control and manage the interaction of VNF with computing, storage, and network resources. It is also responsible for resource management and analytics.

- **VNF Manager:** Responsible for the VNF lifecycle management, such as instantiation, update, query, scaling, termination.

- **Orchestrator:** Responsible for the orchestration and management of NFV infrastructure and software resources and realizing network services on the NFVI.

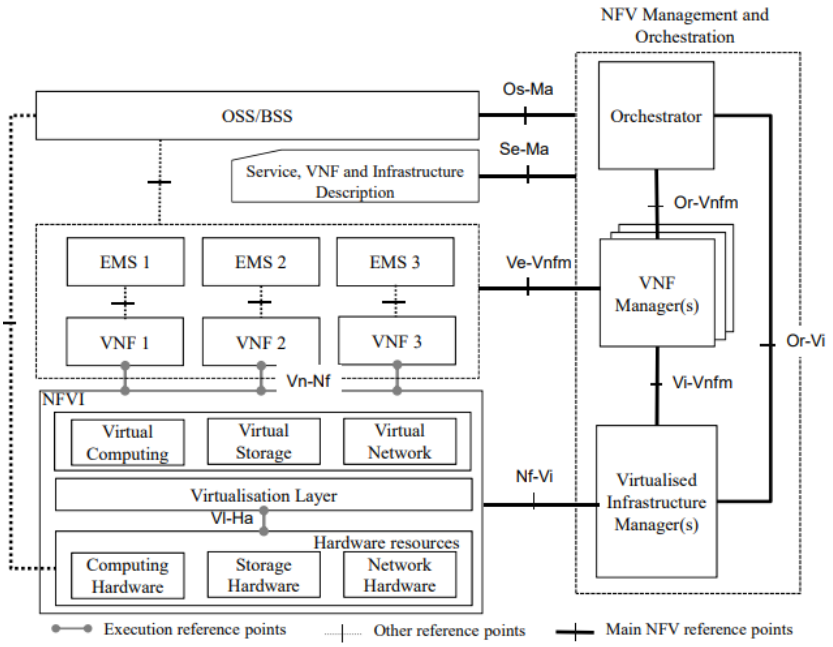


Figure 2.9: NFV reference architectural framework [20]

2.2.3 Relationship between SDN and NFV

Network Functions Virtualisation is highly complementary to Software Defined Networking. If both solutions can be combined, producing a reliable ecosystem, as shown in figure 2.10, they can generate immense value[18].

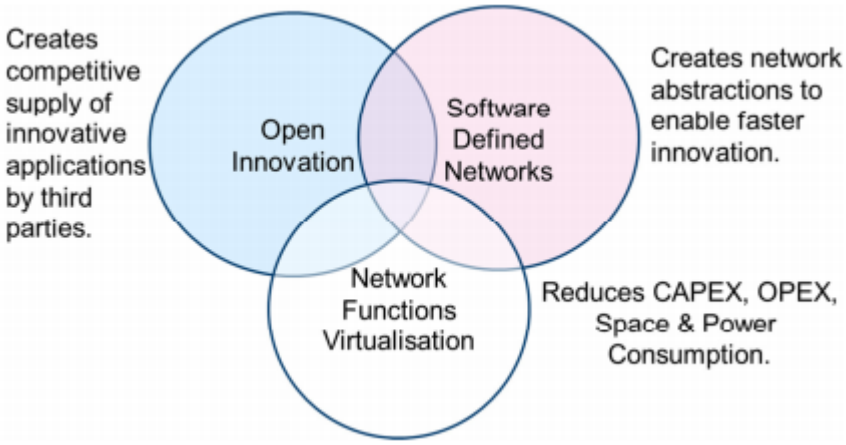


Figure 2.10: Relationship between SDN and NFV

The SDN architecture can be integrated with NFV architectural framework by identifying possible design patterns and associated requirements. There are mainly three SDN components that can be positioned in the NFV architectural framework[22]:

- **SDN resources:** Positioned on physical resources, virtual resources, or as a VNF.

- **SDN Controller:** Positioned on the VIM, on the NFV Infrastructure, as a VNF, or as part of the Operating Support System (OSS)/Business Support System (BSS).

- **SDN applications:** Positioned as part of the VIM, virtualized as a VNF, as part of an EM or as part of the OSS/BSS.

2.2.4 Multi-Access Edge Computing

Edge computing as an evolution of cloud computing brings application hosting from centralized datacentres down to the network edge, closer to consumers and the data generated by applications. Edge computing is acknowledged as one of the key pillars for meeting the demanding Key Performance Indicators (KPIs) of 5G, especially as far as low latency and bandwidth efficiency are concerned[23].

Multi-access Edge Computing enables the implementation of MEC applications as software-only entities that run on top of a Virtualisation infrastructure located in or close to the network edge. The Multi-access Edge Computing framework shows the general entities involved. These can be grouped into system, host, and network level entities[24].

The Multi-access Edge Computing framework is illustrated in 2.11.

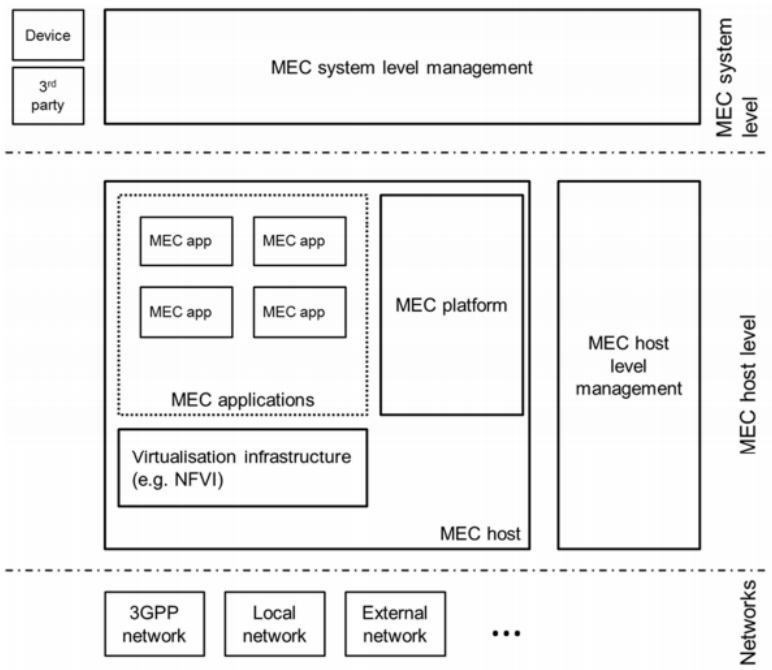


Figure 2.11: Multi-access Edge Computing Framework [24]

The framework consists mainly on the following components [24]:

- **MEC host:** Entity that contains a MEC platform and a Virtualisation infrastructure which provides compute, storage, and network resources, to run MEC applications.

- **MEC platform:** Collection of essential functionality required to run MEC applications on a particular Virtualisation infrastructure and enable them to provide and consume

MEC services and offer them.

- **MEC applications:** Instantiated on the Virtualisation infrastructure of the MEC host based on configuration or requests validated by the MEC management.

- **MEC system level management:** Includes the Multi-access edge orchestrator as its core component, which has an overview of the complete MEC system.

- **MEC host level management:** Comprises the MEC platform manager and the Virtualisation infrastructure manager and handles the control of the MEC specific functionality of a particular MEC host and the applications running on it.

In addition, 3GPP 5G system specifications define the enablers for edge computing, allowing a MEC system and a 5G system to interact in traffic routing and policy control related operations collaboratively. MEC features and these complimentary technical enablers of the 5G system allow integration of these systems to create a robust environment for edge computing.

2.3 Orchestration Overview

The telecommunication operator's service delivery architecture is divided into several layers and distinguishes between upper and bottom layers. The upper layers will be responsible for dealing with products and marketing domains, and the bottom ones are related to the resources needed to support the upper layers. These layers are aggregated in two central systems[5]:

- **BSS:** Encompass all the components that are a part of the service provider business model and define all the interfaces regarding the final consumers. BSS includes order capture & management, customer relationship management, mediation, charging, and billing.

- **Operating Support System (OSS):** Encompass all the components of resource and service management.

The components in this system cooperate in providing infrastructure management, planning, service provisioning, monitoring, assurance, and customer care. In the telecommunications ecosystem, Orchestration refers to the coordinated execution of workflows, consisting of operations on top of services and resources that may be contained in several domains. The concept of End-to-End Orchestration was introduced to refer to multi-domain orchestration. It has a broader scope than the one envisioned by the ETSI ISG MANO specification, which is related to the NFV Orchestration only, referred to in section 2.2.2.1[5].

End-to-End orchestration encompasses the Legacy Network Controllers for specific and non-virtualized technologic domains, SDN Controllers for domains where the control plane is decoupled from the data plane (typically SDN domains), and the Network Funtions

Virtualisation Orchestrator (NFVO), which is the component for NFV domains following the ETSI ISG NFV specification. This concept is illustrated in 2.12.

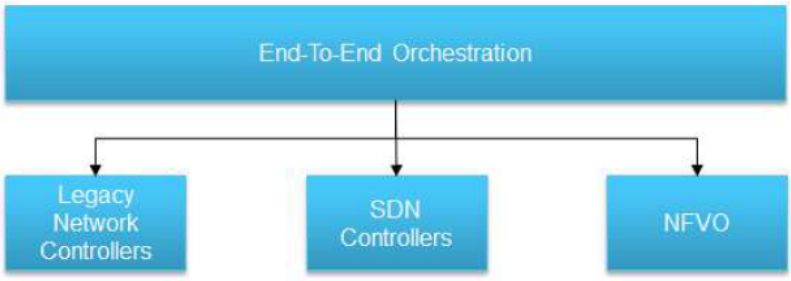


Figure 2.12: End-to-End Orchestration overview [5]

2.3.1 Services and Resources Optimization

As stated in [5], one of the most challenging tasks for network operators is service and resource optimization. At the OSS level, decision-making is based on a set of static rules, being the cause of severe constraints to efficiency by not taking into account the operation's context. Considering the exponential growth and evolution of the 5G networks, according to its architecture, and considering the technological paradigms such as SDN and NFV, if optimization doesn't occur, is expected an even more significant loss of efficiency. The 5G environment will enable services and resources to be highly customizable, making the deployments very context-specific.

In this manner, it is required to address the continuous optimization of services and resources to exploit the benefits acquired by the centralized architecture of the SDN and the efficiency and dynamicity associated with the NFV model. The OSS needs to evolve in parallel with the 5G Networks, and in the core of that evolution, the following points need to be taken into consideration:

- **Services model-driven integration:** LLanguages for service templates are more oriented towards recursivity and reuse of component models, allowing for a more agile approach for service integration and decoupling between service definition and implementation.
- **Analytics driven by AI:** Using cognitive mechanisms in analytics strengthens a richer context-aware characterization of services and resources while fostering learning at the OSS level.
- **Policy-based Management:** Policy systems use declarative policies and intents to enable different teams, ranging from business to engineering, to define policies to govern autonomous processes at the services and resources management layer.

Open Network Automation Platform (ONAP), approached in 2.5.2.1, is an Orchestration Framework that takes into consideration all of these aspects and integrates different data sources, policy systems, and orchestration templates.

2.3.2 Model-Driven Services

A Model-Driven approach focus on building models. Instead of building monolithic network services, the model-driven approach focuses on building generic models, that when are put together, generate the desired network service. The service itself can also be changed since it is a compilation of models and not a standalone approach of the network service.

Network Services will consist of several VNFs, SDN services, or both. VNFs are comprised of modules. In turn, modules consist of Virtual Functions (VFs), virtual links, and ports. One Virtual Function (VF) maps to one virtual machine. SDN services specify underlay (physical) or overlay (virtual) network connectivity, often pertaining to inter-region or WAN connectivity.

The next set of items to get modeled is management data. This includes descriptors, licenses, configurations, and engineering rules[25].

An example of this type of language and one widely used in the ONAP system to define the behavior of some of its components is the Topology and Orchestration Specification for Cloud Applications (TOSCA).

TOSCA [26] is a cloud-centric modeling language. The main goal of this language is to define service templates consisting of different components and the relationships between them, thus, rendering improvements in the management activities of network and cloud applications. The metamodel allows the definitions and description of services by building up a topology and a form to manage it, which is called the Service Template. Figure 2.13 epresents the main elements of a given TOSCA template.

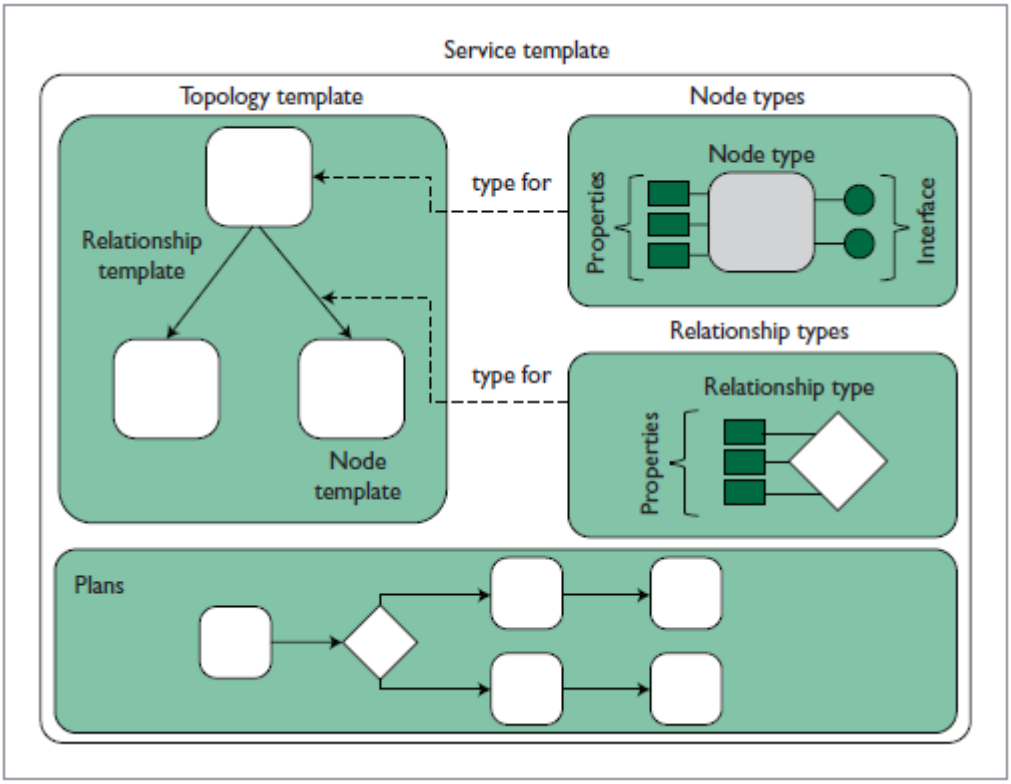


Figure 2.13: TOSCA Service Template [26]

Node Types form a fundamental part of the model and describe service components that can be reused and referred to multiple times when defining the node. The Node Template adds additional information to Node Types, such as how often components can occur or be reused to operate correctly. The Relationship types and Relationship Template act very similarly to the Node Types and Template but represent the dependencies created between Node Types. Node Templates and Relationship Templates combined to form the Topology Template, creating a collection of specific nodes and relationships to give all necessary information pertaining to the described service. Together with the Plans, it generates the full Service Template. Plans are used to manage service lifecycles and provide data to create a running instance of the specific service.

In ONAP [27], TOSCA is used by Service Design & Creation (SDC) o describe services, resources, and their relationships. SDC offers the possibility of adding new TOSCA models to define new types of resources through APIs, UI, or running custom scripts. After uploading TOSCA models, SDC stores these new resource types in the Catalog and makes them available to designers.

2.4 Virtual Content Delivery Network

Content delivery is a breakthrough technology allowing enterprises to distribute a new generation of scalable, accelerated, rich Web-based content, including TV-quality streaming media, for e-business and knowledge-sharing solutions. Content delivery replicates content

to the "edge" of the network, minimizing the distance from the point at which content is requested and where it is served[28].

Content Delivery Networks (CDN) have evolved to improve user perceived Quality of Service when accessing Web content. A CDN creates replicas of the original content into distributed cache servers closer to users, allowing content to be delivered to end-users more reliable and timely [29].

With these mechanisms in place, CDNs boost overall network performance by improving accessibility and ensuring content accuracy with replication.

A CDN usual functions are as follows [30]:

- **Request redirection and content delivery services:** Direct a request to the closest suitable CDN cache server using mechanisms to bypass congestion.
- **Content outsourcing and distribution services:** Replicate and cache content from the origin server to the web servers.
- **Content negotiation services:** Attend the specific needs of each user.
- **Management services:** Manage the network components, handle accounting, and monitor and report content usage.

In figure 2.14, the general architecture CDN is presented.

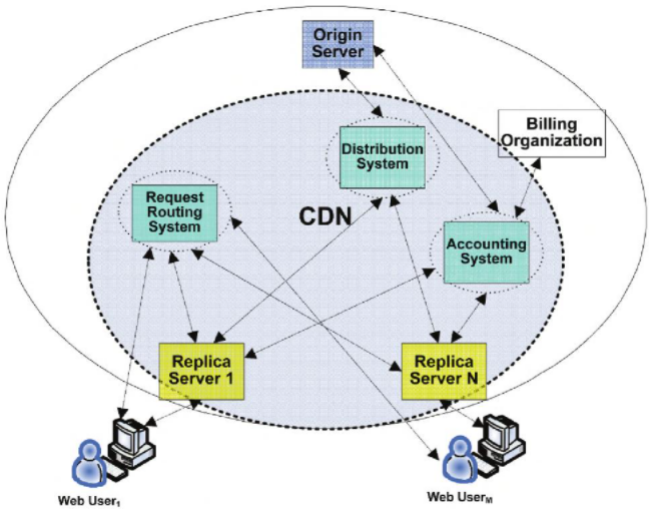


Figure 2.14: Architectural components of a CDN [30]

The content delivery-component consists of the origin server and a set of replica servers responsible for delivering the appropriate content copies to the requesting users.

Working in conjunction with the delivery component is the request routing component, the distribution component, and the accounting component, which are in charge of directing clients requests to the proper edge servers, moving the content from the origin server to the edge servers, and ensuring consistency in the caches, and, maintain a log of client accesses and records the usage of the servers, respectively.

The CDN architecture requires a set of supporting services and capabilities, in addition to the deployment of replica servers at the network's edge.

The edge servers must be installed at various geographically dispersed locations to be effective for many users and cover broad areas.

The creation of a CDN requires the following key components[31]:

- **Replica placement mechanisms:** Needed to decide the replica server locations and adaptively fill them with appropriate content before the customer's requests arrival, making the servers pro-actively updated.
- **Content update mechanisms:** Required to automatically check for changes on the host site and retrieve updated material for consumption at the network's edges.
- **Active measurement mechanisms:** Enables access routers to have immediate access to a real-time picture of the Internet traffic to recognize the fastest route from the requesting users to the replica servers in any traffic situation.
- **Replica selection mechanisms:** Enables access routers to accurately locate the closest and most available edge server from which the end users can retrieve the required content.
- **Re-routing mechanisms:** Provide a way to quickly re-route content requests in response to traffic bursts and congestion as revealed by the measurement activity.

With the panoply of instrumentation that CDN offers and with help from the MEC technology when the service is instantiated as a vCDN, it can operate across a range of virtualized infrastructure, from Core data centers to every single Edge data center. It can fully support all types of virtualized network function deployment at different operator network vantage points, further improving the QoS provided by the CDN.

2.5 Technical Tools

In this section, the technical tools chosen for the development of this work are explored, more precisely, the monitoring tools and the orchestration platforms. It is also presented a study on other platforms and frameworks that were considered, and a brief explanation on the decision process behind determining which one to use.

2.5.1 Monitoring Solutions

2.5.1.1 Prometheus

Prometheus is an open-source, metrics-based system used for event monitoring and alerting, relying on a robust data model and query language for providing the required metrics to precisely analyze infrastructure and application performance.

The system provides libraries for the most popular coding languages, such as Python, Java, Go, C#, to develop custom applications. Software such as docker and Kubernetes are already instrumented in the Prometheus client. For third-party software that exposes metrics in a data format not recognized by the Prometheus scrapper, exporters are already available and public to make the necessary conversions to the known structure. These exporters provide metrics for environments that can possibly be running, for example, applications using MySQL, PostgreSQL, SNMP, Kafka, et cetera[32].

The main features of the monitoring tool are the following[32]:

- Multidimensional data model with time series data identified by metric name and key/value pairs
- Flexible Query Language to leverage dimensionality (PromQL)
- No reliance on distributed storage; single server nodes are autonomous
- Time series collection via a pull model over HTTP
- Pushing time series supported via an intermediary gateway
- Targets discovered via service discovery or static configuration
- Multiple modes of graphing and dashboarding support

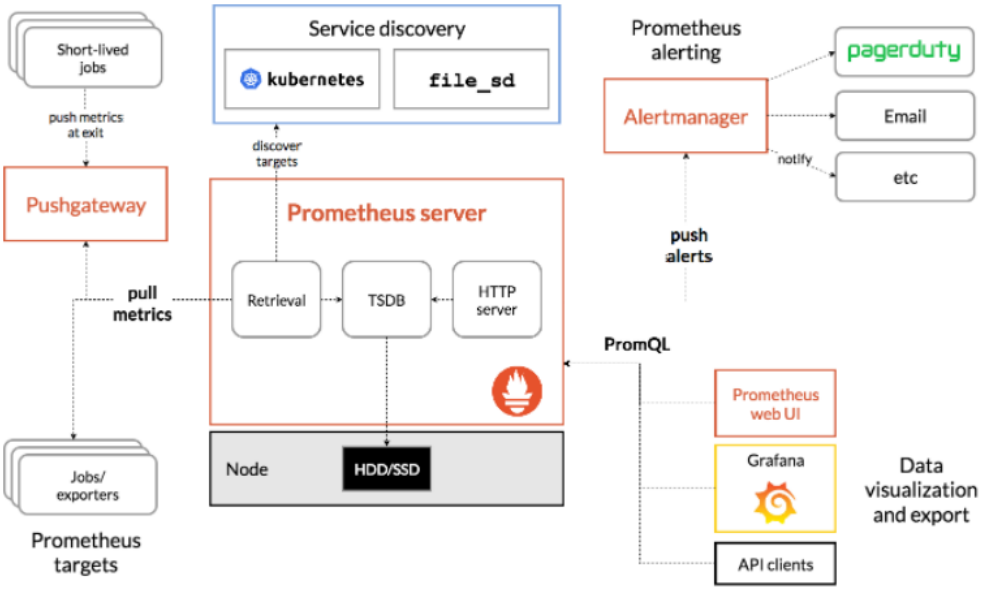


Figure 2.15: Architecture of Prometheus and some of its ecosystem components

Prometheus scrapes metrics from instrumented jobs/exporters directly, based on the target provided by the Service Discovery. For components that cannot be scraped directly, the Prometheus Pushgateway is used to allow short-lived jobs to be pushed into an intermediary job, which Prometheus can scrape.

All scraped samples are stored locally, and Prometheus runs rules over this data to either aggregate and records new time series from existing data or generate alerts. API consumers can be used to visualize the collected data.

In summary, Prometheus provides the following components for a complete monitoring solution [32]:

- Prometheus server to scrape, store and centralize time series data

- Client libraries for instrumenting application code available in programming languages such as Go, Python, Java/JVM, Ruby, .NET, Node.js, Haskell, Erlang, and Rust, to allow the developers to define metrics and add the desired instrumentation to the application code.

- A Service Discovery mechanism, such as Kubernetes, EC2, or Consul, provides the target system to monitor.

- A Push gateway that scraps metrics from applications and passes on the data to Prometheus when the pull method is not possible.

- Multiple special-purpose exporters that typically run on the monitored host to export local metrics, since not always will it be possible to add direct instrumentation to the monitoring application and Prometheus deals with this problem with the aid of exporters, which are pieces of software that will run alongside the target application which the developer wishes to collect metrics.

- An AlertManager tool that receives alerts from Prometheus servers and turns them into notifications. Related alerts can be aggregated into one notification, throttled to reduce pager storms, and different routing and notification outputs can be used configured for each of the different teams monitoring the system.

- A GUI dashboard such as Grafana for reporting purposes facilitates data analysis, especially when the scope is to assess metrics.

2.5.1.2 Other Solutions

There is a vast panoply of offers concerning monitoring solutions for infrastructures that enable it. The most common, besides Prometheus, are influxDB [1], Nagios [2], et cetera.

InfluxDB is only a time-series database, while Prometheus is a fully integrated time-series database and a monitoring system. Similar to Prometheus, it also supports the connection to specific plugins that allow system monitoring.

Although both projects are very similar in their applicable use cases, both open-source and have ample community support, there are some differences to consider to mitigate a higher degree of complexity that the inFluxDB system can introduce. Each Prometheus server runs independently on the system it is deployed and relies only on their local storage for their core functionality[33].

The open source version of inFluxDB acts in the same manner, but the commercial version can make the system management overly complicated because it will have a distributed storage

[1] `https://www.influxdata.com/`
[2] `https://www.nagios.org/`

cluster with storage and queries being handled by many nodes at once. It becomes easier to scale horizontally but adds the complexity of a distributed storage system.

Prometheus ends up being a simpler alternative to run, even needing to shard servers when scalability is necessary. Independent servers running in parallel offers better reliability and failure isolation. Prometheus also ends up being a better alternative than inFluxDB, when the use case is primarily focused on reading metrics, which is the case in this thesis.

Nagios and Prometheus offer different functionalities. The former specifies more on application network traffic and security and the latter on applications running on the monitored infrastructure and the infrastructure itself[34].

Prometheus collects data from applications that push metrics to their API endpoints(or exporters). Nagios uses agents installed on both the network elements and the components it monitors; they collect data using pull methodology.

While also providing various plugins to monitor a diversified range of applications, Nagios focuses much more on monitoring the application network traffic and security, which ends up not suiting the needs of this thesis.

Prometheus is an easier to implement, metrics focused solution. It offers a much wider variety of exporters while Nagios is very limited. Another point against Nagios is that, in terms of visualization of the data collected, it doesn't allow the user to have a custom solution since it comes pre-built into the Nagios system. In contrast, Prometheus allows choosing the best case for the developer's needs, the most common being Grafana.

In the end, Prometheus is a more versatile solution for monitoring with the focus on collecting metrics, which is one of the core features of the system developed, being lightweight on the system and allowing greater modularity by only implementing the plugins needed for a determined solution, easily fitting in most software architectures.

2.5.2 Orchestration Solutions

2.5.2.1 Open Network Automation Platform

ONAP provides a comprehensive platform for real-time, policy-driven orchestration and automation of physical and virtual network functions that will enable software, network, IT and cloud providers and developers to rapidly automate new services and support complete lifecycle management[35].

It also provides a comprehensive design framework to create network services and a run-time framework. The run-time framework includes the ETSI MANO functionality of NFVO and VNFM and goes beyond by adding monitoring and service assurance.

Figure 2.16: ONAP Scope [25]

ONAP primarily interacts with these software systems [25]:

- **Operations support systems (OSS):** OAM, FCAPS (fault, configuration, accounting, performance, security).
- **Business support systems (BSS):** Service provisioning, deployment, service change management, customer information management, SLA management, billing and customer support.
- **Big data applications:** Analytics applications to gain business insights and intelligence from data lakes containing network data collected by ONAP.
- **E-services:** Self-service portals where customers can manage their services.

From 2.16, it is possible to observe that ONAP architecture defines two major systems - design time and run-time - allowing a clear separation between design and operational roles.

The design time environment, which is supported by Service Design & Creation (SDC), is responsible for:

- VNF Onboarding/Validation
- Network Service/SDN Service Design
- Policy Creation
- Workflow Design

The run-time environment is responsible for:

- Service orchestration and lifecycle management:
  - Service Orchestrator
  - SDN Controller (SDN-C)
  - Application Controller (APP-C) and Virtual Function Controller (VF-C)
  - Infrastructure controller (interface to the VIM).
- Monitoring and service assurance
  - Data Collection Analytics and Events (DCAE)
  - Catalogue of all active and available inventory (A&AI)

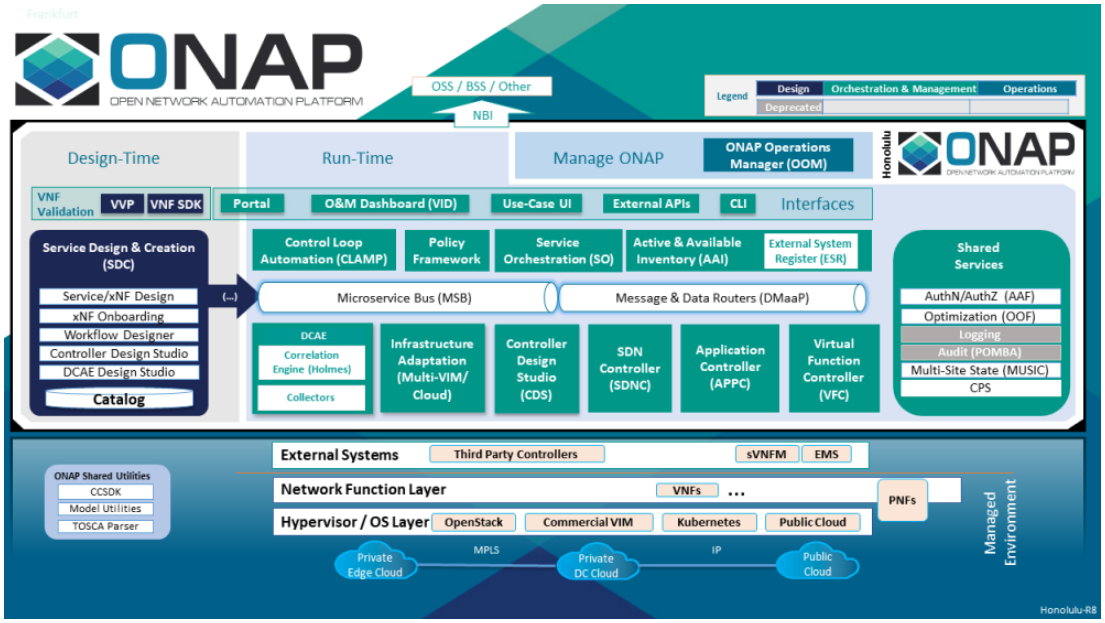


Figure 2.17: ONAP Architecture (Honolulu Release) [36]

The design-time framework brings forth a comprehensive development environment with tools, techniques, and repositories for defining and describing resources, services, and products, including policy design and implementation and an SDK with tools for VNF supplier packaging and validation. On the other hand, the run-time environment executes the rules and policies distributed by the design and creation environment and the controllers that manage physical and virtual networks.

As can be seen on 2.17, ONAP has a significant number of projects that can be instantiated, but for the scope of this thesis, the following are the most relevant[25]:

- **Service Orchestration(SO):** Executes the specified processes by automating sequences of activities, tasks, rules and policies needed for on-demand creation, modification or removal of network, application or infrastructure services and resources, this includes VNFs, Containerized Network Functions (CNFs) and Physical Network Functions (PNFs). Service Orchestration (SO) gets topologies and configurations through resource and recipe models via SDC. The SO is invoked through events or APIs from BSS or manually invoked through Virtual Infrastructure Deployment (VID)[3].SO makes homing (where to place the workload) decisions, resource determination decisions and triggers management actions via one of the four controllers at its disposal. SO also handles errors and rollbacks.

- **Active and Available Inventory (A&AI):** provides real-time views of a system's resources, services, products and their relationships with each other, and also retains a historical view. A&AI not only forms a registry of products, services, and resources, it

[3]The Virtual Infrastructure Deployment (VID) application enables users to instantiate infrastructure services from SDC, along with their associated components, and to execute change management operations such as scaling and software upgrades to existing VNF instances

also maintains up-to-date views of the relationships between these inventory items. A&AI tracks relationships by maintaining a graph database, e.g., subscriber → NS → VNFs → VMs → compute/storage/networking nodes, while allowing discovery, registration, and auditing.

In order to achieve the dynamism of SDN/NFV, A&AI is updated in real time by the controllers as they make changes in the network environment.

- **Policy:** Framework dedicated to comprehensive policy design, deployment, and execution environment and the decision making component in an ONAP system. Run-time policy software module guides the automated system without code - purely through models that allow for dynamic changes.

- **Data Collection, Analytics, and Events (DCAE):** Used for closed control-loop automation, trending, solving chronic problems, capacity planning, service assurance, reporting, and so on. DCAE orchestrates data collectors, microservices, analytic applications, and closed control-loops. Given the model and SDK-driven nature of DCAE, it offers a self-service interface to developers, designers, and operators, interacting with SDC, Policy, and A&AI.

- **Closed Loop Automation Management Platform (CLAMP):** Used to create and configure policies/templates built-in SDC (Service Design and Creation) to create closed control-loop service assurance loops. Control Loop Automation Management Platform (CLAMP), Policy and DCAE actively work in conjunction to detect problems in the network and identify the appropriate remediation, notifying the Service Orchestrator, in order to take action upon the problem. This component requests from DCAE the instantiation of microservices to manage the control loop flow. Furthermore, it creates and updates multiple policies in the Policy Engine that define the closed loop flow. Illustrated in 2.18 is an example of a template being configured.

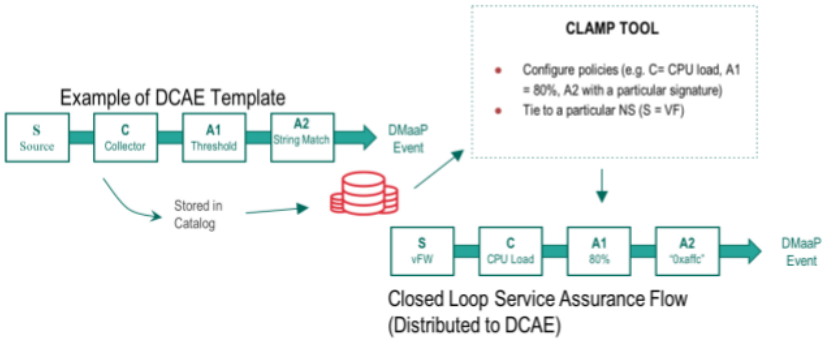


Figure 2.18: CLAMP workflow

2.5.2.2 Other solutions

The chosen solution was ONAP, mainly due to the thesis developed in a collaborative environment between the University of Aveiro and Capgemini Engineering, where the ONAP system was already being used. Nevertheless, other options were considered to justify ONAP as the correct choice.

Another studied solution was Open Source MANO (OSM), the leading competitor of ONAP. In [37], the author also made a comparison between OSM and ONAP regarding NFV Management and Orchestration.

OSM is also an open-source project, which ETSI hosts, and it aims to map the ETSI NFV architecture into its open-source implementation.

Some operations supported by OSM are, similarly to ONAP [37]:

- Orchestration Service for VNFs;
- Support for different VIMs;
- Support for SDN controllers;
- Support for monitoring tools.

OSM also includes a monitoring collector, which collects metrics at the infrastructure level and per VNF and can automate network service orchestration. However, it does not have a module dedicated to monitoring like ONAP provides DCAE. ONAP has every feature existing in OSM and includes a TOSCA and YANG supporting a unified design framework, which helps with end-to-end services orchestration and automation. There are even more solutions that compete against ONAP and OSM. Some are proprietary and lack the advantages of open-source. Others are just less known and, therefore, less used to address these problems.

In table 2.1, it is shown a comparison between ONAP, OSM and other available platforms.

Framework	**Leader**	**VNF Definition**	**SDN**	**NFV**	**Legacy**	**VIM**	**Multiple Domains**
ONAP	Linux Foundation	TOSCA, YANG	✓	✓	✓	✓	✓
OSM	ETSI	YANG	✓	✓		✓	In recent versions
OpenBaton	Fraunhofer Institute Technical U. Berlin	TOSCA		✓		✓	
Cloudify	GigaSpace	TOSCA		✓			

Table 2.1: Comparison between Orchestration Platforms (adapted from [35])

Ultimately, ONAP is a complete solution, as seen in 2.1. Although not every feature is going to be used for the present work, it opens the possibility for future work.

2.6 Related Work

Some work on the End-to-End network Orchestration has already been made and studied. In [38], the authors proposed a 5G network slices management solution.

A network slice is established via negotiation between the customer and the network operator. The customer specifies the needs of connectivity, interconnection, cloud resources, and network functions. The network operator is in charge of implementing those functions and network resources to offer the desired network service according to a given SLA.

For automating the network slices management, the authors were highly motivated by the need to monitor these slices and implement rules that would depend on the monitoring results to maintain and avoid degradation of the negotiated SLA with the consumer. The proposed solution also uses the various components available on the ONAP system, more specifically the DCAE and VNF Event Streaming Collector (VES), used specifically for VNFs monitoring. In case of slice degradation detected through the KPIs sent to DCAE, the Policy Framework, also in ONAP, would then enforce various actions, for instance, allocating more cloud or network resources, terminating tasks, scaling up or down slice components, etc. This actions will assure that the slice could return to its expected performance. The authors also use the CLAMP component for policy enforcement automation to complete the solution.

The authors of [6] and [7], as mentioned in section 1.3, developed a solution to provide the vCDN service an orchestration solution using the ONAP system. This thesis aims to expand upon this service by implementing the necessary components to accomplish a monitoring solution for the system to provide the required scaling, in terms of infrastructure, according to the monitored KPIs. The current work also expands on the ONAP usage. The vCDN only utilises ONAP to deploy the nodes with the desired network configurations to properly serve the clients.

Although the service itself is already integrated with policy-driven orchestration, it can only adjust its content distribution strategy and still needs infrastructure scaling for a real-world scenario.

ONAP offers the capability of not only automating the deployment of network services but also assuring that they maintain the quality of the service throughout their lifecycle, requiring the use of more components in the ONAP system, which is the main focus for the mentioned expansion of the ONAP system, allowing for a scaling solution to the vCDN. The ONAP functionalities extension and the monitoring solution are the main research gap between this work and the studies in [6] and [7].

The work developed in [38], in the monitoring aspect, proves that the solution envisioned and later detailed in Chapter 5, will enable monitoring and infrastructure scaling in [6].

CHAPTER 3

Research Methodology

The following chapter presents the research methodology applied to develop this thesis. The main focus is to understand how the research influenced the choices made in the framework design and how the data should be interpreted both for the monitoring solution of the vCDN system and the closed-loop automation with the Open Network Automation Platform.

As previously stated in 1.3, the main goal is to provide a monitoring system for the vCDN Service and successfully achieve closed-loop automation with the aid of the ONAP platform, focusing on validating the control loop to be used as an automation tool to manage the service assurance of the vCDN. The main objective of the research is to understand if the implementation of an automation control loop is viable and achievable in a real-world scenario, thus, solving the scalability problem of the vCDN that may happen during the overload of the service nodes applying the concept of applied research.

To successfully evaluate the system, it is required to gather and analyze precise values from the monitoring system that was instantiated in the vCDN Node intended to monitor. The data will then be used to feed the ONAP platform, more precisely, the collector belonging to the DCAE component, to be further processed by the platform itself and execute the defined control loop.

Having this in mind, the implemented solution needs to provide quantitative values that are agnostic to the observer's view, and it is only dependent on the vCDN Node. Also, all the metrics used to produce the statistical results for posterior evaluation were collected for this sole purpose, and no other data source was used.

Figure 3.1 illustrates the various steps that represents the research methodology.

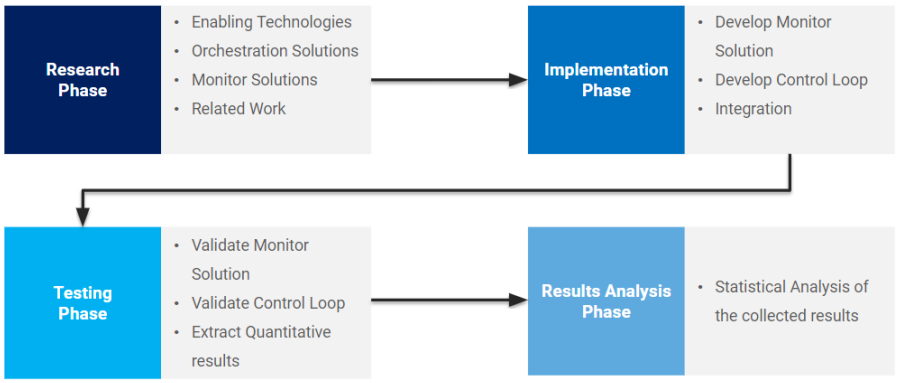


Figure 3.1: Research Methodology

There were essentially four steps for the methodology execution:

- **Research Phase:** It consisted in the study of the enabling technologies such as 5G, NFV, SDN and MEC, as well as the orchestration and monitoring solutions available, and also a compilation of the related work, which were addressed in Chapter 2.
- **Implementation Phase:** Development of the monitoring solution with the Prometheus Framework and the Control Loop with the ONAP system, and this phase ended with the integration of both solutions to be able to execute the required use case properly.
- **Testing Phase:** Phase dedicated to testing the implemented solutions. The main objective was to validate the systems built during the Implementation Phase. After successfully testing the solutions, the required quantitative data was extracted for analysis.
- **Results Analysis Phase:** Final phase and logical consequence of the previous stage, in which the goal was to analyze the previously extracted data from the Testing Phase.

The data used is entirely quantitative, as already briefly said previously. The data collection consisted in using the Prometheus Framework to collect the different hardware metrics from the nodes. Those metrics were gathered from the CPU, RAM, Disk I/O, and Network Interfaces every 10 seconds, and they are a calculation of the mean values of the last 10 seconds to prevent detection of false positives that could happen with load spikes in the nodes.

In a usual scenario, Prometheus would feed the metrics directly to the ONAP collector. Still, the messages that needed to be sent to the platform were manipulated to cause the desired event for testing purposes.

Although monitoring is essential for evaluating if the solution is manageable in a real-world scenario, manipulating the value allows a more controlled experimental environment. Also, it will enable testing the condition where the control loop is triggered by an event caused by an overload on the system.

The connectivity to the collector was tested separately. In this case, a script was in charge of collecting the values from Prometheus by querying his HTTP API and converting the information to the VES Collector format.

The script converts information and establishes the link between Prometheus and the DCAE collector. If this connection was achieved, the connectivity between these components was validated and assumed to work on the real-world scenario.

The next step to validate the solution was to guarantee that the control loop was functional and scale the service when the right conditions were met. The conditions defined for a control loop to be triggered were: CPU or RAM values above 80%.

It is worth mentioning that various delays existed from Capgemini Engineering in supplying the materials that would serve as the infrastructure for the ONAP deployment. Consequently, a delay also occurred in the validation and testing of the solution, causing the work to be postponed.

Due to this slow down on the development, validation and, testing, other planned work was not possible to execute, for example, stress test the vCDN nodes to precisely discover the maximum bandwidth for the network interfaces and the bottleneck for disk I/O operations. Although those values are being extracted from the Prometheus Framework, they are not used in the control loop policy. A message was forged with these values exceeding the established thresholds to trigger these referred conditions, which the VES Collector will receive.

Regarding the vCDN service, not all the desired data was collected due to the service itself not having a load balancer. However, the missing results were not detrimental to the final conclusion or the resolution of the research question. To evaluate this stage, the collected data was based only on instantiation time, and the only data used was when a new node deployment was successful. This was collected by a script that filtered the results so that only valid instantiation times were used for the final calculations.

The instantiation times for deploying vCDN nodes were the most relevant data to collect for the objective established. The number of collected tests was significant to try to make the posterior calculations of the statistical values less error-prone. A total of a hundred and eighteen tests were successfully completed, and those were the times used to calculate the mean value and standard deviation. There is no reason for the high number of tests other than the need to attenuate the results of the calculations.

The approach was beneficial because all the collected metrics during the experiment were quantitative, answering whether the instantiation times were acceptable to provide successful service assurance or if the monitoring system and the control loop required further refinement.

The results, although they prove that the closed-loop automation is possible and within adequate time to replace human interaction in the process, lack the evaluation of the vCDN node in terms of the system load after the instantiation of a new node. Still, the main question was answered, making the outcome outweigh this limitation that was faced during the development of the system.

CHAPTER 4

ONAP and vCDN Components Assessment for Scenarios Implementation

The present chapter introduces the proposed scenario for the implemented solution. The goal is to explain the components involved in constructing the solution with ONAP, to address the needs of the envisioned use case. The chapter also presents the technologies and frameworks used to achieve the final result.

4.1 Proposed Scenario

The 5G networks aim to improve the way customers use mobile networks, and because of the demanding use cases, systems that operate on the network must also evolve to meet the requirements. With the increased accessibility and high speeds associated with 5G, the data consumption, more specifically, video streaming, will be at an all-time high, as previously stated in Chapter 1, section 1.1.

Having this in mind, the services delivered by the network providers must have mechanisms that allow dealing with the traffic increase to meet the demands of today's mobile networks. These mechanisms will be essential in scenarios where a significant accumulation of people occurs, like metropolitan areas where people consume the most of this type of data.

A very likely location where this might happen is at public transportation, such as bus stations or train stations, at the end of the working hours, where people will be on their way home and start to consume multimedia content to be kept entertained. The amount of traffic generated at these locations can be overwhelming for systems that were not designed to handle the increase in volume.

Such is the case of the vCDN service, that, although it supports the placement of new nodes based on the predicted location of the users, along a train track, it can't scale itself in

the proportion of the resources overload caused on the system components, such as CPU or RAM.

The focus is to, despite the demanding increases in computational resources due to the higher traffic generation, the service maintains the same quality when delivering the content to the consumers and avoid its disruption or degradation.

Figure 4.1 represents a high level implementation illustration of this scenario.

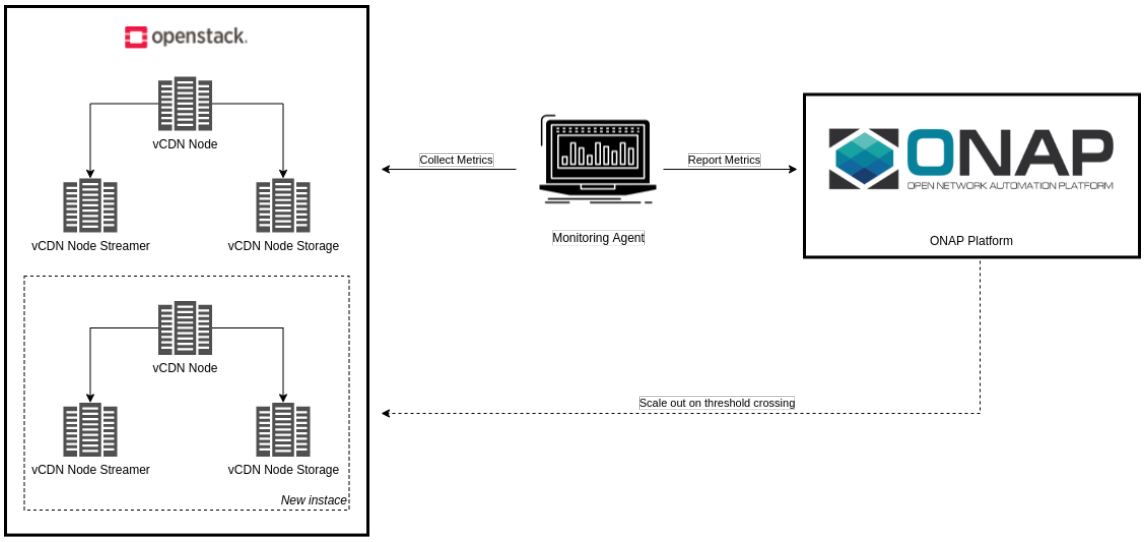


Figure 4.1: High Level Architecture for vCDN Scaling

The designed system aims to address the issue that will come with this growth in network flow by leveraging the capabilities of ONAP and providing network automation through the means of control loops.

The control loop is a mechanism that will be triggered at a certain point in time and will provide the necessary signals and actions inside the ONAP framework to execute the adjustments in the service successfully.

The deployed framework will focus on the constant monitoring of the vCDN service and analyze the collected data. The data will be forwarded to the ONAP infrastructure. The platform will treat it and trigger the necessary mechanisms to enable the service scaling, thus providing the resources for quality assurance.

4.2 System Requirements

To successfully implement the described in section 4.1, several components must work in conjunction.

The first required component will be the monitoring system. As stated earlier in this document, the advance on the development was heavily dependant on the monitoring implementation.

As it was studied in Chapter 2, the monitor solution uses the Prometheus Framework. Prometheus represents a significant role in the architecture since it is the single source of VNFs measurements. Since the VNFs of the vCDN are Virtual Machine Instances running on

the OpenStack platform, it is only needed to run the Prometheus exporters on these VMs, and each of them also runs a server to scrape the metrics. The VMs are instantiated by the ONAP and based on specific TOSCA templates already pre-defined. They have support for Python 3.8, which is a requirement to run the script that will collect the metrics from the Prometheus servers send them over to the ONAP collector.

In terms of the ONAP itself, since it is a very modular platform, not all the components are required to be running to execute the desired solution. The necessary parts are the following:

- Service Design and Creation (SDC)
- Service Orchestrator (SO)
- Active and Available Inventory (A&AI)
- Application Controller (APPC)
- Data Movement as a Platform (DMaaP)
- Data Collection Analytics and Events (DCAE)
- Policy Framework

These are the main components required to deploy the control loop associated with the vCDN service and meet the requirements of the defined use-cases.

There are other components that ONAP uses to properly function, for instance, the Application Authorisation Framework (AAF), Multi VIM/Cloud, Virtual Function Controller(VF-C), and SDN Controller (SDN-C). Still, since these are secondary for understanding the implementation of the use case, they will not be addressed in further detail.

4.2.1 Service Design & Creation

SDC [25] is the unified tool, for the ONAP system, for design-time activities such as onboarding VNFs, creating services, creating policies, creating workflows, onboarding data collectors, onboarding analytic apps and testing them, approving and distributing artifacts to run time. This component manages the content catalog and logical assemblies of the catalog items to define how and when VNFs are realized in a target environment.

The SDC manages two levels of assets [39]:

- **Resource:** Fundamental capability implemented either in software alone or as software interacting with a hardware device. Each Resource combines one or more Virtual Function Components (VFCs), along with all the information necessary to instantiate, update, delete, and manage the Resource. A Resource also includes license-related information. There are three kinds of Resource:
  - Infrastructure (the Cloud resources, e.g., Compute, Storage)
  - Network (network connectivity functions & elements); example: a Virtual Network Function (VNF)
  - Application (features and capabilities of a software application); example: a load-balancing function

- **Service:** A well formed object comprising one or more Resources. Service Designers create Services from Resources, and include all of the information about the Service needed to instantiate, update, delete, and manage the Service

There are four major components of SDC which are:

- **Catalog:** The repository of assets at the Resource and Service levels.
- **Design Studio:** Used to create, modify and add Resource and Service definitions on the Catalog.
- **Certification Studio:** Used to test new assets at all levels.
- **Distribution Studio:** Used to deploy certified assets. From the Distribution Studio, new Services, including their underlying Resources, are deployed to lab environments for testing purposes and into production after certification is complete..

Once assets are on-boarded, the information provided by the vendor is translated into SDC internal resource models. The service provider will use SDC to enrich the resource model further to meet the provider's environment and compose resources into service models. The model includes the description of the asset and references to SDC functions needed for the lifecycle management of the asset. The tested models will then be distributed to the ONAP execution environment as Deployment Artifacts.

The Deployment Artifacts include the asset definition with instructions to ONAP for the creation and management of an instance of the asset in the network. Currently, SDC imports and retains information from Heat Templates for cloud infrastructure creation, YANG XML files for state data manipulated by the Network Configuration Protocol, TOSCA files for specifying cloud infrastructure, and specific vendor provided scripts[39].

4.2.2 Service Orchestrator

The Master Service Orchestrator (MSO) [40], commonly known as SO manages orchestration at the top level and facilitates additional orchestration that takes place within underlying controllers. It also marshals data between the various controllers so that the process steps and components required for the execution of a task or service are available when needed. The MSO's primary function is to automate end-to-end service instance provisioning activities. MSO is responsible for the instantiation and release and subsequent migration and relocation of VNFs to support overall end-to-end service instantiation, operations, and management. The procedure which the MSO needs to run is obtained from the Service Design and Creation (SDC) component of ONAP. Controllers (Network and Application) participate in service instantiation and are the primary layers in ongoing service management; for example, control loop actions, service migration and scaling, service configuration, and service management activities.

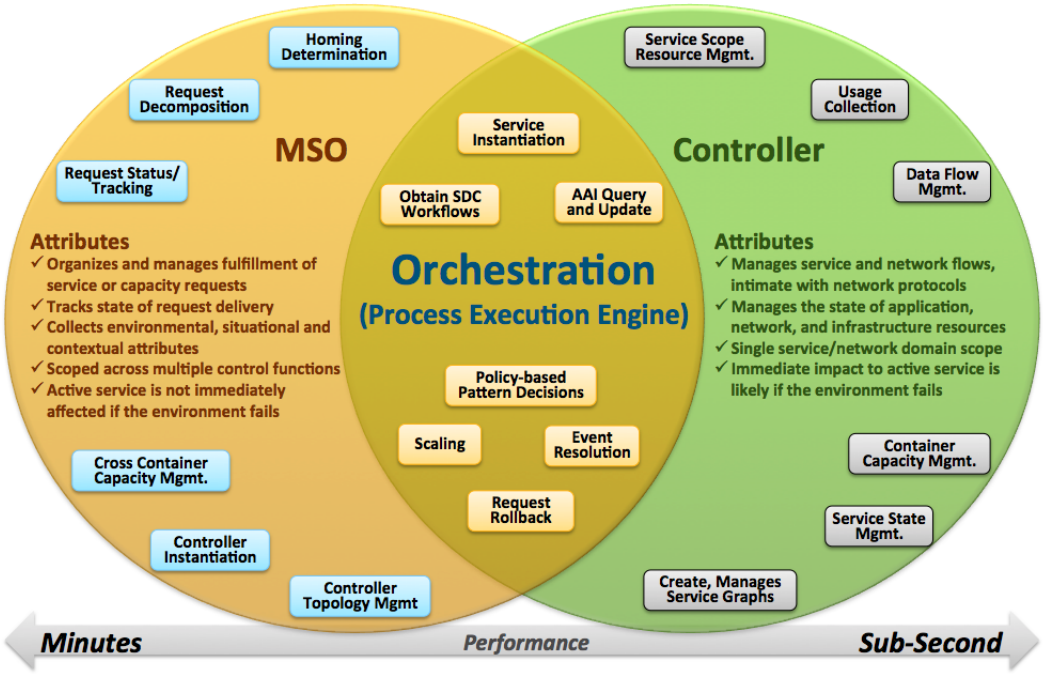


Figure 4.2: Orchestration

Figure 4.2 represents the two major domains that use the orchestration: Service Orchestration and Service Control, embodied by the MSO and the Application/Network Controllers, respectively. Each major domain, will perform orchestration for:

- Service delivery or changes to an existing service
- Service scaling, optimization, or migration
- Controller instantiation
- Capacity management

ONAP interfaces with the cloud provider's infrastructure control interface for infrastructure orchestration. Regardless of the focus of the orchestration, all workflows must include steps to update A&AI with configuration information, identifiers, and IP Addresses.

Regarding the Application and Network Controller orchestration, for the former, MSO sends requests to Application Controllers to obtain the application-specific component of the service procedure from SDC and execute the orchestration workflow. MSO ensures that the Application Controller completes its resource configuration as defined by the procedure. For the latter, MSO obtains compatible Network Controller information from A&AI and requests LAN or WAN connectivity to be established and configured. This may be done by asking the Network Controller to receive its resource procedure from SDC. It is the responsibility of MSO to request (virtual) network connectivity between the components and ensure that the selected Network Controller completes the network configuration workflow[40].

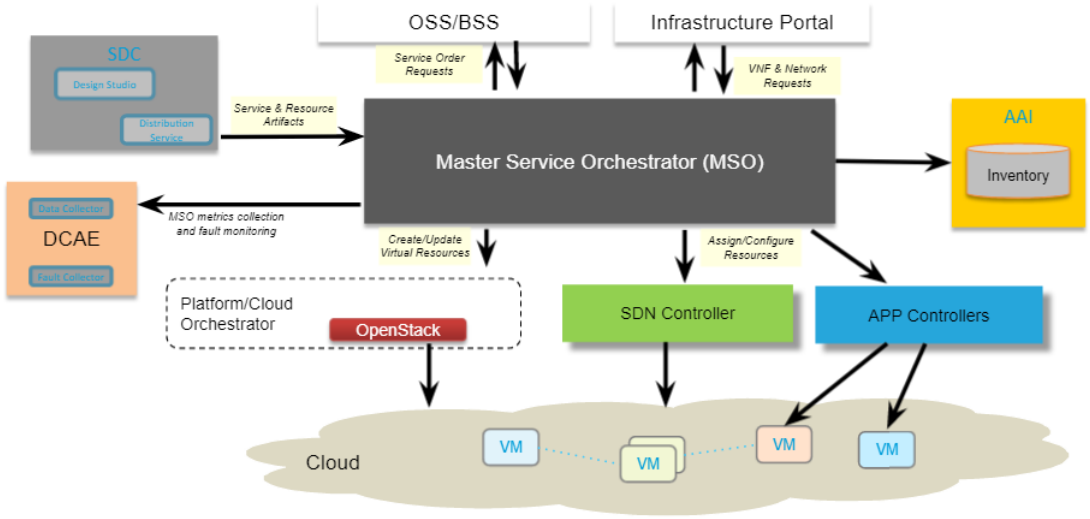


Figure 4.3: Service Orchestrator High Level Architecture

4.2.3 Active and Available Inventory (A&AI)

It provides real-time views of a system's resources, services, products, and relationships with each other and retains a historical standpoint. A&AI not only forms a registry of products, services, and resources. It also maintains up-to-date views of the relationships between these inventory items. To achieve the dynamism of SDN/NFV, A&AI is updated in real-time by the controllers as they make changes in the network environment[41].

The functional diagram of this component is show in Figure 4.4.

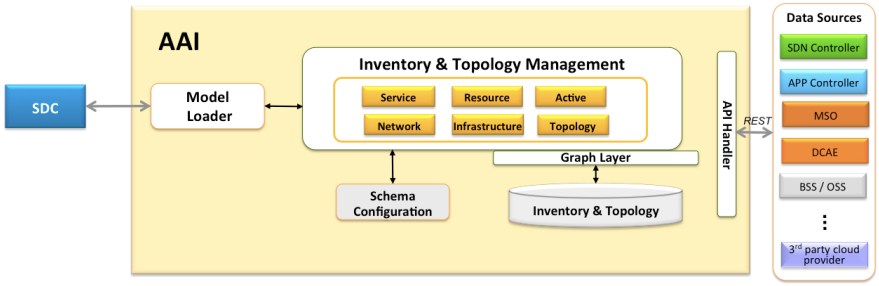


Figure 4.4: Active and Available Inventory (A&AI) functional diagram

To execute inventory and topology management, A&AI uses a central registry to create a global view of inventory and network topology. A&AI receives updates from various inventory masters distributed throughout the ONAP infrastructure and persists enough to maintain the global outlook. As transactions occur, A&AI persists asset attributes and relationships into the federated view based on configurable metadata definitions for each activity that determine what is relevant to the A&AI inventory. A&AI provides standard APIs to enable queries from various clients regarding inventory and topology. The A&AI global view of relationships is

necessary for forming aggregate views of detailed inventory across the distributed master data sources.

In addition to inventory and topology management, A&AI provides the ability to do inventory administration, meaning it doesn't require a system shutdown to perform functions on the metadata models stored, such as updating and dynamically version them, as requested by the system. Data in A&AI is continually updated in real-time as changes are made within the cloud. Because A&AI is metadata-driven, new resources and services can be added quickly with SDC catalog definitions, using the A&AI model loader, thus eliminating the need for lengthy development cycles. In addition, new inventory item types can be added quickly through schema configuration files.

The A&AI subsystem uses graph data technology to store relationships between inventory items. Graph traversals can then be used to identify chains of dependencies between items.

A&AI data views can be used by homing logic during real-time service delivery, root cause analysis of problems, impact analysis, et cetera[42].

4.2.4 Application Controller (APPC)

Application controllers, such as APPC, receive orchestrated requests from the MSO, obtaining application-specific components and attributes from SDC. The MSO continues to be responsible for ensuring that the Application Controller completes its Resource configuration as defined by the workflow[43].

The APPC is one of the components of the Open Enhanced Control, Orchestration, Management, and Policy platform, and is responsible for handling the Life Cycle Management (LCM) of VNFs.

ONAP includes a generic Application Controller that receives commands from ONAP components, such as MSO, DCAE, or the Portal, and uses these commands to manage the life cycle of Services, Resources (virtual applications and Virtual Network Functions), and their components[44].

4.2.5 Data Movement as a Platform (DMAAP)

DMaaP is a premier platform for high performing and cost effective data movement services that transports and processes data from any source to any target with the format, quality, security, and concurrency required to serve the business and customer needs.

The component consists of three major functional areas (illustrated in 4.5) [45]:

- **Data Filtering:** Data preprocessing at the edge via data analytics and compression to reduce the data size needed to be processed.
- **Data Transport:** Transport of data intra and inter data centers. The transport will support both file based and message based data movement. The Data Transport process needs to provide the ability to move data from any system to any system with minimal latency, guaranteed delivery and highly available solution that supports a self-subscription model that lowers initial cost and improves time to market.

- **Data Processing:** Low latency and high throughput data transformation, aggregation, and analysis. The processing will be elastically scalable and fault-tolerant across data centers. The Data processing needs to provide the ability to process both batch and near real-time data.

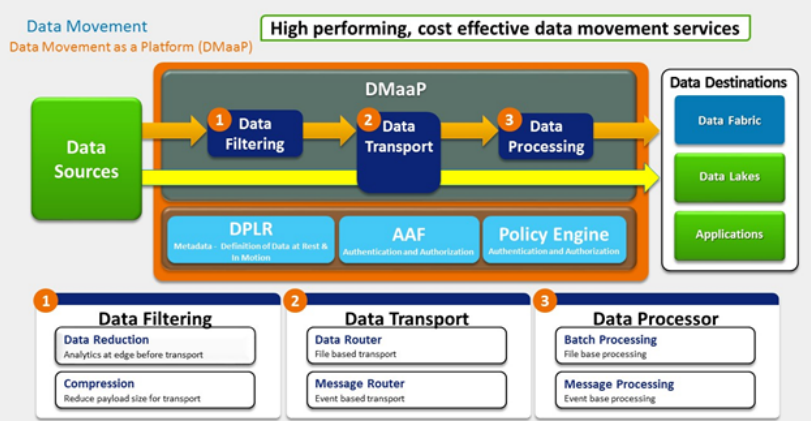


Figure 4.5: DMaaP

4.2.6 Policy Framework

The Policy Framework is the decision making component in an ONAP system. It allows to specify, deploy, and execute the governance of the features and functions in your ONAP system, be they closed loop, orchestration, or more traditional open loop use case implementations.

The ONAP Policy Framework architecture separates policies from the platform supporting them. The framework supports the development, deployment, and execution of any policy in ONAP. The Policy Framework is metadata (model) driven to make policy development, deployment, and execution as flexible as possible and can support rapid development. It also complies with the TOSCA modeling approach for policies.

This framework has five essential capabilities [46]:

- Capable of being triggered by an event or invoked, and making decisions at run time.
- Capable of managing policies for various Policy Decision Points (PDP) or policy engines.
- Metadata driven, allowing policies to be deployed, modified, upgraded, and removed as the system executes.
- Flexible model driven policy design approach for policy type programming and specification of policies.
- Extensible, allowing straightforward integration of new PDP, policy formats, and policy development environments.

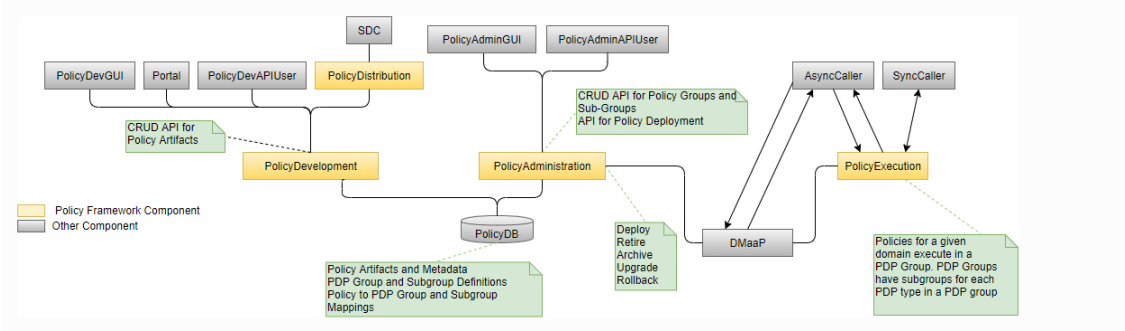


Figure 4.6: Policy Framework Architecture

The Policy Development component implements the functionality for development of policy types and policies.

The Policy Development component implements the functionality for developing policy types and policies. Policy Administration is responsible for the deployment life cycle of policies and interworking with the mechanisms required to orchestrate the nodes and containers on which policies run. Policy Administration is also responsible for the administration of policies at run time, ensuring that policies are available to users, executing correctly, and monitoring the state and status of policies.

Policy Execution is the set of PDP running in the ONAP system. It is responsible for making policy decisions and for managing the administrative state of the PDP as directed by Policy Administration[46].

4.2.7 DCAE

DCAE [47] is the data collection and analysis subsystem of ONAP. Its tasks include collecting measurement, fault, status, configuration, and other types of data from network entities and infrastructure that ONAP interacts with, applying analytics on collected data, and generating intelligence for other ONAP components such as Policy, APPC, and SDNC to operate upon, completing the ONAP's close control loop for managing network services and applications. DCAE Platform supports the functions to deploy, host, and perform Life Cycle Management applications of Service components. It uses the TOSCA model of the control loop, specified by a triggering call, and then interacts with the underlying networking and computing infrastructure to deploy and configure the virtual apparatus needed to form the control loop. DCAE Service components are the functional entities that realize the collection and analytics needs of ONAP control loops.

4.2.7.1 VNF Event Streaming Collector

VES Collector is one of the various services ONAP DCAE offers, and it is the service for metics collection that will be used on the scope of this project.

VES Collector is a RESTful collector for processing JSON messages into DCAE. The collector supports individual events or event batches posted to collector end-point(s) and publishes them to interface/bus for other applications to subscribe. The collector verifies the source and validates the events against the VES schema before distributing to DMaaP Message

Router (MR) topics for downstream systems to subscribe. The VES Collector also supports configurable event transformation functions and distribution to DMaaP MR topics[48].

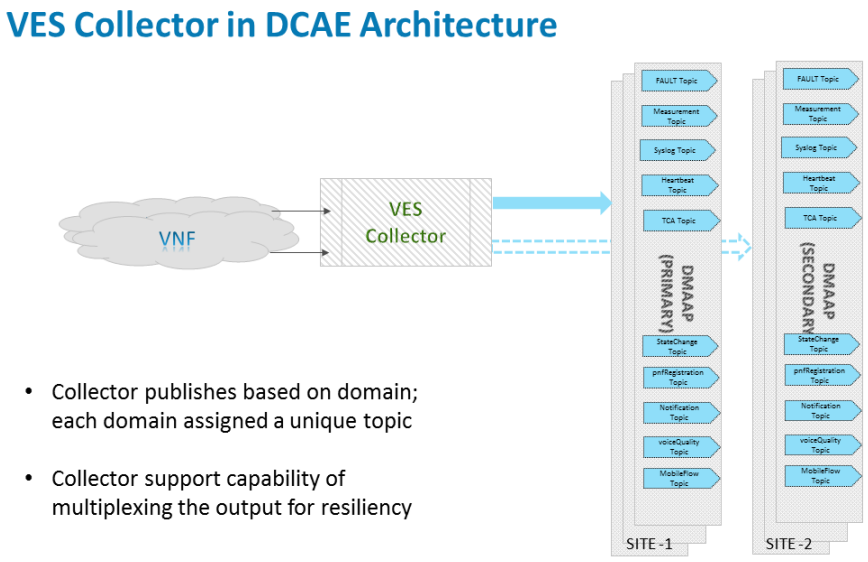


Figure 4.7: VES Collector in DCAE Architecture

4.2.7.2 Closed Loop Automation Management Platform (CLAMP)

CLAMP is a platform for designing and managing control loops. It is used to visualize a control loop, configure it with specific parameters for a particular network service, then deploy or undeploy it[49]. The policies that CLAMP will configure are created on SDC to produce the closed-loop service assurance. Once deployed, the user can also update the loop with new parameters during runtime and suspend and restart it. It interacts with other systems to deploy and execute the control loop.

The closed loop paradigm can be seen in Figure 4.8 [50]:

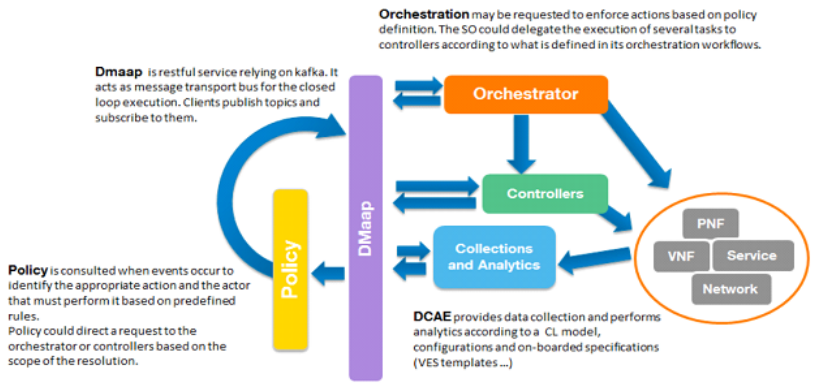


Figure 4.8: Closed Loop Paradigm

Design and execution is illustrated in Figure 4.9.

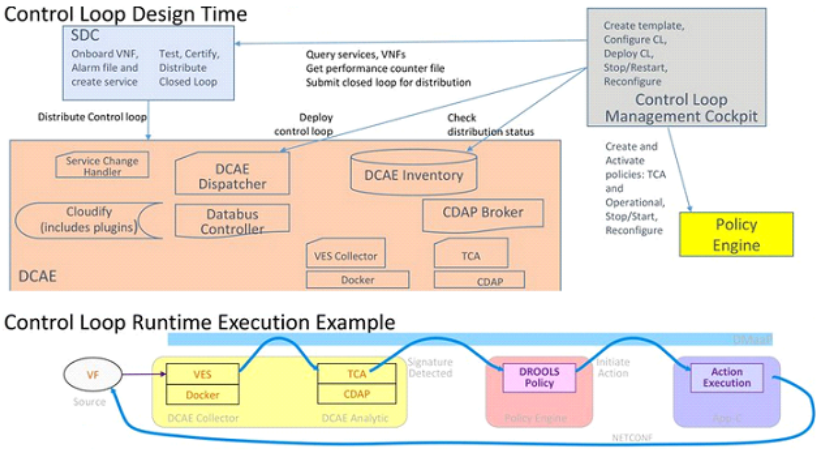


Figure 4.9: Closed Loop Execution

4.3 vCDN Service

The vCDN service is where the solution was implemented and tested. The service itself considers that users will connect to the Video on Demand service the operator provides via a 5G network in a high-speed mobility scenario.

The vCDN system is composed of the vCDN Nodes and the vCDN Engine. The vCDN Nodes are the data plane components of the vCDN system, which are placed in the MEC and thus closer to the end-user. They hold video content and deliver it to end-users. These nodes, which can be considered distributed caches, are connected to a central controller (vCDN Engine), which controls the cache contents of each node, having an overall view of the vCDN topology. The vCDN Engine can communicate with the infrastructure's MANO to deploy non-existent nodes in available MEC hosts along the path of movement of users[51].

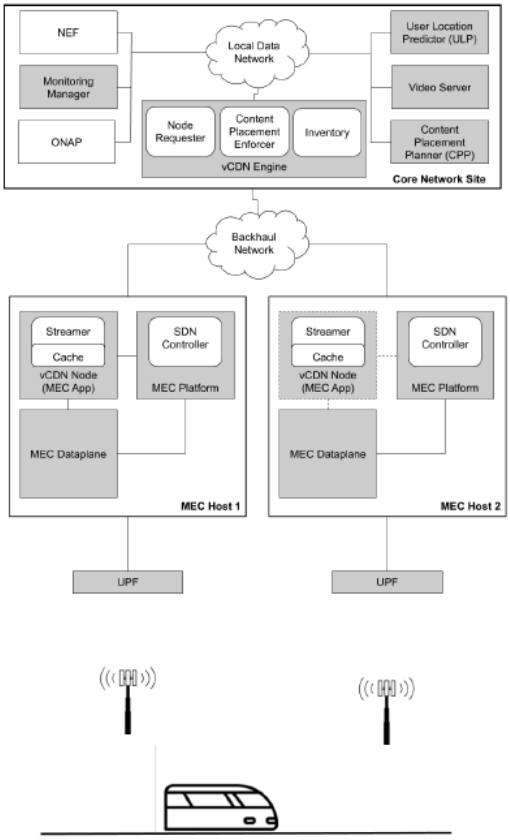


Figure 4.10: vCDN High-Level architecture

Although the vCDN service is reasonably complex and has many components that allow for a complete functioning system, the purpose of this thesis will not focus on all of these components. For the scope of the present document, only the vCDN Nodes were considered, more specifically the vCDN Node Streamer and the vCDN Node Storage, which are detailed further ahead.

4.3.1 vCDN Node

The vCDN node is a key element within a CDN infrastructure and comprises the distributed elements that enable the use of edge DCs to optimize the transmission of VR/AR content to UE. Herein, it is composed of the following elements [51]:

- **vCDN Node Streamer:** Component that establishes an HTTP session with video or VR/AR applications in the UE to deliver the video or VR/AR content to be displayed.
- **vCDN Node Storage:** Component that enables the local persistence of VR/AR content to be consumed by the streamer application.

4.3.1.1 vCDN Node Streamer

Component that establishes an HTTP session with video or VR/AR applications in the UE to deliver the video or VR/AR content to be displayed. Its main functions are:

- Handle client requests. It intercepts the request if it is a request for a video chunk triggering a cache search. If there is a cache miss, it forwards the request to the video origin in order to obtain the missing chunk.
- It has an internal cache, in RAM, to load the next chunk that will probably be requested. The chunk for this load can only come from the vCDN Node Storage.
- Request video chunks to the video origin and send them to the vCDN Node Storage.
- Send monitoring data regarding connected clients, video information, cache status and viewing status to the Monitoring Manager (MM).
- Load video chunks to the storage upon request from an external entity (using the APIs)
- Influence the traffic in the MEC dataplane via the Mp1 interface.
- Load video chunks in bulk, in response to an API request, from either the video origin or a peer cache.

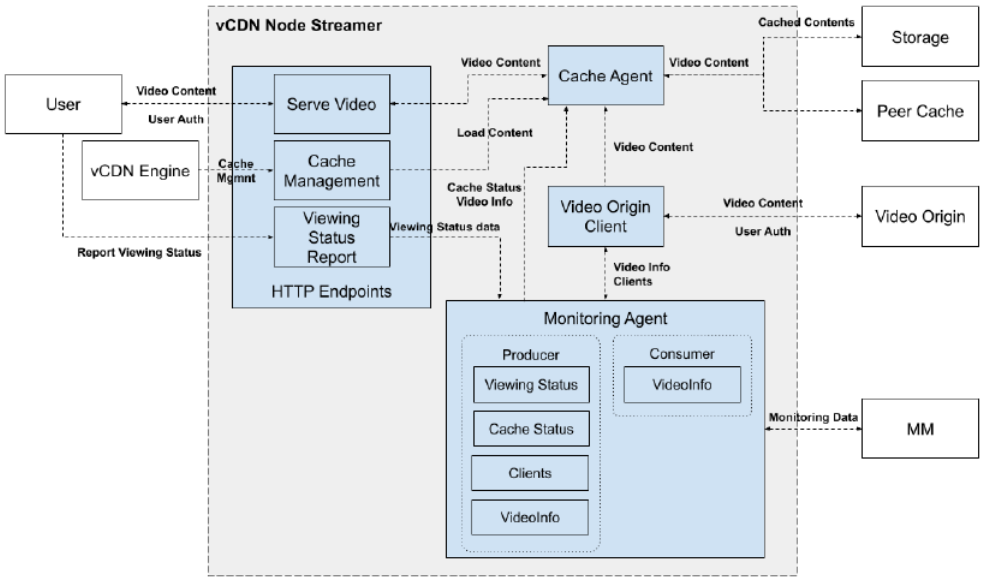


Figure 4.11: vCDN Node Streamer internal architecture

The vCDN Node Streamer, whose internal architecture is presented in figure 4.11, is composed by four essential modules: HTTP Endpoints that implement methods to deliver data or to be controlled by other entities, Cache agent that treats all the cache related functions, Video Origin Client that provides communication with the video origin and Monitoring Agent, which can send and receive monitoring data to and from the MM.

4.3.1.2 vCDN Node Storage

Component that enables the local persistence of VR/AR content to be consumed by the streamer application. Its features include [51]:

- The ability to receive video chunks from the streamer and store them in a key-value manner.

- Provides chunks to the streamer also using a key-value manner.

Since the vCDN Node uses a remote storage to cache content, the response time of the CDN node will increase. To counter this situation the CDN node uses an internal RAM cache where the next chunk of the video that the user is likely to consume is stored before-hand.

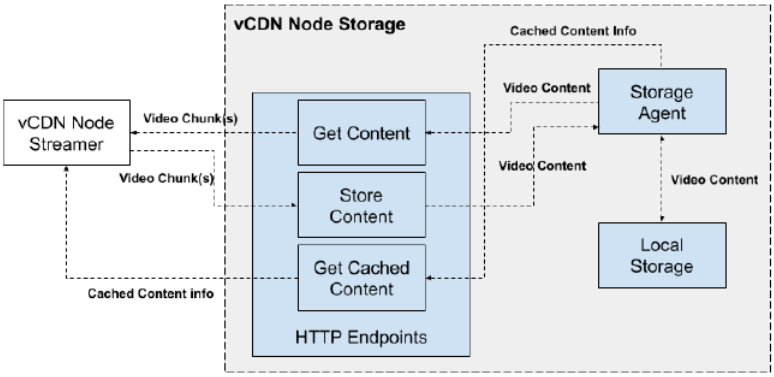


Figure 4.12: vCDN Node Storage internal architecture

The vCDN Node Storage, with its internal architecture presented in figure 4.12, is the entity capable of persistently holding content. Similarly to the Streamer, this entity also implements HTTP endpoints that enable it to communicate with the streamer. It also implements a storage agent that handles content storage.

CHAPTER 5

Implementation

This chapter explores the implementation of the scenario presented in section 4.1. Throughout the chapter, it is shown the envisioned architecture for the scenario and an explanation about the configurations used for some of the components that were introduced in section 4.2. The implementation of the system also reflects the final system used to test and gather results that are addressed in Chapter 6.

Taking into account the study made in section 4.2, for each of the required ONAP components, to understand at a deeper level, it was possible to conceive the lower-level system architectures for the system to successfully deploy the closed-loop for a fully automated scale-out operation.

The following sections will explain the architectural implementation of the solution.

5.1 Monitoring

The monitoring system for the vCDN service was developed to scan only the nodes mentioned in section 4.3 - vCDN Node Streamer and vCDN Node Storage - since these nodes are the vital ones to maintain a service assurance to the customers connected to the service.

Having this in mind, these nodes are deployed in the form of Virtual Machines on the OpenStack[1] Platform. The monitoring solution focuses on collecting the hardware measures from each of the virtual machines, including CPU, RAM, Network Interfaces traffic, and Disk I/O.

The solution is focused on using the Prometheus framework and an available official exporter that works in conjunction with it - Node Exporter - to implement the collection of metrics from the virtual machines.

The Node Exporter actively collects metrics on the system it is deployed in, and the Prometheus Server pulls those metrics at a timed interval between each pull, thus, providing all the necessary metrics to be later analyzed inside the ONAP system.

[1] `https://www.openstack.org/`

On the virtual machines, a script (converter) was running, on a local machine, to collect the metrics from the server and send them over to ONAP, more specifically the VES Collector, which belongs to the DCAE component. To avoid reading possible load spikes in each node, the sent metrics are already a calculated average of the last 10 seconds of readings, minimizing the risk of these events causing a control loop trigger.

This script has the responsibility of not only querying and gathering the collected metrics but also translating them to the appropriate format that is supported by the VES Collector. There are various formats that the collector accepts, but for the implementation, the Common Event Format v28.3 (CEF v28.3) [2] was used, defined in the VES 5.4 Specification [3], because it is the only acceptable format by the TCA Microservice.

By the end of this conversion, the script will then send the formatted message via HTTP Request to the exposed API of the VES Collector.

The VES Collector will receive and process the information to further be processed by the TCA microservice instantiated inside of DCAE. TCA will be addressed in section 5.2.

In figure 5.1 it is illustrated the architectural representation of the monitoring section of the system.

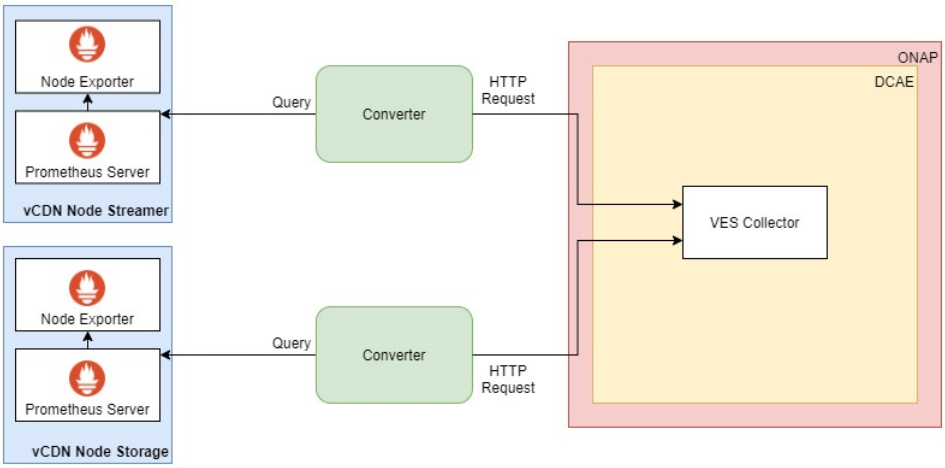


Figure 5.1: Monitoring System Architecture

For the designed architecture, the converter was deployed on a local computer with access to the Capgemini Engineering network using a VPN, due to the fact that the ONAP framework was deployed inside the company's private network, but it is also possible to deploy them directly inside the vCDN nodes and make the proper adjustments to be able to route the HTTP Requests to the VES Collector API.

The full code and for the python script and the file for Prometheus configuration are available in Appendix A.

[2] `https://github.com/onap/dcaegen2-collectors-ves/blob/master/dpo/data-formats/VES-5.28.3-dataformat.json`

[3] `https://docs.onap.org/projects/onap-vnfrqts-requirements/en/latest/Chapter8/ves_5_4_1/VESEventListener.html`

5.2 Data Analysis and Decision

To analyze the data sent by the Prometheus Server into the VES Collector, ONAP provides a microservice, ready to be instantiated for these purposes, which is the already referred TCA Microservice.

After VES Collector receives a message that is compliant with the defined specification, as previously stated, it is in charge of constructing the message to TCA. It is now that DMaaP takes action and receives the message in a specific topic utilized by the collector, and TCA listens for incoming messages on that same topic.

TCA will periodically probe the topic for new messages. Once it reads a message, it will begin comparing the values delivered in the message with the thresholds that were previously defined in its policies. If any of the thresholds were crossed, an ONSET or ABATED message is generated.

The purpose of this generated message is to notify the Policy Framework that a specific limit has been crossed, and it is now time to activate the policy related to that crossing if one is defined.

These two components communicate the same way as the collector and TCA communicate between them. TCA will publish the message to a specific DMaaP topic, and Policy Framework constantly monitors that channel for possible incoming messages.

After the Policy Framework is requested to take action, it will call the actors and, in this project's case, trigger the steps required to execute a successful scale-out operation without the need for human intervention.

The configurations for the relevant components will be detailed in the next section.

In figure 5.2 is illustrated the high-level architecture and interaction between monitoring system and the ONAP, since the reception of metrics to the trigger of the control loop from the Policy Framework and afterward calling the necessary actors for the defined use case.

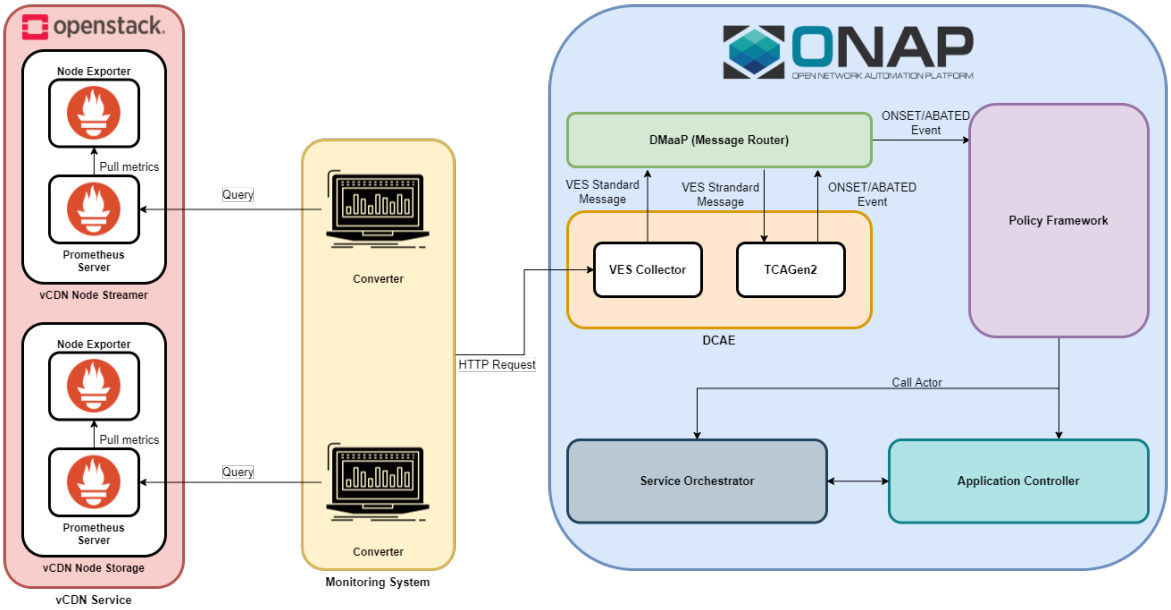


Figure 5.2: Monitoring, Analytics and Decision System Architecture

5.3 ONAP Components

The current section explains the configurations made in some of the ONAP components to deploy the control loop for the presented used case.

The main components that need specific configurations are the Service Design & Creation Configuration, the Policy Framework that will be configured through CLAMP to distribute the policies for the TCA microservice, which requires the Policy XACML PDP Engine, and the operational policy that requires a different PDP - Policy Drools PDP - both using TOSCA modeling language to define the policies, as already stated in section 4.2. The following sections will explain how these components work together to deploy the control loop.

5.3.1 Service Design & Creation Configuration

The Service Design & Creation is the design-time component of the ONAP Framework. To instantiate a control loop, the first step is to design a service, which will later be used by CLAMP.

To design it, Virtual Functions must be instantiated to assign it to a determined service. In this case, the used VFs are relative to the previously developed system, outside of this thesis scope, the vCDN service nodes.

To instantiate these nodes, a HEAT[4] template was provided by the Capgemini Engineering team, and they also handled the VF creation process.

After creating the VF, the next step is to create a service that incorporates the desired VF. The creation of this service will enable the direct attachment of the artifacts needed for the control loop to operate correctly on that service. For this project, only one artifact must be attached, which is the Threshold Crossing Analytics microservice.

[4] `https://docs.openstack.org/heat/rocky/template_guide/hot_spec.html`

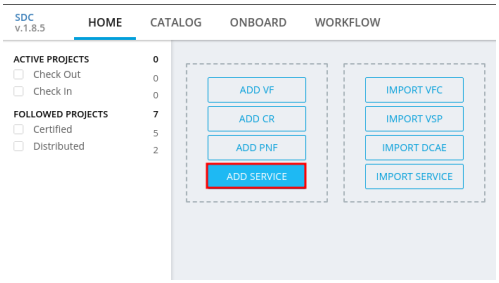
The blueprint for this DCAE microservice can be found in the ONAP public repository[5], although this version of TCA, 1.2.1, had a bug that was reported[6] to the ONAP team, during the development of this thesis.

The bug was resolved by a member of the ONAP community, upgrading the TCA to version 1.2.2, which caused the blueprint to change.

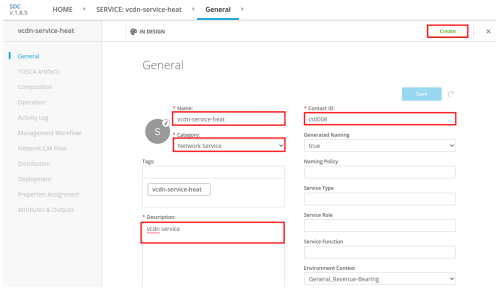
This will allow deploying policies in CLAMP to this specific TCA instance, which is a requirement for defining which thresholds will trigger the control loop. This configuration will be addressed in section 5.3.2.

After the service is created and has all the necessary requirements, the SDC will distribute it to the ONAP system, making it available for use in other components. In this case, the service will be available in CLAMP for further configuration.

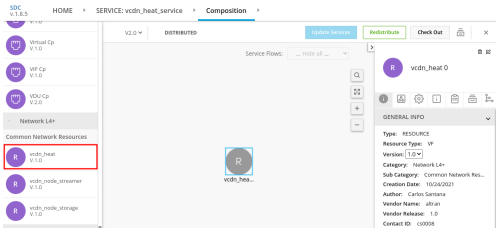
In figure 5.3 it's represented the process of deploying the service in the SDC.



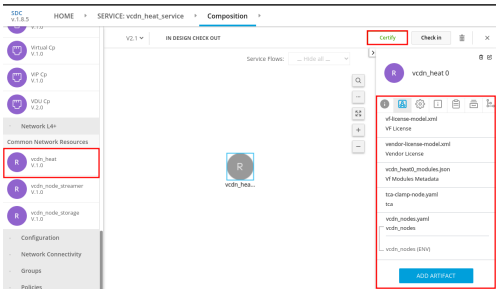
(a) Add Service Option



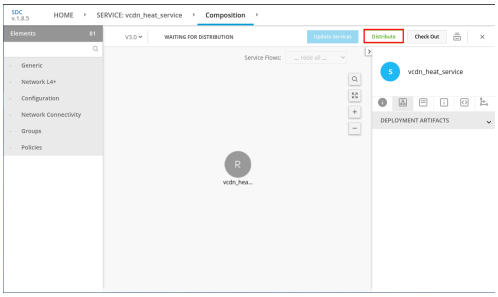
(b) Fill mandatory input parameters



(c) Drag & Drop the desired VF for the service



(d) Add artifact to the VF node



(e) Distribute the service

Figure 5.3: Sequence of actions to create a Service

[5] `https://git.onap.org/integration/tree/docs/files/scaleout/latest-tca-guilin.yaml`
[6] `https://lists.onap.org/g/onap-discuss/topic/84607999#23466`

At this point, the service is ready to be manipulated in CLAMP and create the necessary policies.

5.3.2 Policy Framework (CLAMP) Configuration

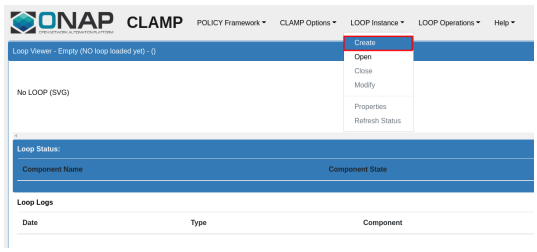
The Policy Framework has the objective of storing all the created policies, relying on many other components to achieve it. Some of those components are required to design the control loop, the policies, and load them to the respective engine. The CLAMP project is used for designing, while the Policy Drools PDP Engine and Policy XACML PDP will be used to load those policies. Since both engines are TOSCA compliant, CLAMP will create TOSCA compliant policies that each engine will translate to the respective format.

Before beginning the implementation, the service must be distributed by SDC to the rest of the ONAP components, including CLAMP, becoming possible to proceed with the configuration and instantiation of the control loop.

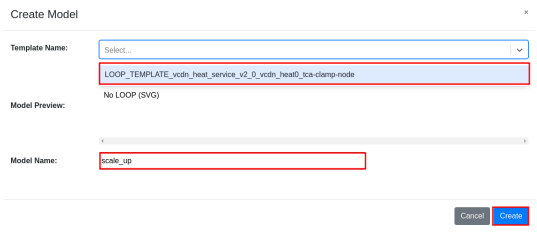
CLAMP provides the users with a user interface to declare the intended policies in each component. For the proposed use case, two distinct policies are required, one for TCA, which will be in charge of detecting the threshold crossings for the metrics received from the VES Collector and, an operational policy that will decide how the control loop will behave once a crossing is detected.

As mentioned in section 5.2, the TCA will generate an ONSET message every time a crossing is detected, triggering the operational policy, thus, performing the control loop.

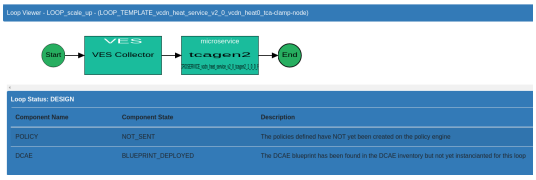
Figure 5.4 represents the first stage of the process, the creation of the control loop. The relevant fields are all highlighted in red.



(a) Create Option



(b) Fill the form with required parameters

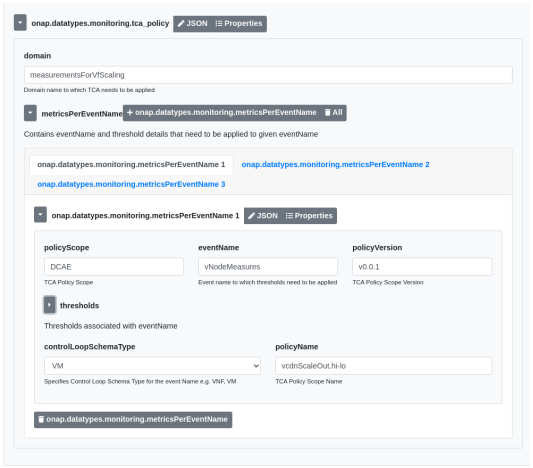


(c) Loop Created

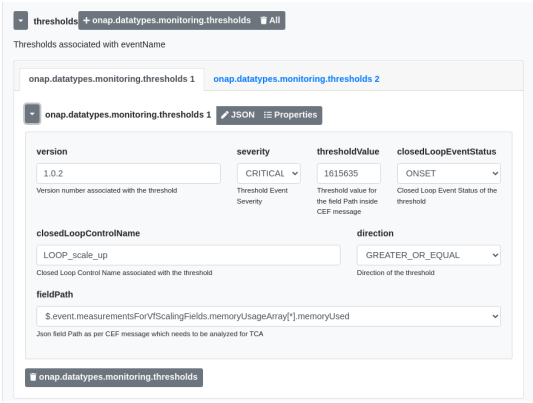
Figure 5.4: Sequence of actions to create a Control Loop

In section a) and b) of Figure 5.4 we can see the creation of the control loop. More specifically, on b), the service previously distributed to CLAMP will appear in the templates available to modify. This action is possible due to the artifact for the TCA microservice being uploaded on the service creation phase. Otherwise, the service would not be eligible for serving as a template for the control loop.

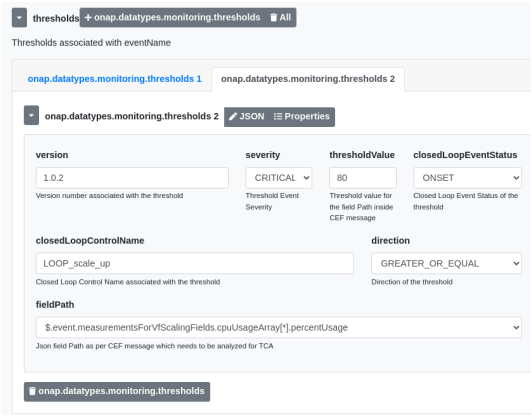
At this point, the control loop was created, connecting the VES Collector and the TCA instantiated on the service. With the TCA now available, it is possible to configure it as represented in Figure 5.5 The menu for inputting the parameters will appear after clicking on the TCA block in the diagram.



(a) Event Names



(b) Memory Threshold



(c) CPU Threshold

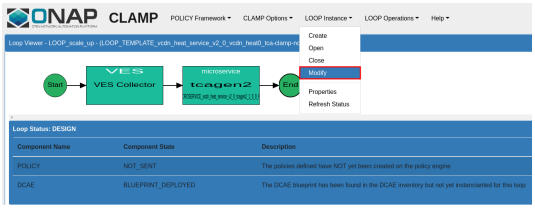
Figure 5.5: TCA Configuration

As it is possible to see in Figure 5.5, there are many relevant parameters to be filled in each field.

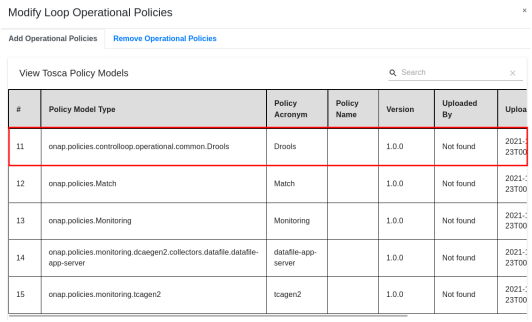
In 5.5a), the most relevant fields are the *domain* and *eventName* that must be matched with the fields sent by the VES collector in order to properly function. Also, the *controlLoop-SchemaType* must be set to *VM* type to be able to fetch the correct information from the reporting entity, which is registered as a vServer on the A&AI. If the type is set to *VNF*, the A&AI will try to search for a VNF name that does not exist in Inventory and fail to produce a successful result for the control loop.

In 5.5b) and 5.5c) is the configuration for which thresholds the TCA will generate ONSET messages. The most relevant field here is the *closedloopControlName* that must match the name given to the control loop at creation, with *LOOP_* as prefix.

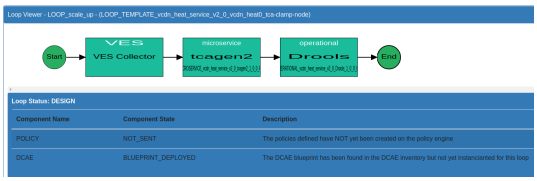
For the control loop to be complete, it requires the operational policy. The illustration of the process on how to add the operational policy to the control loop is in Figure 5.6



(a) Modify Option



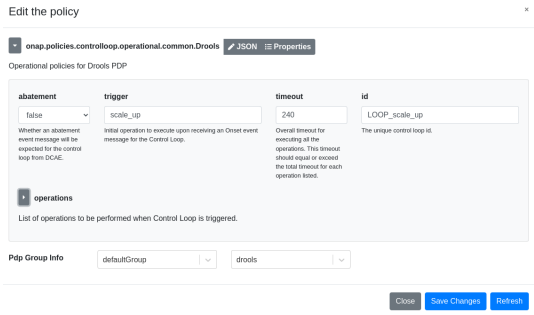
(b) Operational Policy



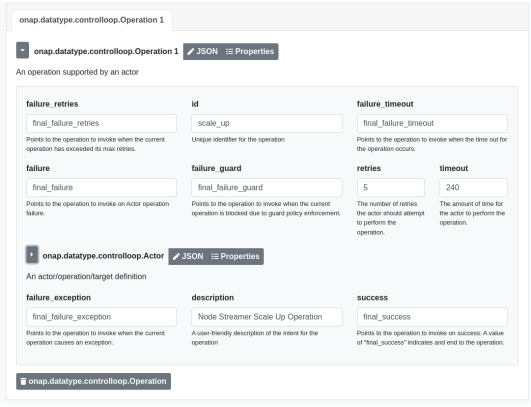
(c) Control Loop Completed

Figure 5.6: Adding an operational policy to the control loop

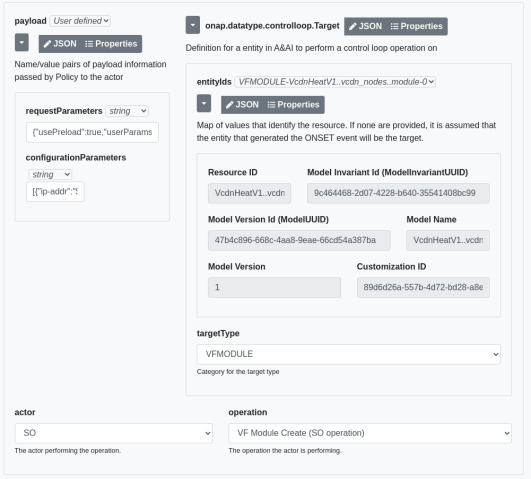
The last step for the configuration process of the control loop is to modify the default operational Drools policy design. The same process applies to opening the policy configuration window as before with TCA. Process illustrated in Figure 5.7.



(a) Operational Policy Header



(b) Operational Policy Operations



(c) Operational Policy Actor

Figure 5.7: Operational Policy Configuration

To properly define the operational policy, it also must be considered that the value of some fields must be correctly defined to link each policy and operation defined in it. The field *trigger* in the policy header (Figure 5.7a)) must match the *id* in Figure 5.7b) in order to execute the correct operation when the control loop is triggered.

In Figure 5.7c), it shows the configuration of the correct actor and the correct reference for the VF Module that needs to be instantiated when the control loop activates.

With the configuration for each policy ready, they now need to be deployed into their

respective engines. TCA policy will be deployed into the Policy XACML PDP Engine and the operational Drools policy into Policy Drools PDP Engine.

CLAMP also provides a mechanism to create and submit the policies in each engine. This deployment can be done manually by accessing each one through their offered APIs[7], which is, essentially, the same process CLAMP does but automates it in a more user-friendly approach.

After submitting each policy, there is only one last step to complete the instantiation of the control loop, which is deploying the microservice to DCAE. Once again, CLAMP provides an easy and intuitive way to perform this action with a menu option that automatically executes all the intermediary deployment steps.

Both the submission of the policies and the deployment are illustrated in Figure 5.8.

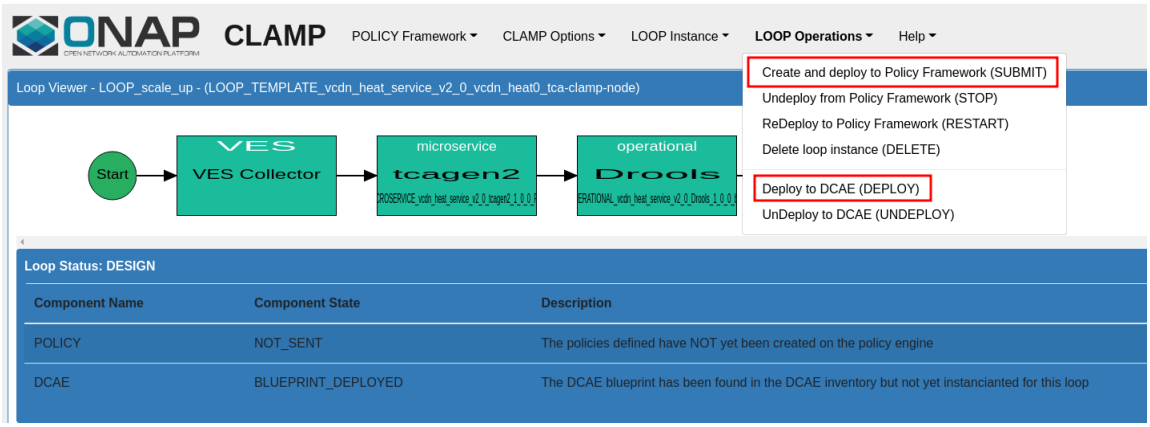


Figure 5.8: Policies submission and DCAE deployment

A simplified version of the entire process can be visualized in Figure 5.9.

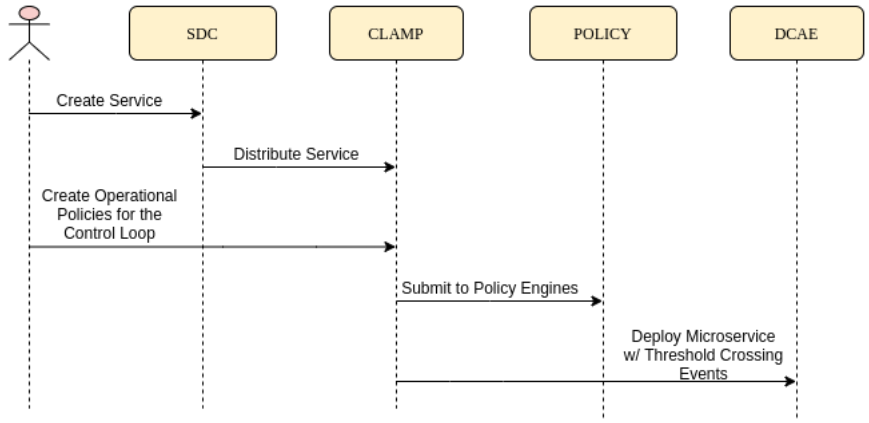


Figure 5.9: CLAMP configuration sequence

Finally, the control loop is ready to be used by the ONAP framework and provides automation for scale-out operations. The tests and results made on the control loop will be explained in Chapter 6.

[7] `https://docs.onap.org/projects/onap-policy-parent/en/latest/offeredapis.html`

All the policies configurations are available for consulting in Appendix B. The figures relative to the implementation are also provided in Appendix C for better resolution.

CHAPTER 6

Validation and Analysis

This chapter presents the tests executed for the implemented solution. The main focus of this chapter is to explain how the tests operate and how they validate the envisioned scenario for the vCDN scale out with the execution of a control loop defined in the ONAP framework. The tests are divided into different phases, first, validating the monitoring solution for the vCDN service, and the other validating the correctness and effectiveness of the control loop.

The tests were divided into two stages to evaluate if the presented solution was viable.

First of all, the proposed system focused heavily on providing a monitoring solution to the vCDN system, and validation of the ONAP components was dependant on the metrics that the monitoring could provide, so it was of utmost importance to have a reliable and solid system monitoring.

We can then separate the tests into the following two phases:

- Validation of the monitoring system
- Validation of the control loop

The testing was addressed in this manner because it was not possible to stress test the vCDN nodes. Thus, to test its functionality, the message that would trigger a response from the control loop was forged.

Each one of these phases are detailed in the following sections of this chapter.

6.1 Monitoring System Testbed and Validation

The monitoring system had essentially two requirements: Extract metrics from the target system and send them over to the VES Collector.

The first approach to validating this solution was to deploy the Prometheus Framework with the node exporter and configure it to collect metrics.

The Virtual Machine that was used did not belong to the vCDN system. For this test, the infrastructure provided was an Openstack instance with 2 vCPUs, 4GB RAM, and 40GB of

disk space. In terms of computational power, it is the same resources that the vCDN nodes use after being instanced except for disk space which is only 20GB for each node.

After Prometheus is set up, the script is deployed on the Virtual Machine and starts reading and converting the values to the format accepted by VES Collector.

Since, to validate the monitoring system, the entire ONAP system is not required, it was used a standalone[1] version of the VES Collector.

Once the conversion is made, and a new message is constructed, it is validated against the schema by the script itself and sends the message to the collector if all is valid.

If all these components are functioning correctly, it is safe to assume that the connection between the three essential elements - VM, Monitoring System, and ONAP - is fully functional and ready to be deployed on the vCDN nodes. Note that it is possible to input a custom endpoint for the VES Collector before running the script, so it's only needed to change from the standalone VES Collector to the ONAP endpoint.

Finally, the commands to deploy Prometheus Framework and the script were added to the HEAT template to instantiate the vCDN service. Once again tested, now on the vCDN nodes, to check the connectivity between all the components, and it succeeded, validating the monitoring system deployment.

6.2 Control Loop Validation

The validation of the control loop is the second essential part of the entire system and what executes the vCDN service action per the use case scenario.

As previously stated, to test this part of the system, the messages to trigger the policies in the control loop had to be forged.

To simulate the message, the monitoring script was used to generate one with the actual metrics being collected from the testing Virtual Machine. Only the values that would cause a threshold crossing in the TCA microservice were altered. This step also helped to validate the correctness of the generated messages from the script once again. The next step was to trigger the control loop with the altered message.

To do this, Postman [2] was used to send the message, with the altered value to trigger the control loop, to the VES Collector endpoint in the ONAP Framework.

From this point, after sending the message, the control loop testing begins.

The first component to monitor is the VES Collector, which already returns a response to Postman indicating if the action was successful or not. If the VES accepts the message, it will forward it to the DMaaP topic where TCA is listening to incoming messages.

When TCA finally reads the message, the following message is presented (Figure 6.1):

[1] `https://docs.onap.org/en/elalto/submodules/dcaegen2.git/docs/sections/services/ves-http/installation.html`

[2] `https://www.postman.com/`

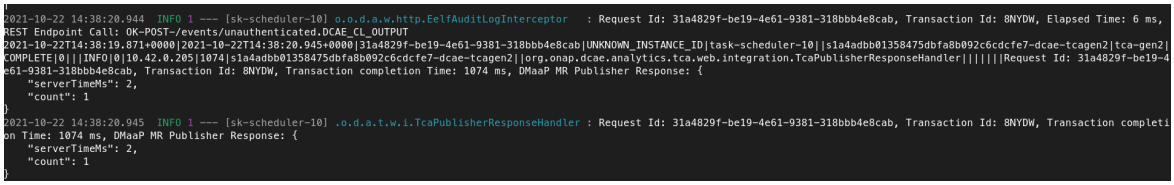


Figure 6.1: TCA information after successfully receiving the message from VES

TCA generates the ONSET message to send to the Policy Framework. Once it is received, it will trigger the policy involved in executing the control loop, as it is shown in Figure 6.2.

Figure 6.2: Policy message after receiving the ONSET event from TCA

The Policy Framework then proceeds with a series of intermediate steps to acquire all the necessary information for the request and send it to the Service Orchestrator for further processing.

When the Policy finishes gathering all the data, the Service Orchestrator will receive the message from Policy and start to execute the workflow associated with the scale-out command.

SO will send a message reporting the success of the operation, as demonstrated in Figure 6.3.

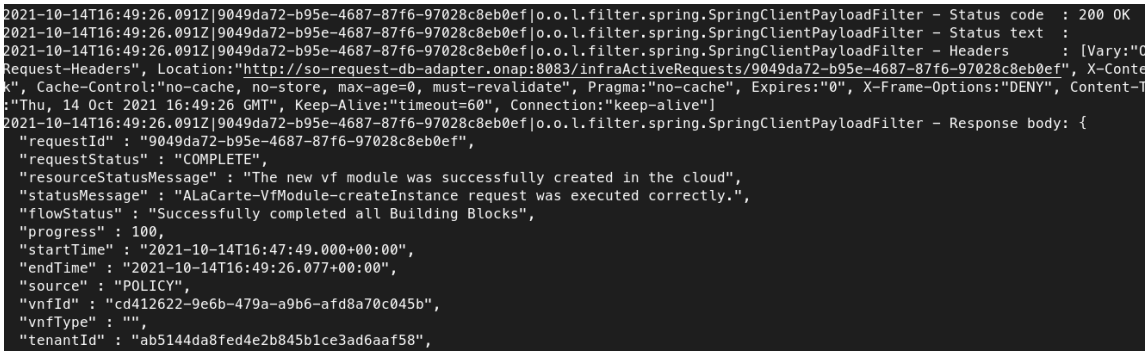


Figure 6.3: SO reporting the successful operation "VF Module Create"

Once SO reports this to Policy, the operation completes and, Policy Framework also declares it as a successful operation (Figure 6.4).

```
 "notificationTime": "2021-10-14 16:49:31.574555+00:00",
 "history": [
   {
     "actor": "SO",
     "operation": "VF Module Create",
     "target": "OperationalTarget(targetType=VFMODULE, entityIds={resourceID=VcdnHeatV1..vcdn_nodes..module-0, modelInvariantId=9c4
54468-2d07-4228-b640-35541408bc99, modelVersionId=47b4c896-668c-4aa8-9eae-66cd54a387ba, modelName=VcdnHeatV1..vcdn_nodes..module-0,
odelVersion=1, modelCustomizationId=89d6d26a-557b-4d72-bd28-a8ea9ac8a032})",
     "start": 1634230068981,
     "end": 1634230171573,
     "subRequestId": "9049da72-b95e-4687-87f6-97028c8eb0ef",
     "outcome": "Success",
     "message": "200 Success"
   }
 ]
```


Figure 6.4: Policy success notification

In Figure 6.5 it is possible to see a simplified sequence diagram illustrating the control loop interaction.

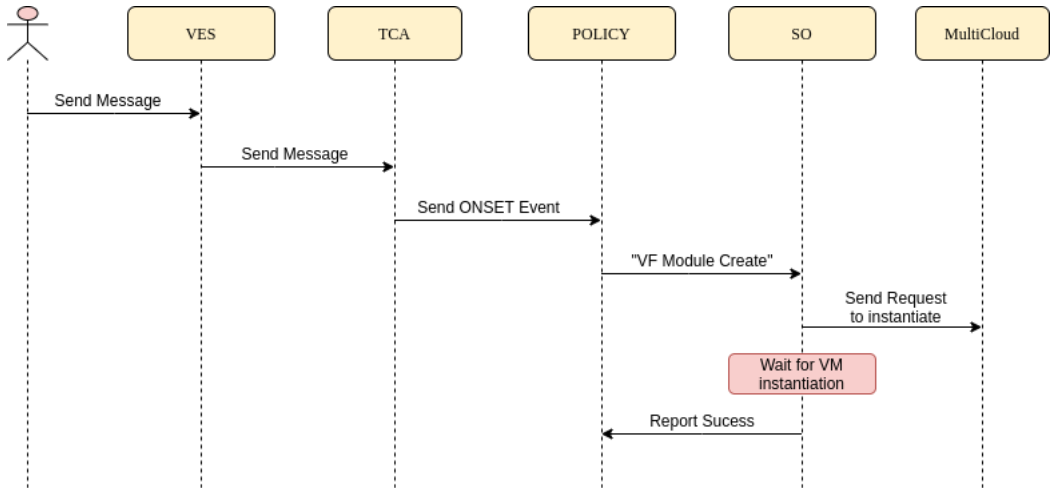


Figure 6.5: Control Loop Sequence

Having validated the control loop, tests were run afterward to measure the time instantiating a new vCDN node would take from the moment the control loop was triggered.

6.3 Testing and Evaluation

After validating both the monitoring system and the control loop, a testing phase for the control loop was carried out. The objective was to measure the time it would take to instantiate a new node after detecting an anomaly.

For this, a script was built in Python. It had the function of sending the anomaly message, measuring the time it took to instantiate the new node, delete the newly deployed VM after stopping the timer, and repeating the process.

To accurately measure this, the timer would start as soon as the anomaly was sent from the VM and then continuously probe the A&AI via its API to verify when the new instance was in the state *Active*. The results were then parsed to a CSV file.

A total of hundred and eighteen tests were executed, always with the same logic to measure the elapsed time. To better analyze these values, Table 6.1 shows the minimum, maximum, average, and standard deviation of all the tests performed.

	Min	Max	Avg	SD
Elapsed Time (s)	69.93	236.4	146.5	48.94

Table 6.1: Elapsed time for node instantiation

As we can see from the results, there were very low times and some spikes up to a maximum o 236.4 seconds, but the average time was acceptable. Something that could lead to the higher times is that the Openstack environment was not used exclusively for the testing. Other instances were active and performing activities, which could lead to some delays in Openstack.

Overall, with 146.5 seconds for average time for instantiation, it shows that the quality of service can be maintained in a fast and automated way, without the need for the usual human system maintenance, improving the Quality of Service vCDN system.

The results obtained regarding the instantiation times and the framework's validation show promising results. The automated orchestration of the network is a step further to improve the Quality of Service on the networks.

With the present solution, the scenarios in which it could be applied are endless. The monitoring solution was made to be versatile to integrate with any system running on the 5G Network.

Also, ONAP provides a solution that is possible to adapt to other systems without altering the policies that are already implemented.

Both components of the development achieved great flexibility to be adopted into other systems. With the promising results in the instantiation times, the services on the networks can start to slowly migrate into fully automated management systems.

CHAPTER 7

Conclusion

The objective of this thesis was to develop a mechanism that would provide a way to assure the quality of the vCDN service in a heavy workload environment and study the effectiveness of the solution implemented.

Although it was not possible to test the fully functional system on a testbed that would resemble more a real-life scenario, the results taken from the tests in a controlled environment are also very promising, considering that there are close to no adaptations needed to make in the deployed solution to be able to adapt it in a real-world system.

In Chapter 2 was researched the most relevant and emerging technologies that provide the necessary knowledge for the design and implementation of this project's final solution. It was a vital step for understanding all the underlying technologies at hand to deliver the services of modern mobile networks. This chapter begins with an explanation of the 5G Network. It proceeds to describe the most important key enablers, which are Software Defined Networks, Network Functions Virtualisation, and Multi-Access Edge Computing, for implementing the 5G use cases. These concepts are also the enablers of the vCDN service, which is used, intending to improve and expand its functionalities as the end goal.

Since this thesis was mainly focused on monitoring the vCDN service, an analytical study on the available monitoring tools had to be conducted to discover the best-suited solution to deploy in conjunction with the vCDN system. The chapter finishes with an overview of the most common orchestration solutions available in the market, which led to acquiring knowledge on how network orchestration is achieved with the capability of these tools.

With all the information gathered, it was possible to envision the overall system architecture based on the proposed use case and the methodology to develop the solution. The architecture proposed was aligned to provide the vCDN service with a scale-out solution for when the system was overloaded. Besides the previous study, since the ONAP framework is highly modular, an assessment of the required system components was also executed.

All the work until here enabled the beginning of the system implementation. Although ONAP is a powerful platform with a great community supporting it, it is not an easy framework to work.

It has a very steep learning curve, and each time a new component of the framework is added to the equation, the curve only gets steeper. The required elements for the implemented solution were numerous, which added even more complications.

ONAP is a framework that requires a lot of computational power to operate correctly. For the particular solution of this thesis, which required so many active components simultaneously, it was troublesome, even for Capgemini Engineering, to find a suitable machine that had all the necessary resources to run the system flawlessly.

Despite the encountered problems with the ONAP framework, after the successful implementation and validation of the use case, the results showed that a scale-out control loop is possible and viable in the vCDN system, without requiring a great amount of time needed to set up and instantiate a new node, offering the possibility of a solution for service assurance in the vCDN service and also also the possibility of expanding this solution to other systems on the edge of the network.

7.1 Future Work

ONAP will keep evolving in the future, and therefore, its capabilities will also be expanded, allowing for more use cases to be addressed. In the vCDN service, there is still room for several improvements.

First of all, a logical expansion to the vCDN is to detect when the service is not being used in a way that justifies more than one instance active at that time and performs a scale in, all in an automated fashion. Also, to provide better results, the development of a load balancer to the vCDN service would be optimal to prove that the scaling out of the system with just one VM was sufficient to improve the previous overloaded edge nodes. In this case, after the implementation of the load balancer, it could provide valuable insights to make the intended stress test, addressed previously in Chapter 3.

Regarding the monitoring side of the implementation, it could be interesting to adapt the solution to work in conjunction with Artificial Intelligence and even create a new microservice to deploy on DCAE to better evaluate the needs of the vCDN nodes instead of using fixed thresholds with the TCA microservice. This could even bring more advantages for predicting when the service would be under heavy workloads and begin the instantiation of new nodes before the system is overloaded.

Lastly, blockchain is a relatively new technology. Some studies are beginning to emerge for Service Layer Agreements management using the blockchain and smart contracts to guarantee a more secure and trustworthy deployment of policies in cloud environments. This could provide valuable but challenging research on integrating this type of technology with the available orchestration platforms and arm the 5G services with better security.

References

[1] Cisco, "Cisco visual networking index (VNI) global mobile data traffic forecast update, 2017-2022 white paper," *Computer Fraud & Security*, pp. 3–5, 2019. [Online]. Available: http://www.gsma.com/spectrum/wp-content/uploads/2013/03/Cisco_VNI-global-mobile-data-traffic-forecast-update.pdf.

[2] *Mobile data traffic forecast – Mobility Report - Ericsson.* [Online]. Available: https://www.ericsson.com/en/mobility-report/dataforecasts/mobile-traffic-forecast.

[3] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013, ISSN: 01636804. DOI: 10.1109/MCOM.2013.6461195.

[4] K. Joshi and T. Benson, "Network Function Virtualization," *IEEE Internet Computing*, vol. 20, no. 6, pp. 7–9, 2016, ISSN: 10897801. DOI: 10.1109/MIC.2016.112.

[5] S.A. Altice Labs, Altran, IT, IT Center, Nokia, Onesource, PDM&FC, Ubiwhere, Universidade de Coimbra, "Mobilizador 5G - D2.1," no. March, pp. 1–87, 2018.

[6] J. Aires, P. Duarte, B. Parreira, and S. Figueiredo, "Phased-vCDN Orchestration for flexible and efficient usage of 5G edge infrastructures," *IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2019 - Proceedings*, 2019. DOI: 10.1109/NFV-SDN47374.2019.9040097.

[7] ——, "An orchestrated 5g-enabled deep vcdn system," *Internet Technology Letters*, vol. 3, no. 6, e189, 2020. DOI: https://doi.org/10.1002/itl2.189. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/itl2.189. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/itl2.189.

[8] 5G PPP Architecture Working Group, "View on 5G Architecture," *Version 3.0, June 2019*, no. June, pp. 21–470, 2019. DOI: 10.5281/zenodo.3265031. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2019/07/5G-PPP-5G-Architecture-White-Paper_v3.0_PublicConsultation.pdf.

[9] International Telecommunication Union, *Setting the scene for 5G: Opportunities & Challenges*, July. 2018, pp. 1–56, ISBN: 9789261275815.

[10] 3GPP, "3GPP Tr 22.829," vol. 0, no. Release 16, pp. 1–53, 2019.

[11] European Telecommunications Standards Institute, "5G; Service requirements for next generation new services and markets (3GPP TS 22.261 version 15.5.0 Release 15)," vol. 0, p. 53, 2018. [Online]. Available: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx.

[12] NGMN, "5G White Paper," *A Deliverable by the NGMN Alliance*, p. 124, 2015. [Online]. Available: https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2015/NGMN_5G_White_Paper_V1_0.pdf.

[13] NGMN Alliance, "Service-Based Architecture in 5G," pp. 1–17, 2018.

[14] ETSI, "TS 128 535 - V16.0.0 - 5G; LTE; Management and orchestration; Management services for communication service assurance; Requirements (3GPP TS 28.535 version 16.0.0 Release 16)," vol. 0, 2020.

[15] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (sdn): A survey," *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833, 2016. DOI: https://doi.org/10.1002/sec.1737. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.1737. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1737.

[16] O. N. Foundation, "SDN Architecture," *SDN Architecture*, no. 1.1, 2016. DOI: 10.1007/978-3-319-46769-6_3.

[17] M. Olsson, S. Sultana, S. Rommer, L. Frid, and C. Mulligan, "Architecture Overview," *EPC and 4G Packet Networks*, pp. 17–64, 2013. DOI: 10.1016/b978-0-12-394595-2.00002-5.

[18] P. Greendyk, A. Parikh, and S. Tripathi, "Network Functions Virtualisation – Introductory White Paper," *Building the Network of the Future: Getting Smarter, Faster, and More Flexible with a Software Centric Approach*, no. 1, pp. 293–329, 2017. DOI: 10.1201/9781315208787.

[19] B. Canada, J. Erfanian, B. B. Smith, P. Willis, P. Eardley, A. Leadbeater CableLabs, T. Nakamura, D. Clarke, S. Goeringer CenturyLink, M. Bugenhagen, C. -, J. Huang, R. Ren, P. Zhao, L. Deng, K. Obana, A. Khan, J. Triay Marques Orange, B. Chatras, C. Lac Portugal Telecom, A. Gamelas, J. Carapinha Rogers, A. S. Markman Telecom, D. Lee, W. Choi, K. Kim Sprint, S. Manning STC, A. I. Alsubhi, S. S. Mahmoud Swisscom, M. Brunner Telecom Italia, C. Corbi, E. Demaria Telefonica, D. López, F. Javier Ramón Salguero, A. Elizondo Armengol Telenor, P. Grønsund, P. Waldemar Vodafone, S. Sabater, and A. Neal, "Network Functions Virtualisation – White Paper on NFV priorities for 5G," *ETSI White Paper*, no. 1, pp. 1–15, 2017. [Online]. Available: https://portal.etsi.org/nfv/nfv_white_paper_5g.pdf.

[20] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrik, "Architectural Framework," *Rationale-Based Software Engineering*, vol. 1, pp. 241–254, 2008. DOI: 10.1007/978-3-540-77583-6_17.

[21] ETSI, *Etsi gs nfv-man 001 v1.1*. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf.

[22] ——, "Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework," *ETSI GS NFV-EVE 005 V1.1.1*, 2015, ISSN: 19448007. DOI: 10.1029/90GL02597.

[23] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, K.-W. Wen, K. Kim, R. Arora, A. Odgers, L. M. Contreras, and S. Scarpina, "MEC in 5G networks," *ETSI White Paper*, no. 28, pp. 1–28, 2018. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf.

[24] ETSI, "Multi-access Edge Computing (MEC); Framework and Reference Architecture," vol. 1, pp. 1–21, 2020.

[25] A. Kapadia, *ONAP Demystified*. 2018, ISBN: 1720879915.

[26] T. Binz, G. Breiter, F. Leyman, and T. Spatzier, "Portable cloud services using TOSCA," *IEEE Internet Computing*, vol. 16, no. 3, pp. 80–85, 2012, ISSN: 10897801. DOI: 10.1109/MIC.2012.43.

[27] P. O. F. Turin, "Master Degree Thesis in Computer Engineering Prototyping a Network Service based on the ONAP Platform," 2019.

[28] E. Summary and C. N. Overview, "The Cisco Content Delivery Network," pp. 1–18, 2000.

[29] D. Santos, R. Silva, D. Corujo, R. L. Aguiar, and B. Parreira, "Follow the User: A Framework for Dynamically Placing Content Using 5G-Enablers," *IEEE Access*, vol. 9, pp. 14 688–14 709, 2021, ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3051570.

[30] R. Buyya, M. Pathan, and A. Vakali, *Content Delivery Networks*, ser. Lecture Notes in Electrical Engineering. Springer Berlin Heidelberg, 2008, ISBN: 9783540778875. [Online]. Available: https://books.google.pt/books?id=p2C2cZkTrmsC.

[31] N. Bartolini, E. Casalicchio, and S. Tucci, "A walk through content delivery networks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2965, pp. 1–25, 2004, ISSN: 16113349. DOI: 10.1007/978-3-540-24663-3_1.

[32] *Overview | prometheus*. [Online]. Available: https://prometheus.io/docs/introduction/overview/.

[33] *Comparison to alternatives | prometheus*. [Online]. Available: https://prometheus.io/docs/introduction/comparison/#prometheus-vs-nagios.

[34] *Prometheus vs nagios | logz.io*. [Online]. Available: https://logz.io/blog/prometheus-vs-nagios-metrics/.

[35] A. P. SA, "OREOS - Orchestration and Resource optimization for rEliable and lOw-latency Services - D2.1," 2021.

[36] *Introduction — onap master documentation*. [Online]. Available: https://docs.onap.org/en/honolulu/guides/onap-developer/architecture/onap-architecture.html.

[37] N. Simoes, "NFV Management and Orchestration: Analysis of OSM and ONAP," pp. 1–5, 2019.

[38] V. Q. Rodriguez, F. Guillemin, and A. Boubendir, "5G E2E Network Slicing Management with ONAP," *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN 2020*, no. Icin 2020, pp. 87–94, 2020. DOI: 10.1109/ICIN48450.2020.9059507.

[39] *Service Design and Creation (SDC) - Developer Wiki - Confluence*. [Online]. Available: https://wiki.onap.org/pages/viewpage.action?pageId=1015837.

[40] *Master Service Orchestrator (MSO) - Developer Wiki - Confluence*. [Online]. Available: https://wiki.onap.org/pages/viewpage.action?pageId=1015834.

[41] *Introduction — onap master documentation*. [Online]. Available: https://docs.onap.org/en/guilin/guides/onap-developer/architecture/onap-architecture.html#onap-architecture.

[42] *Active and Available Inventory (AAI) - Developer Wiki - Confluence*. [Online]. Available: https://wiki.onap.org/pages/viewpage.action?pageId=1015836.

[43] *Controllers - Developer Wiki - Confluence*. [Online]. Available: https://wiki.onap.org/display/DW/Controllers.

[44] "OpenECOMP Application Controller User Guide," no. January, 2017.

[45] *DMaaP – Data Movement as a Platform proposal - Developer Wiki - Confluence*. [Online]. Available: https://wiki.onap.org/pages/viewpage.action?pageId=3247130.

[46] *Policy Framework Architecture — onap master documentation*. [Online]. Available: https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html#overview.

[47] *Architecture — onap master documentation*. [Online]. Available: https://docs.onap.org/projects/onap-dcaegen2/en/latest/sections/architecture.html.

[48] *VNF Event Streaming (VES) Collector — onap master documentation*. [Online]. Available: https://docs.onap.org/projects/onap-dcaegen2/en/latest/sections/services/ves-http/index.html?highlight=ves.

[49] *CLAMP - Control Loop Automation Management Platform — onap master documentation*. [Online]. Available: https://docs.onap.org/projects/onap-clamp/en/latest/.

[50] B. Sousa, J. Luís, N. Lourenço, M. Curado, M. Araujo, B. Parreira, K. Velasquez, and D. Abrey, "Arquitetura e especificação da Plataforma OREOS," 2021.

[51] B. Parreira, P. Duarte, J. Silva, D. Corujo, R. Silva, and D. Santos, "Dynamic vCDN deployment in Edge Computing environments (DIVEC)),"

Appendix A

This appendix contains the monitoring system configurations for the Prometheus Server and the monitoring script.

Prometheus Configuration File

```yaml
global:
  scrape_interval:     10s
  evaluation_interval: 10s

scrape_configs:
  - job_name: "prometheus"
    static_configs:
    - targets: ["localhost:9090"]

  - job_name: "openstack"
    honor_labels: true
    openstack_sd_configs:
      - identity_endpoint: http://192.168.89.4:5000/v3
        username: #######
        project_name: IT_ALTRAN
        project_id: 1ea949d8a2c44946bd64cfc4f55149c8
        domain_id: default
        password: ################################
        role: instance
        region: RegionOne
        port: 8081

    relabel_configs:
      - source_labels: [__meta_openstack_instance_status]
        action: keep
        regex: ACTIVE

      - source_labels: [__meta_openstack_instance_name]
        target_label: instance

      - source_labels:
        - __meta_openstack_address_pool
        - __meta_openstack_tag_prometheus_io_address_pool
        action: replace
```

```
      regex: "provider;provider"
      target_label: __tmp_keep_target
      replacement: "true"

    - source_labels:
      - __meta_openstack_address_pool
      - __meta_openstack_tag_prometheus_io_address_pool
      - __tmp_keep_target
      action: replace
      regex: "provider;;"
      target_label: __tmp_keep_target
      replacement: "true"

    - source_labels: [__tmp_keep_target]
      regex: true
      action: keep

    - source_labels: [__meta_openstack_instance_name]
      regex: prometheus_monitoring_altran
      action: keep

    - source_labels: [__meta_openstack_instance_id]
      target_label: vm_id
```

Listing A.1: Prometheus Server Configuration

app.py

```
from collect_metrics import Collector
import common.common_utils as common_utils
import argparse

def start(ip, port):

    collector = Collector()
    metrics = collector.collect_readings('http://192.168.89.122:9090')
    print(metrics)

    if(common_utils.validate_json(metrics)):
        common_utils.send_message(ip, port, metrics)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-i', '--ip', type=str, help='VESCollector IP Address',
    ↪  required=False, default='10.12.82.65')
    parser.add_argument('-p', '--port', type=str, help='VESCollector Port', required=False,
    ↪  default='30417')
    args = parser.parse_args()

    IP_ADDR = args.ip
    PORT = args.port

    start(IP_ADDR, PORT)
```

Listing A.2: Main application

collect_metrics.py

```
from common.common_utils import metrics_to_dict, metrics_to_dict_mem, reorder
from prometheus_client import PrometheusClient
import converter
import time

class Collector:

    def __init__(self):
      self.vm_info  = dict()
      self.cpu_metrics = dict(dict())
      self.cpu_values = dict()
      self.memory_metrics = dict()
      self.disk_metrics = dict(dict())
      self.disk_values = dict()
      self.net_interface_metrics = dict(dict())
      self.net_values = dict()
      self.eventID = 0

    def collect_readings(self, url):

      prom = PrometheusClient(url)

      start = time.time() - 10
      end = time.time()

      #CPU Metrics

      for cpu in prom.query_range("node_cpu_seconds_total", start, end,
      ↪  '1s')["data"]["result"]:
        final_value = 0
        self.vm_info["vm_name"] = cpu["metric"]["instance"]
        self.vm_info["vm_id"] = cpu["metric"]["vm_id"]
        for values in cpu["values"]:
          final_value += float(values[1])
        self.cpu_values[cpu["metric"]["mode"]] = final_value/len(cpu["values"])
        self.cpu_metrics[cpu["metric"]["cpu"]] = self.cpu_values.copy()

      #Memory Metrics

      self.memory_metrics["total"] = metrics_to_dict_mem(url, "node_memory_MemTotal_bytes",
      ↪  start, end)
      self.memory_metrics["cached"] = metrics_to_dict_mem(url, "node_memory_Cached_bytes",
      ↪  start, end)
      self.memory_metrics["buffer"] = metrics_to_dict_mem(url, "node_memory_Buffers_bytes",
      ↪  start, end)
      self.memory_metrics["free"] = metrics_to_dict_mem(url, "node_memory_MemFree_bytes",
      ↪  start, end)
      self.memory_metrics["inactive"] = metrics_to_dict_mem(url,
      ↪  "node_memory_Inactive_bytes", start, end)
```

```
44          self.memory_metrics["slab_unrecl"] = metrics_to_dict_mem(url,
            ↪  "node_memory_SUnreclaim_bytes", start, end)
45          self.memory_metrics["slab_recl"] =  metrics_to_dict_mem(url,
            ↪  "node_memory_SReclaimable_bytes", start, end)
46
47          #Disk Metrics
48
49          self.disk_metrics["write_seconds"] = metrics_to_dict(url,
            ↪  "node_disk_write_time_seconds_total", start, end)
50          self.disk_metrics["read_seconds"] = metrics_to_dict(url,
            ↪  "node_disk_read_time_seconds_total", start, end)
51          self.disk_metrics["io_time"] = metrics_to_dict(url, "node_disk_io_time_seconds_total",
            ↪  start, end)
52          self.disk_metrics["written_bytes"] = metrics_to_dict(url,
            ↪  "node_disk_written_bytes_total", start, end)
53          self.disk_metrics["read_bytes"] = metrics_to_dict(url, "node_disk_read_bytes_total",
            ↪  start, end)
54          self.disk_metrics["writes_merged"] = metrics_to_dict(url,
            ↪  "node_disk_writes_merged_total", start, end)
55          self.disk_metrics["reads_merged"] = metrics_to_dict(url,
            ↪  "node_disk_reads_merged_total", start, end)
56
57          #Reorder dictionaries (switch keys and values)
58          disk_metrics_reordered = reorder(self.disk_metrics)
59
60          #Network Metrics
61
62          self.net_interface_metrics["rx_bytes"] = metrics_to_dict(url,
            ↪  "node_network_receive_bytes_total", start, end)
63          self.net_interface_metrics["rx_drop"] = metrics_to_dict(url,
            ↪  "node_network_receive_drop_total", start, end)
64          self.net_interface_metrics["rx_errors"] = metrics_to_dict(url,
            ↪  "node_network_receive_errs_total", start, end)
65          self.net_interface_metrics["rx_packets"] = metrics_to_dict(url,
            ↪  "node_network_receive_packets_total", start, end)
66          self.net_interface_metrics["tx_bytes"] =  metrics_to_dict(url,
            ↪  "node_network_transmit_bytes_total", start, end)
67          self.net_interface_metrics["tx_drop"] = metrics_to_dict(url,
            ↪  "node_network_transmit_drop_total", start, end)
68          self.net_interface_metrics["tx_errors"] = metrics_to_dict(url,
            ↪  "node_network_transmit_errs_total", start, end)
69          self.net_interface_metrics["tx_packets"] = metrics_to_dict(url,
            ↪  "node_network_transmit_packets_total", start, end)
70
71          #Reorder dictionaries (switch keys and values)
72
73          net_metrics_reordered = reorder(self.net_interface_metrics)
74
75          return converter.convert(self.eventID, self.vm_info, self.cpu_metrics,
            ↪  self.memory_metrics, disk_metrics_reordered, net_metrics_reordered)
```

```
76
77          #print(json.dumps(self.cpu_metrics["data"]["result"], indent=4, sort_keys=True))
78          #print(json.dumps(temp_dict_net, indent=4, sort_keys=True))
```

Listing A.3: Metrics Collection Module

converter.py

```
1   import message_data_struct.ves_54_pb2 as ves_54_pb2
2   import time
3   import os
4   import uuid
5   from google.protobuf import json_format
6
7   def convert(eventID, vm_info, cpu_metrics, memory_metrics, disk_metrics,
    ↪  interface_metrics):
8
9           message = ves_54_pb2.header()
10
11          common_header = message.event.commonEventHeader
12
13          eventID +=1
14
15          #Required fields
16          common_header.version = 3.0
17          common_header.domain = "measurementsForVfScaling"
18          common_header.eventId = ("{}{:07d}").format("Mfvs", eventID)
19          common_header.eventName = ("{}_{}_{}").format("Mfvs", os.getlogin(),
            ↪  "perf_metrics")
20          common_header.eventType = "applicationVnf"
21          common_header.startEpochMicrosec = time.time() * 1000
22          common_header.priority = "Normal"
23          common_header.reportingEntityName = vm_info["vm_name"]
24          common_header.reportingEntityId = vm_info["vm_id"]
25          common_header.sequence = 0
26          common_header.sourceName = "vCDN_Node"
27          common_header.lastEpochMicrosec = time.time() * 1000
28
29
30
31          ### Measurement Fields
32          meas_fields = message.event.measurementForVfScalingFields
33
34          # Required Fields
35          meas_fields.measurementFieldsVersion = 2.0
36          meas_fields.measurementInterval = 10
37
38          # Optional Fields (Measures)
39
40          # Memory Usage Array
```

```
41          mem_usage = meas_fields.memoryUsageArray.add()
42
43
44          if(memory_metrics):
45              if(all(value is not None for value in memory_metrics.values())):
46                  mem_usage.memoryUsed = (int(memory_metrics["total"]) -
                    ↪  int(memory_metrics["free"]) - (int(memory_metrics["cached"]) +
                    ↪  int(memory_metrics["buffer"])))/1024
47                  mem_usage.memoryFree = int(memory_metrics["free"])/1024
48                  mem_usage.memoryConfigured = int(memory_metrics["total"])/1024
49                  mem_usage.vmIdentifier = str(uuid.uuid1())
50                  mem_usage.memoryCached = int(memory_metrics["cached"])/1024
51                  mem_usage.memoryBuffered = int(memory_metrics["buffer"])/1024
52                  mem_usage.memorySlabRecl = int(memory_metrics['slab_recl'])/1024
53                  mem_usage.memorySlabUnrecl = int(memory_metrics['slab_unrecl'])/1024
54
55          # CPU Usage Array
56
57          if(cpu_metrics):
58              for metric in cpu_metrics:
59                  if(all(value is not None for value in cpu_metrics[metric].values())):
60                      cpu_usage = meas_fields.cpuUsageArray.add()
61                      cpu_usage.cpuIdentifier = str(metric)
62                      total = float(cpu_metrics[metric]['idle']) +
                        ↪  float(cpu_metrics[metric]['irq']) +
                        ↪  float(cpu_metrics[metric]['nice'])  + \
63                      float(cpu_metrics[metric]['softirq']) +
                        ↪  float(cpu_metrics[metric]['steal']) + \
64                      float(cpu_metrics[metric]['system']) +
                        ↪  float(cpu_metrics[metric]['user']) +
                        ↪  float(cpu_metrics[metric]['iowait'])
65                      cpu_usage.cpuIdle = float(cpu_metrics[metric]['idle'])
66                      cpu_usage.percentUsage = ((total -
                        ↪  float(cpu_metrics[metric]['idle']))/total) * 100
67                      cpu_usage.cpuUsageInterrupt = float(cpu_metrics[metric]['irq'])
68                      cpu_usage.cpuUsageNice = float(cpu_metrics[metric]['nice'])
69                      cpu_usage.cpuUsageSoftIrq = float(cpu_metrics[metric]['softirq'])
70                      cpu_usage.cpuUsageSteal = float(cpu_metrics[metric]['steal'])
71                      cpu_usage.cpuUsageSystem = float(cpu_metrics[metric]['system'])
72                      cpu_usage.cpuUsageUser = float(cpu_metrics[metric]['user'])
73                      cpu_usage.cpuWait = float(cpu_metrics[metric]['iowait'])
74
75          # NIC Performance Array
76          if(interface_metrics):
77                  for nic in interface_metrics:
78                      if(all(value is not None for value in
                        ↪  interface_metrics[nic].values())):
79                          nic_usage = meas_fields.nicPerformanceArray.add()
80                          nic_usage.nicIdentifier = str(nic)
81                          nic_usage.valuesAreSuspect = "false"
```

```
82                          nic_usage.receivedTotalPacketsDelta =
                            ↪  float(interface_metrics[nic]['rx_packets'])
83                          nic_usage.receivedOctetsDelta =
                            ↪  float(interface_metrics[nic]['rx_bytes'])
84                          nic_usage.receivedDiscardedPacketsDelta =
                            ↪  float(interface_metrics[nic]['rx_drop'])
85                          nic_usage.receivedErrorPacketsDelta =
                            ↪  float(interface_metrics[nic]['rx_errors'])
86                          nic_usage.transmittedOctetsDelta =
                            ↪  float(interface_metrics[nic]['tx_bytes'])
87                          nic_usage.transmittedTotalPacketsDelta =
                            ↪  float(interface_metrics[nic]['tx_packets'])
88                          nic_usage.transmittedDiscardedPacketsDelta =
                            ↪  float(interface_metrics[nic]['tx_drop'])
89                          nic_usage.transmittedErrorPacketsDelta =
                            ↪  float(interface_metrics[nic]['tx_errors'])
90
91          #Disk Performance Array
92
93          disk_usage = meas_fields.diskUsageArray.add()
94          if(disk_metrics):
95                  for disk in disk_metrics:
96                      if(all(value is not None for value in disk_metrics[disk].values())):
97                          disk_usage.diskIdentifier = str(disk)
98                          disk_usage.diskMergedReadLast =
                            ↪  float(disk_metrics[disk]['reads_merged'])
99                          disk_usage.diskMergedWriteLast =
                            ↪  float(disk_metrics[disk]['writes_merged'])
100                         disk_usage.diskOctetsReadLast =
                            ↪  float(disk_metrics[disk]['read_bytes'])
101                         disk_usage.diskOctetsWriteLast =
                            ↪  float(disk_metrics[disk]['written_bytes'])
102                         disk_usage.diskTimeReadLast =
                            ↪  float(disk_metrics[disk]['read_seconds'])
103                         disk_usage.diskTimeWriteLast =
                            ↪  float(disk_metrics[disk]['write_seconds'])
104                         disk_usage.diskIoTimeLast = float(disk_metrics[disk]['io_time'])
105
106         json_msg = json_format.MessageToJson(message)
107         return json_msg
```

Listing A.4: Converter Module

prometheus_client.py

```
1  import requests
2  class PrometheusClient:
3      def __init__(self, url):
4        self.url = url
5        self.endpoint = '/api/v1/'
6
7      def query(self, expression):
8          req = requests.get(self.url + self.endpoint + 'query', params={'query':
           ↪  expression})
9          return req.json()
10
11     def query_range(self, expression, start, end, step):
12         req = requests.get(self.url + self.endpoint + 'query_range', params={'query':
           ↪  expression, 'start': start, 'end': end, 'step': step})
13         return req.json()
```

Listing A.5: Prometheus HTTP API Module

common_utils.py

```python
from jsonschema import validate
import requests
import json
import urllib3
from prometheus_client import PrometheusClient

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

def validate_json(json_msg):

    f = open('common/CommonEventFormat_28.4.1.json', 'r')
    schema = json.loads(f.read())
    try:
        validate(instance = json.loads(json_msg), schema = schema)
    except Exception as e:
        print(e)
        return False
    return True

def send_message(ip, port, msg):
    url = ("https://{}:{}/eventListener/v5").format(ip, port)
    headers = {'Authorization': 'Basic c2FtcGxlMTpzYW1wbGUx', 'Content-Type':
    ↪  'application/json'}
    response = requests.post(url, headers=headers, data=msg, verify=False)
    print(response.text)

def metrics_to_dict(url, query, start, end):
    prom = PrometheusClient(url)
    entity_values = dict()
    for entity in prom.query_range(query, start, end, '1s')["data"]["result"]:
        final_value_disk = 0
        for values in entity["values"]:
            final_value_disk += float(values[1])
        entity_values[entity["metric"]["device"]] = final_value_disk/len(entity["values"])

    return entity_values.copy()

def metrics_to_dict_mem(url, query, start, end):
    prom = PrometheusClient(url)
    single_value = 0
    for entity in prom.query_range(query, start, end, '1s')["data"]["result"]:
        final_value_disk = 0
        for values in entity["values"]:
            final_value_disk += float(values[1])
        single_value = final_value_disk/len(entity["values"])

    return single_value

def reorder(metric_dict):
```

```
    temp_dict = dict(dict())
    temp_dict2= dict()
    temp_list_metrics = []
    temp_list_entity = []
    for metric in metric_dict:
        temp_list_metrics.append(metric)
        for value in metric_dict[metric]:
            temp_list_entity.append(value)


    temp_dict = dict.fromkeys(list(temp_list_entity))
    temp_dict2 = dict.fromkeys(list(temp_list_metrics))

    for value in temp_dict:
        temp_dict[value] = temp_dict2.copy()

    for metric in metric_dict:
        for value in temp_dict:
            temp_dict[value][metric] = metric_dict[metric][value]
    return temp_dict
```

Listing A.6: Common utilities package

ves_54.proto

```
// Compile with
// protoc -I=. --python_out=. ./ves_54.proto

syntax = "proto2";

message header{
    required event event = 1;
}

message event{
    required commonEventHeader commonEventHeader = 1;
    optional MeasurementFields measurementForVfScalingFields = 2;
}

message commonEventHeader{
    required double version = 1;
    required string eventName = 2;
    required string domain = 3;
    required string eventId = 4;
    optional string eventType = 5;
    optional string nfcNamingCode = 6;
    optional string nfNamingCode = 7;
    optional string sourceId = 8;
    required string sourceName = 9;
    optional string reportingEntityId = 10;
```

```
    required string reportingEntityName = 11;
    required string priority = 12;
    required double startEpochMicrosec = 13;
    required double lastEpochMicrosec = 14;
    required int32 sequence = 15;
}

message MeasurementFields{
    required double measurementFieldsVersion = 1;
    required double measurementInterval = 2;
    repeated memoryUsageArray memoryUsageArray = 3;
    repeated cpuUsageArray cpuUsageArray = 4;
    repeated diskUsageArray diskUsageArray = 5;
    repeated nicPerformanceArray nicPerformanceArray = 6;
}

message memoryUsageArray{
    required double memoryUsed = 1;
    required double memoryFree = 2;
    required string vmIdentifier = 3;
    optional double memoryConfigured = 4;
    optional double memoryCached = 5;
    optional double memoryBuffered = 6;
    optional double memorySlabRecl = 8;
    optional double memorySlabUnrecl = 9;
}

message cpuUsageArray{
    required string  cpuIdentifier = 1;
    required double percentUsage = 2;
    optional double cpuIdle = 3;
    optional double cpuUsageInterrupt = 4;
    optional double cpuUsageNice = 5;
    optional double cpuUsageSoftIrq = 6;
    optional double cpuUsageSteal = 7;
    optional double cpuUsageSystem = 8;
    optional double cpuUsageUser = 9;
    optional double cpuWait = 10;
}

message diskUsageArray{
    required string diskIdentifier = 1;
    optional double diskMergedReadLast = 2;
    optional double diskMergedWriteLast = 3;
    optional double diskOctetsReadLast = 4;
    optional double diskOctetsWriteLast = 5;
    optional double diskOpsReadLast = 6;
    optional double diskOpsWriteLast = 7;
    optional double diskTimeReadLast = 8;
    optional double diskTimeWriteLast = 9;
```

```
76      optional double diskIoTimeLast = 10;
77  }
78
79  message nicPerformanceArray{
80      required string nicIdentifier = 1;
81      required string valuesAreSuspect = 2;
82      optional double receivedTotalPacketsAccumulated = 3;
83      optional double receivedTotalPacketsDelta = 4;
84      optional double receivedOctetsAccumulated = 5;
85      optional double receivedOctetsDelta = 6;
86      optional double receivedErrorPacketsAccumulated = 7;
87      optional double receivedDiscardedPacketsAccumulated = 8;
88      optional double receivedDiscardedPacketsDelta = 9;
89      optional double receivedErrorPacketsDelta= 10;
90      optional double transmittedDiscardedPacketsAccumulated = 11;
91      optional double transmittedErrorPacketsAccumulated = 12;
92      optional double transmittedOctetsAccumulated = 13;
93      optional double transmittedOctetsDelta = 14;
94      optional double transmittedTotalPacketsAccumulated = 15;
95      optional double transmittedTotalPacketsDelta = 16;
96      optional double transmittedDiscardedPacketsDelta = 17;
97      optional double transmittedErrorPacketsDelta = 18;
98  }
```

Listing A.7: VES Standard Message Definition (Google Protocol Buffers)

Appendix B

This appendix contains the configurations for the policies used in the control loop.

Threshold Crossing Analytics Policy

```
{
  "tca.policy": {
    "domain": "measurementsForVfScaling",
    "metricsPerEventName": [
      {
        "policyScope": "DCAE",
        "thresholds": [
          {
            "version": "1.0.2",
            "severity": "CRITICAL",
            "thresholdValue": 1615635,
            "closedLoopEventStatus": "ONSET",
            "closedLoopControlName": "LOOP_scale_up",
            "direction": "GREATER_OR_EQUAL",
            "fieldPath":
            ↪  "$.event.measurementsForVfScalingFields.memoryUsageArray[*].memoryUsed"
          },
          {
            "version": "1.0.2",
            "severity": "CRITICAL",
            "thresholdValue": 80,
            "closedLoopEventStatus": "ONSET",
            "closedLoopControlName": "LOOP_scale_up",
            "direction": "GREATER_OR_EQUAL",
            "fieldPath":
            ↪  "$.event.measurementsForVfScalingFields.cpuUsageArray[*].percentUsage"
          }
        ],
        "eventName": "vNodeMeasures",
        "policyVersion": "v0.0.1",
        "controlLoopSchemaType": "VM",
        "policyName": "vcdnScaleOut.hi-lo"
      },
      {
```

```
        "policyScope": "DCAE_Scale_Down",
        "thresholds": [
          {
            "version": "1.0.2",
            "severity": "CRITICAL",
            "thresholdValue": 403908,
            "closedLoopEventStatus": "ONSET",
            "closedLoopControlName": "LOOP_scale_down",
            "direction": "LESS_OR_EQUAL",
            "fieldPath":
            ↪  "$.event.measurementsForVfScalingFields.memoryUsageArray[*].memoryUsed"
          },
          {
            "version": "1.0.2",
            "severity": "CRITICAL",
            "thresholdValue": 20,
            "closedLoopEventStatus": "ONSET",
            "closedLoopControlName": "LOOP_scale_down",
            "direction": "LESS_OR_EQUAL",
            "fieldPath":
            ↪  "$.event.measurementsForVfScalingFields.cpuUsageArray[*].percentUsage"
          }
        ],
        "eventName": "vNodeMeasures",
        "policyVersion": "v0.0.1",
        "controlLoopSchemaType": "VM",
        "policyName": "vcdnScaleDown.hi-lo"
      },
      {
        "policyScope": "DCAE",
        "thresholds": [
          {
            "version": "1.0.2",
            "severity": "CRITICAL",
            "thresholdValue": 60,
            "closedLoopEventStatus": "ONSET",
            "closedLoopControlName": "LOOP_rebuild",
            "direction": "GREATER_OR_EQUAL",
            "fieldPath": "$.event.measurementsForVfScalingFields.cpuUsageArray[*].cpuWait"
          }
        ],
        "eventName": "vNodeRebuild",
        "policyVersion": "v0.0.1",
        "controlLoopSchemaType": "VM",
        "policyName": "vcdnRebuild.hi-lo"
      }
    ]
  }
}
```

Listing B.8: TCA Policy Configuration

Drools Operational Policy

```
{
  "abatement": false,
  "operations": [
    {
      "failure_retries": "final_failure_retries",
      "id": "scale_up",
      "failure_timeout": "final_failure_timeout",
      "failure": "final_failure",
      "operation": {
        "payload": {
          "requestParameters": "{\"usePreload\":true,\"userParams\":[]}",
          "configurationParameters":
          ↪  "[{\"ip-addr\":\"$.vf-module-topology.vf-module-parameters.param[9]\",\
          ↪  "oam-ip-addr\":\"$.vf-module-topology.vf-module-parameters.param[16]\",\
          ↪  "enabled\":\"$.vf-module-topology.vf-module-parameters.param[23]\"}]"
        },
        "target": {
          "entityIds": {
            "resourceID": "VcdnHeatV1..vcdn_nodes..module-0",
            "modelInvariantId": "9c464468-2d07-4228-b640-35541408bc99",
            "modelVersionId": "47b4c896-668c-4aa8-9eae-66cd54a387ba",
            "modelName": "VcdnHeatV1..vcdn_nodes..module-0",
            "modelVersion": "1",
            "modelCustomizationId": "89d6d26a-557b-4d72-bd28-a8ea9ac8a032"
          },
          "targetType": "VFMODULE"
        },
        "actor": "SO",
        "operation": "VF Module Create"
      },
      "failure_guard": "final_failure_guard",
      "retries": 5,
      "timeout": 240,
      "failure_exception": "final_failure_exception",
      "description": "Node Streamer Scale Up Operation",
      "success": "final_success"
    }
  ],
  "trigger": "scale_up",
  "timeout": 240,
  "id": "LOOP_scale_up"
}
```

Listing B.9: Drools Policy Configuration

Appendix C

This appendix contains the figures from Chapter 5, relative to the implementation steps.

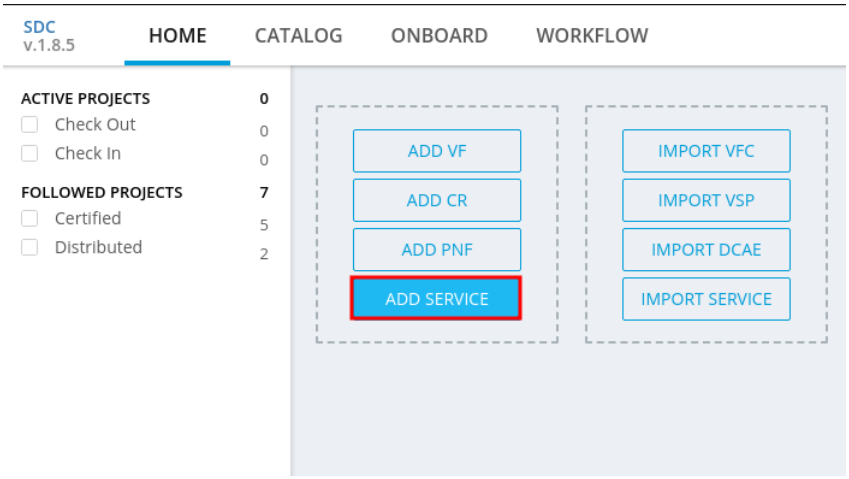


Figure relative to 5.3a)

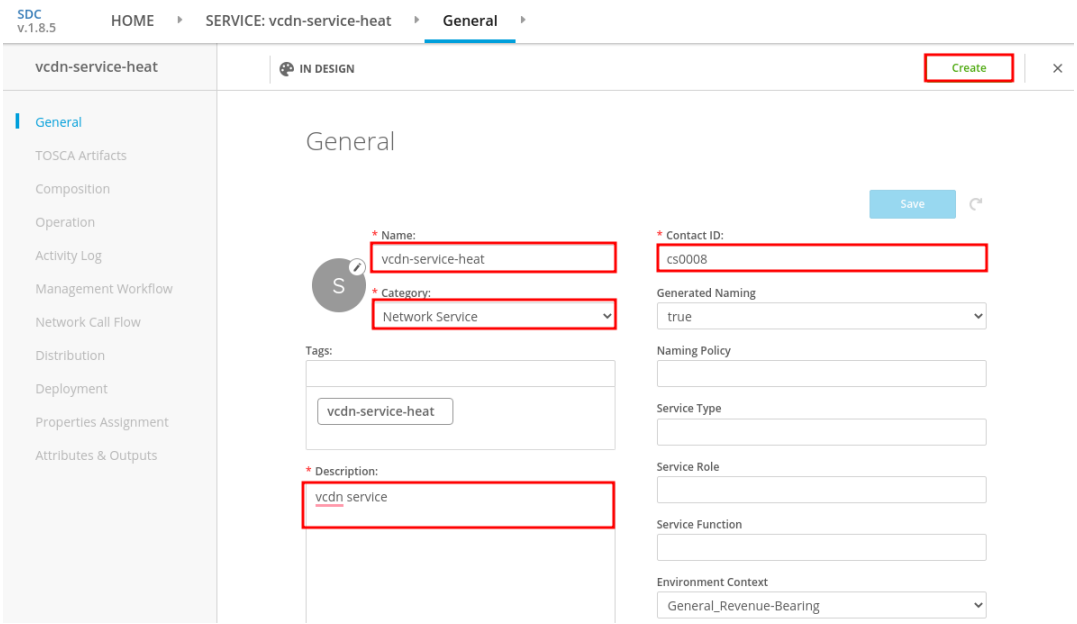


Figure relative to 5.3b)

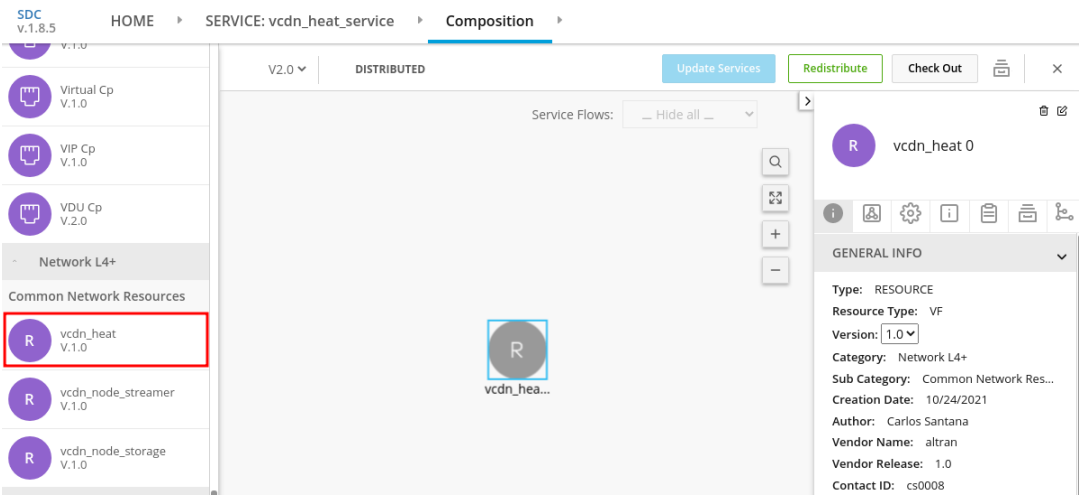


Figure relative to 5.3c)

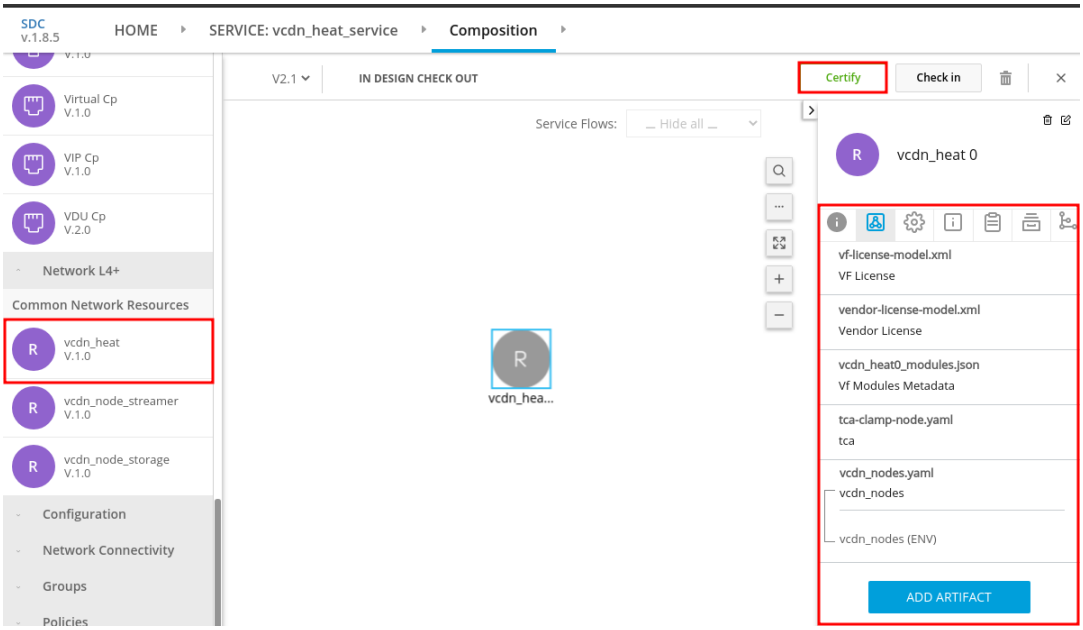


Figure relative to 5.3d)

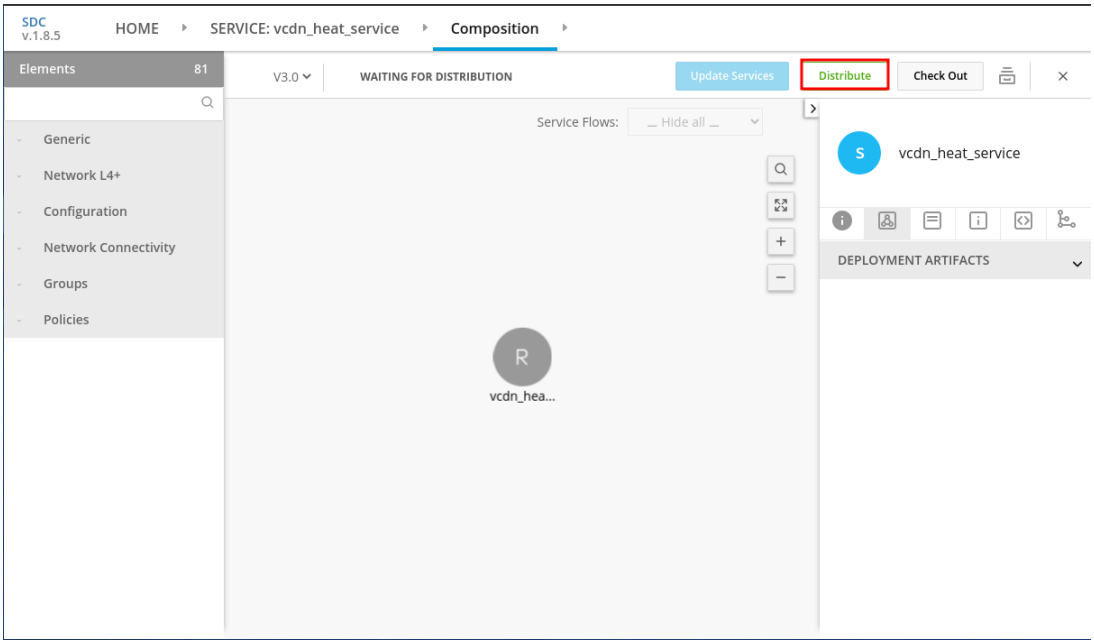


Figure relative to 5.3e)

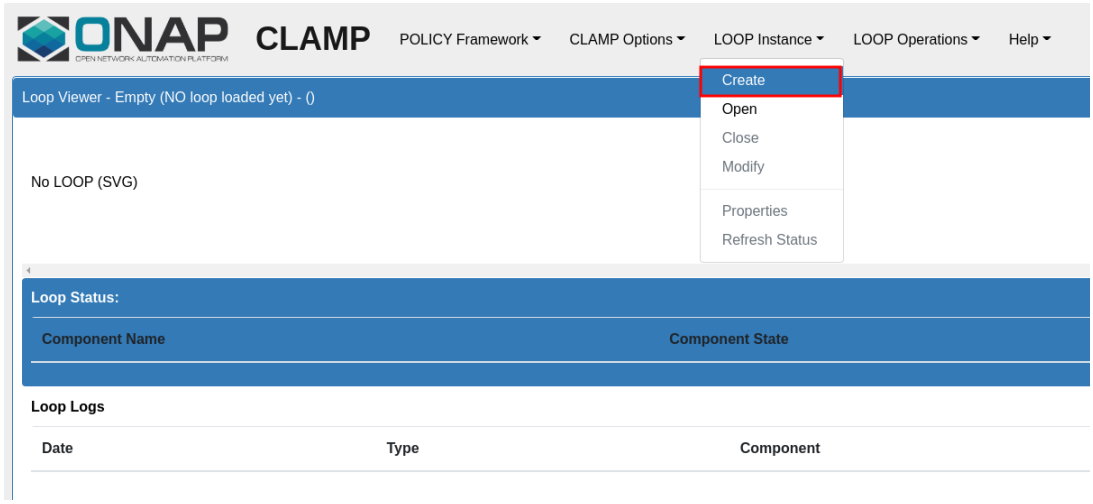


Figure relative to 5.4a)

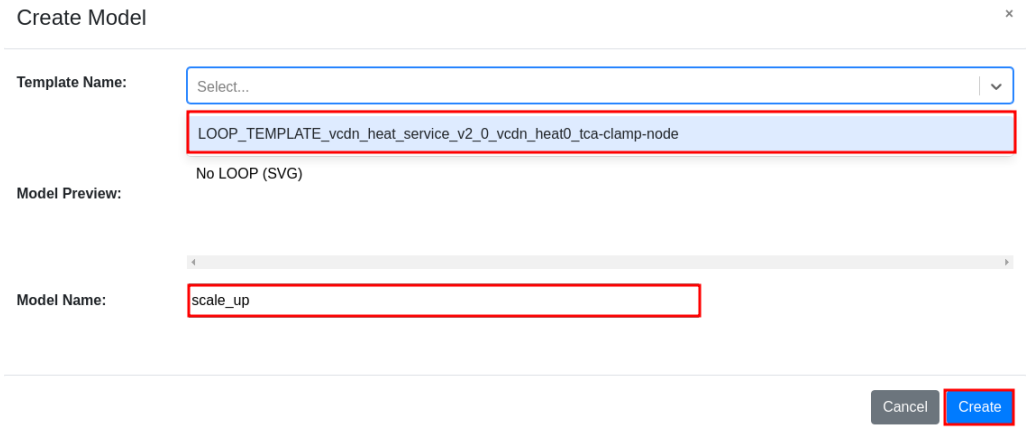


Figure relative to 5.4b)

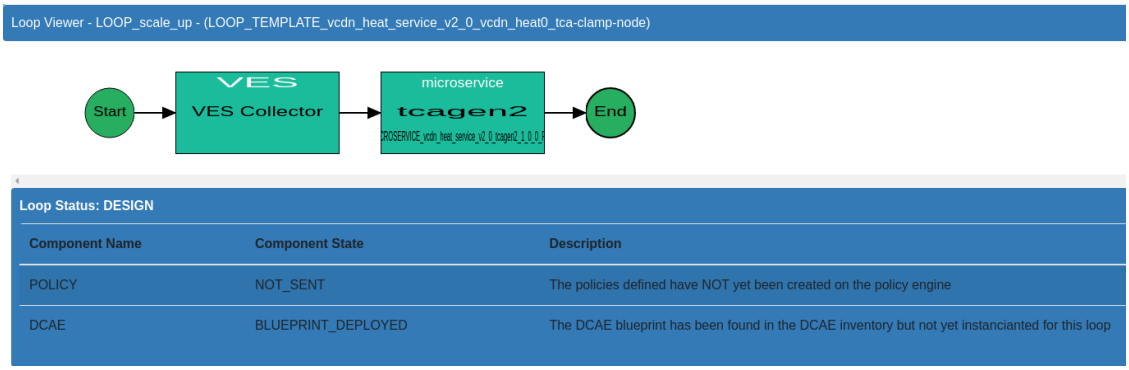


Figure relative to 5.4c)

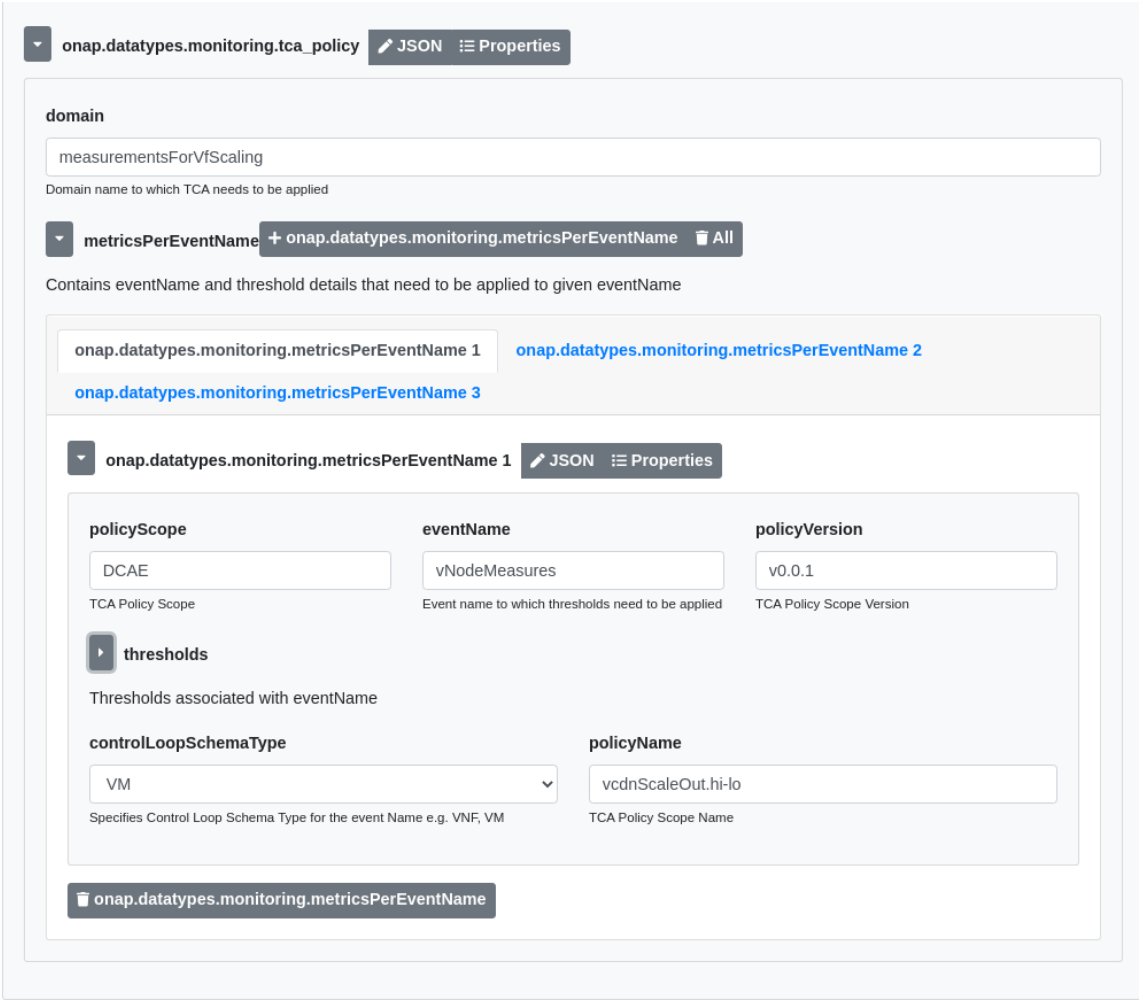


Figure relative to 5.5a)

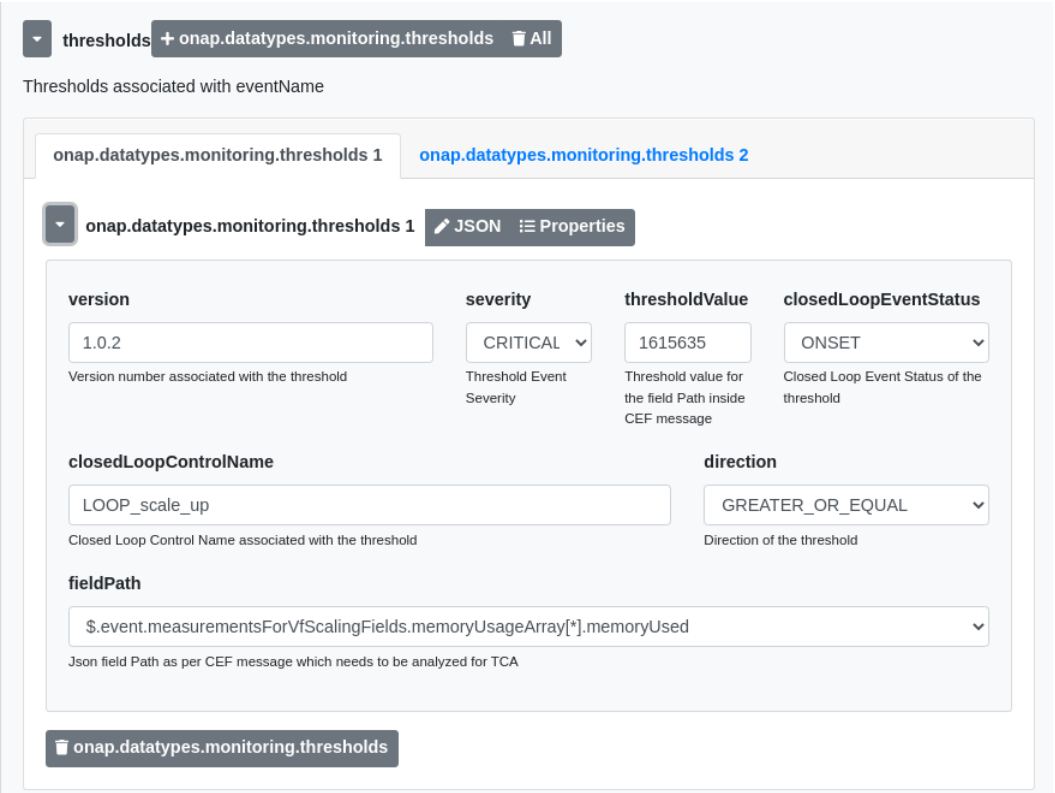


Figure relative to 5.5b)

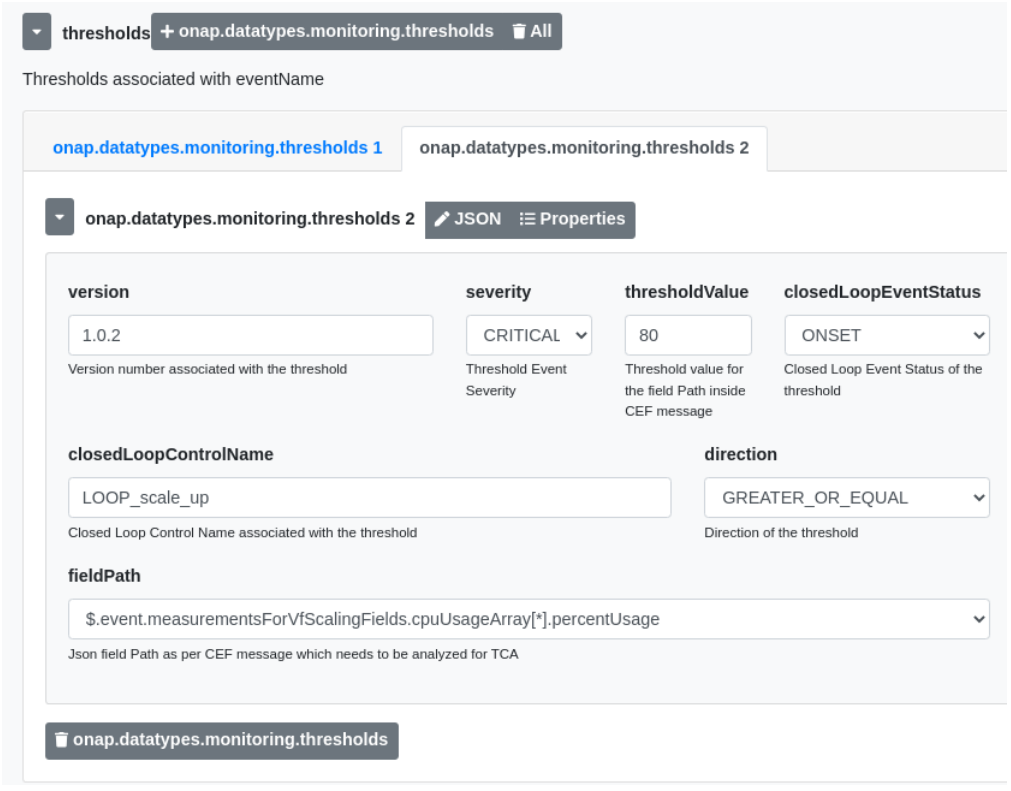


Figure relative to 5.5c)

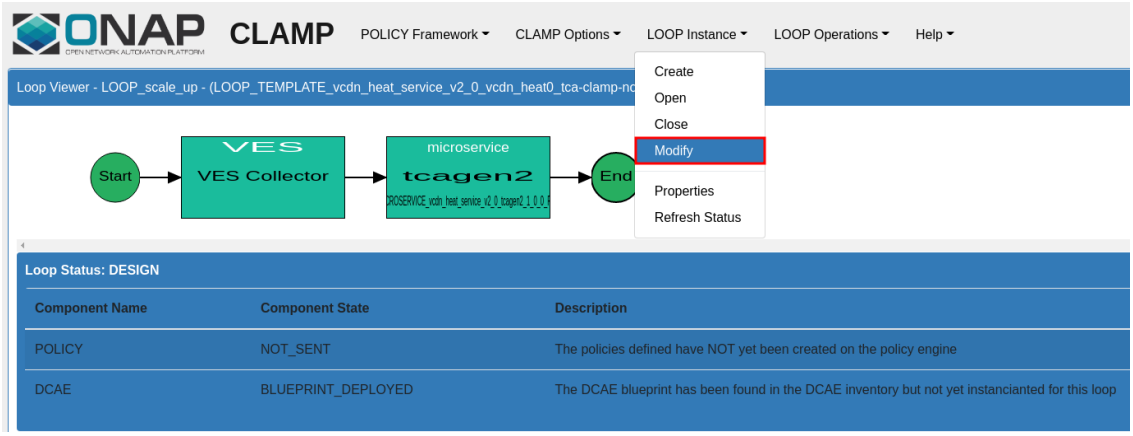


Figure relative to 5.6a)

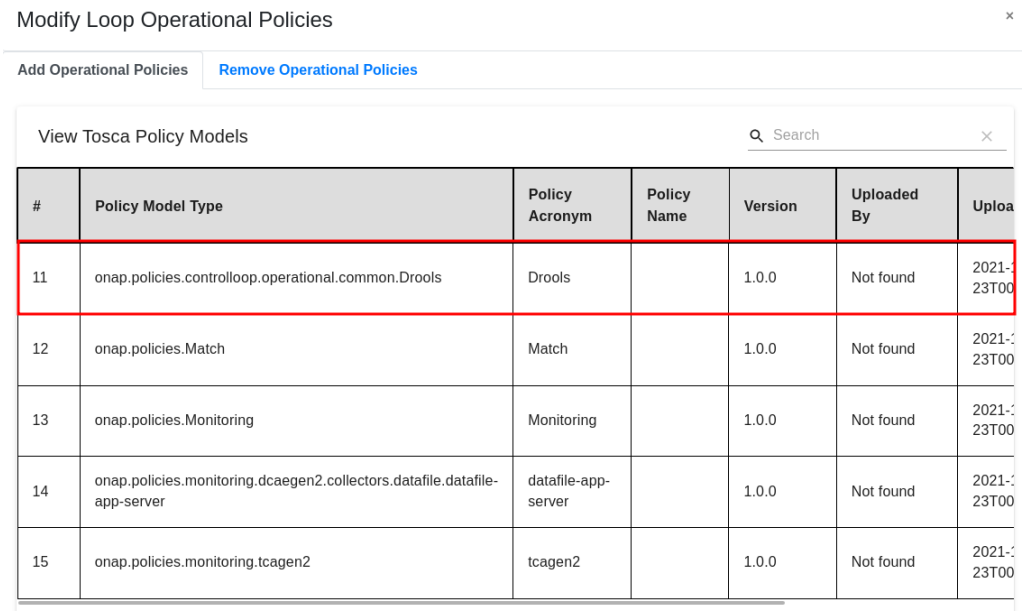


Figure relative to 5.6b)

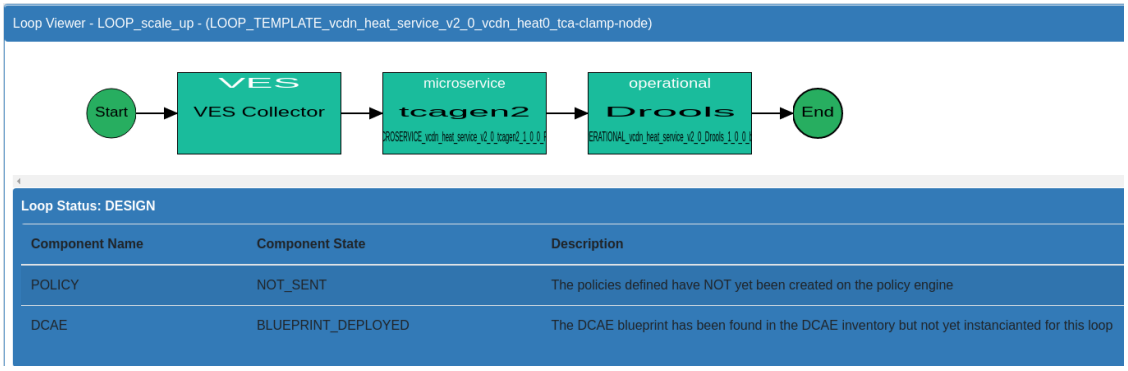


Figure relative to 5.6c)

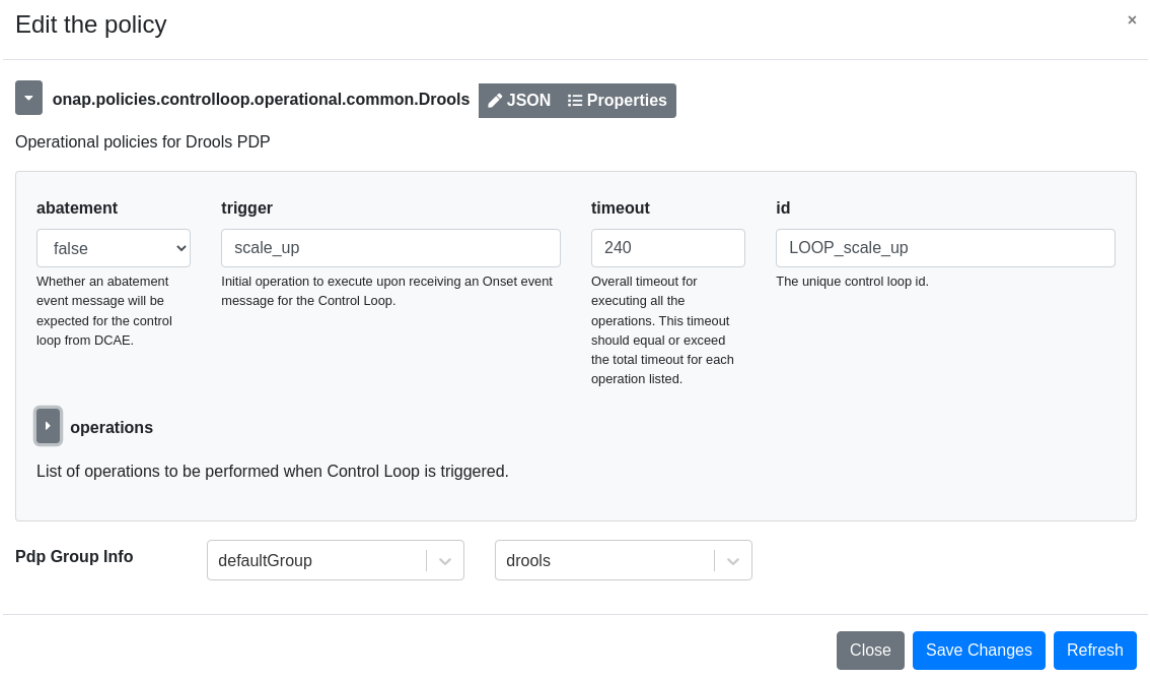


Figure relative to 5.7a)

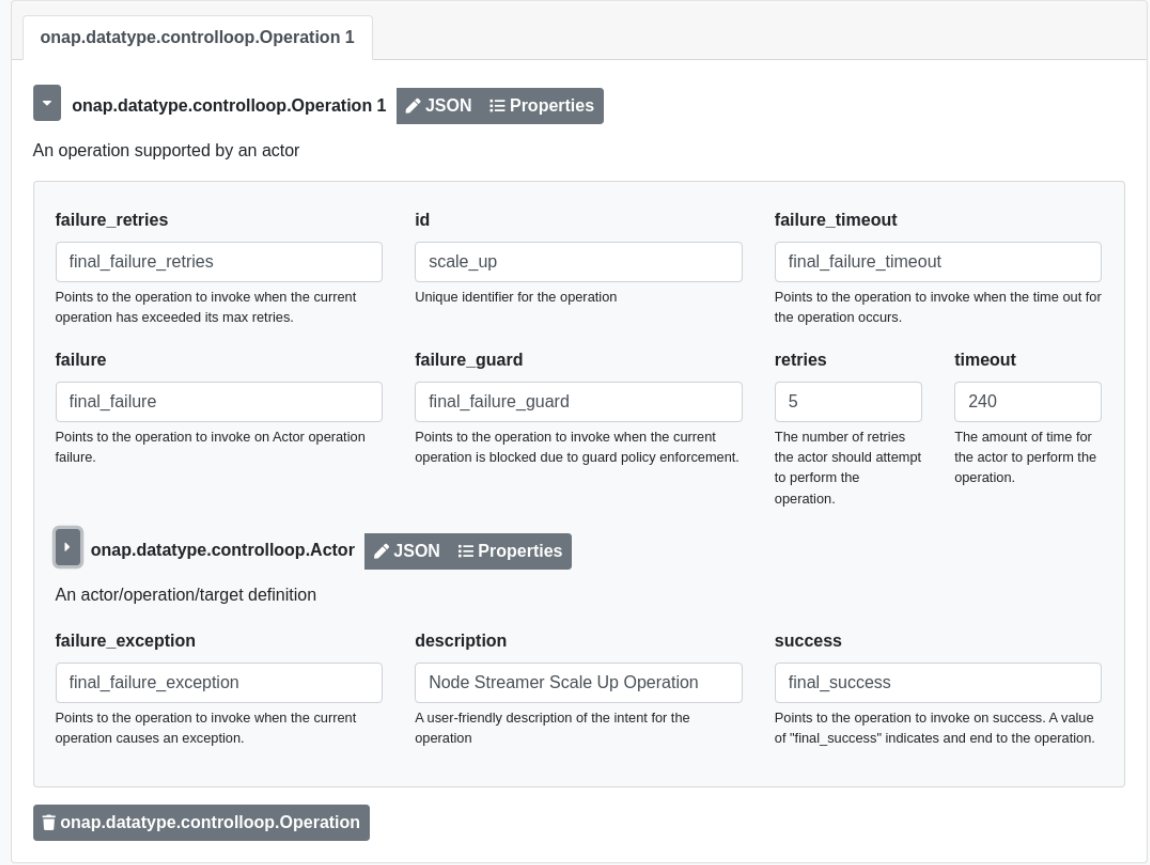


Figure relative to 5.7b)

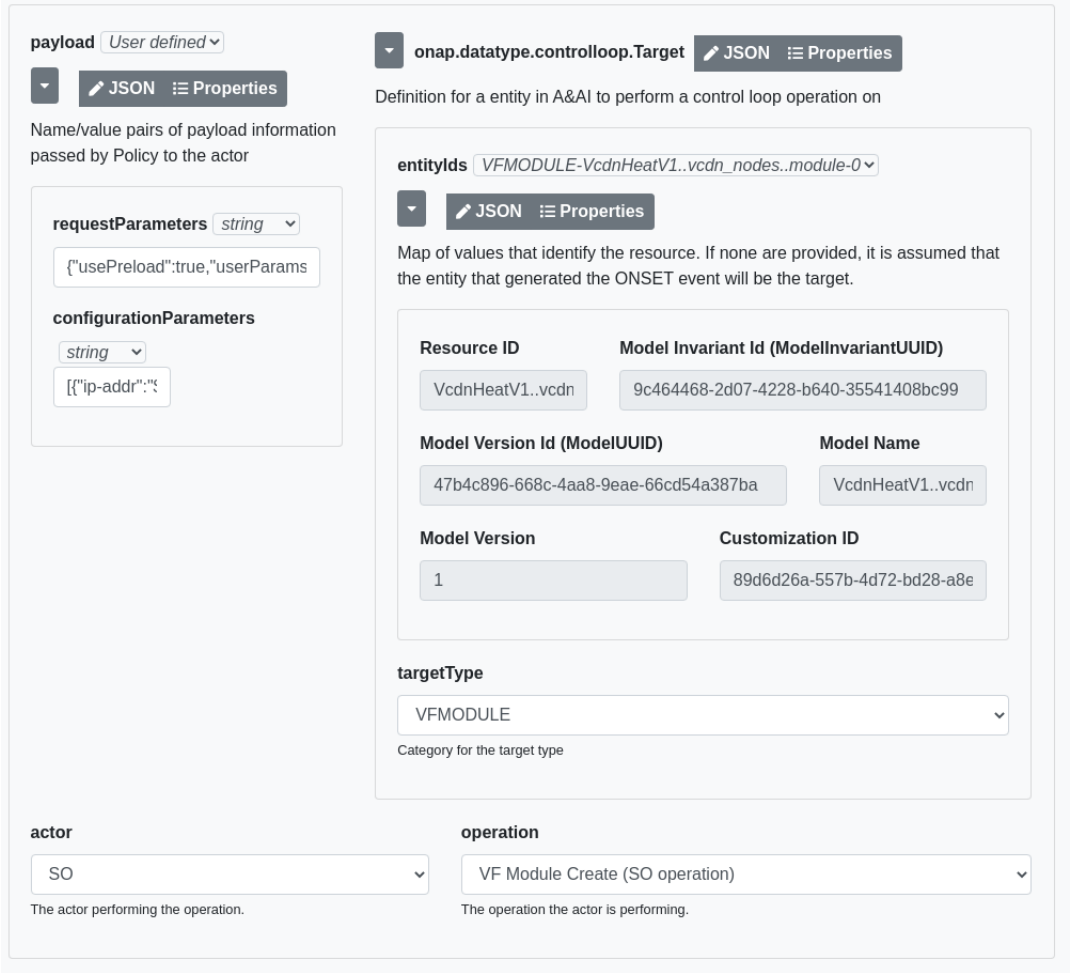


Figure relative to 5.7c)