



**Rafael das Neves  
Simões Direito**

**Integração Contínua no 5GASP  
5GASP Continuous Integration**





Universidade de Aveiro  
2021

**Rafael das Neves  
Simões Direito**

**Integração Contínua no 5GASP  
5GASP Continuous Integration**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor Diogo Nuno Pereira Gomes, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Rui Luís Andrade Aguiar, Professor catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.





Dedico este trabalho a toda a minha família, pelo seu incansável apoio.



**o júri / the jury**

presidente / president

Prof. Doutor José Luís Guimarães Oliveira  
professor catedrático da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Paulo Alexandre Ferreira Simões  
professor associado da Universidade de Coimbra

Prof. Doutor Diogo Nuno Pereira Gomes  
professor auxiliar da Universidade de Aveiro



## **agradecimentos / acknowledgements**

Primeiramente, agradeço à minha família: aos meus pais, à minha irmã, aos meus avós, aos meus tios-avós e à minha madrinha, que ao longo da minha vida sempre moveram montanhas para me proporcionar as condições necessárias para ser bem sucedido, quer no meu percurso académico, quer no meu percurso como pessoa. Um enorme obrigado à minha família. Agradeço também aos meus amigos, por todo o seu incansável apoio e amizade. Foram 5 duros anos até chegar a este momento. 5 anos que certamente teriam sido mais difíceis sem o vosso companheirismo. Não posso, contudo, deixar de agradecer particularmente ao Rafael Teixeira, que me acompanhou desde o início desta jornada e foi um dos meus maiores apoios nos momentos de maior dificuldade. Agradeço, do mesmo modo, ao Eduardo Santos, por toda a sua amizade, disponibilidade e apoio. Deixo também um enorme agradecimento ao meu orientador, Prof. Doutor Diogo Gomes, por me ter facultado a oportunidade de participar ativamente no projeto europeu 5GASP e, por todos os dias, me ajudar a superar barreiras. Obrigado pelo seu apoio e pela confiança que depositou em mim. Aproveito para também agradecer ao Instituto de Telecomunicações - Aveiro, por me fornecer todos os recursos e condições necessárias para a realização deste trabalho (BIL 2021/00045 – 101016448 – 5GASP). Um obrigado particular ao João Pedro Fonseca, meu colega de trabalho, que tão bem me acolheu e me apoiou diariamente desde o início do meu percurso no Instituto de Telecomunicações. A todos vós, o meu mais sincero obrigado.



## Palavras Chave

5G, NFV, NetApp, OSM, Integração Contínua, Automação, Validação, Monitorização.

## Resumo

A ampla adoção do paradigma NFV pelos operadores de rede comprova a importância do mesmo no futuro das redes de telecomunicações. Este paradigma permite que os operadores de rede consigam acelerar o processo de desenvolvimento dos serviços por estes oferecidos, separando o hardware das funcionalidades oferecidas. Contudo, dado que o paradigma NFV apenas foi globalmente adotado recentemente, surgem diversos entraves. Os operadores de rede necessitam de se assegurar da fiabilidade e do correto comportamento das funções de rede virtualizadas, algo que se apresenta como um enorme desafio. Assim, é necessário o desenvolvimento de ferramentas de validação, capazes de validar as funções de rede desenvolvidas num paradigma NFV. O 5GASP é um projeto europeu que pretende diminuir o tempo necessário para implementar uma ideia e a lançar no mercado. Isto é suportado pela criação de uma testbed 5G completamente automatizada e que fornece suporte para ferramentas de Integração Contínua num ambiente seguro e confiável. Esta dissertação, estando alinhada com os objetivos do projeto 5GASP, aborda o desenvolvimento de ferramentas para a validação de NetApps. Desta forma, este documento introduz dois mecanismos para validar NetApps. A primeira ferramenta desenvolvida é responsável por validar estaticamente uma NetApp, antes desta ser implantada na testbed do projeto 5GASP, e é denominada por NetApp Package Validator. Relativamente a esta ferramenta, este documento foca-se no seu módulo de validação de descritores, que valida os descritores de uma NetApp através de testes sintáticos, semânticos e de referência, estando apto para validar NetApps desenvolvidas de acordo com diferentes modelos de informação. A segunda ferramenta disponibiliza um pipeline de validação. Este pipeline valida funcionalmente as NetApps, bem como o seu comportamento, após terem sido implantadas numa testbed 5G. Para além disto, recolhe diversas métricas que ajudam a entender o comportamento das NetApps. É expectável que ambas as ferramentas sejam integradas no ecossistema do projeto 5GASP. Este documento apresenta a definição de requisitos, a arquitetura e a implementação destas ferramentas, apresentando também os resultados da sua implementação.





**Keywords**

5G, NFV, NetApp, OSM, Continuous Integration, Automation, Validation, Monitoring.

**Abstract**

The wide adoption of an NFV-oriented paradigm by network operators proves the importance of NFV in the future of communication networks. This paradigm allows network operators to speed up the development process of their services, decoupling hardware from the functionalities provided by these services. However, since NFV has only been recently globally adopted, several questions and difficulties arose. Network operators need to ensure the reliability and the correct behavior of their Virtualized Network Functions, which poses severe challenges. Thus, the need for developing new validation tools, which are capable of validating network functions that live in an NFV ecosystem. 5GASP is a European project which aims to shorten the idea-to-market process by creating a fully automated and self-service 5G testbed and providing support tools for Continuous Integration in a secure and trusted environment, addressing the DevOps paradigm. Being aligned with 5GASP's goals, this dissertation mainly addresses the development of tools to validate NetApps. To accomplish this, this document introduces two different mechanisms for validating NetApps. The first tool is responsible for statically validate the NetApps before they are deployed in 5GASP's testbeds, being denominated by NetApp Package Validator. Regarding this tool, during this document the focus is its Descriptors Validator Module, which validates the NetApp descriptors through syntactic, semantics, and reference validation and supports NetApps developed according to different Information Models. The second tool comprises an automated validation pipeline. This pipeline validates the functionality and the behavior of the NetApps once they are deployed in a 5G-testbed. Besides, it collects several metrics to enable a better understanding of the NetApp's behavior. Both tools are expected to be integrated with the 5GASP's ecosystem. This document presents the requirements definition, architecture, and implementation of these tools and presents their results and outputs.



# Contents

|                                                                                 |            |
|---------------------------------------------------------------------------------|------------|
| <b>Contents</b>                                                                 | <b>i</b>   |
| <b>List of Figures</b>                                                          | <b>v</b>   |
| <b>List of Tables</b>                                                           | <b>vii</b> |
| <b>Acronyms</b>                                                                 | <b>ix</b>  |
| <b>1 Introduction</b>                                                           | <b>1</b>   |
| 1.1 Motivation . . . . .                                                        | 2          |
| 1.2 Goals . . . . .                                                             | 2          |
| 1.3 Dissertation Outline . . . . .                                              | 2          |
| 1.4 Contributions . . . . .                                                     | 3          |
| <b>2 Background Concepts and Work Proposal</b>                                  | <b>5</b>   |
| 2.1 Evolution of the Networking Paradigms . . . . .                             | 5          |
| 2.2 Network Function Virtualization . . . . .                                   | 6          |
| 2.2.1 Virtual Network Functions, Network Services, and Network Slices . . . . . | 8          |
| 2.2.2 ETSI MANO . . . . .                                                       | 8          |
| 2.3 MANO Platforms . . . . .                                                    | 10         |
| 2.3.1 ONAP . . . . .                                                            | 11         |
| 2.3.2 SONATA and 5GTANGO . . . . .                                              | 12         |
| 2.3.3 OSM . . . . .                                                             | 13         |
| 2.4 DevOps and Continuous Integration . . . . .                                 | 19         |
| 2.4.1 DevOps Methodology . . . . .                                              | 20         |
| 2.4.2 Continuous Integration . . . . .                                          | 22         |
| 2.4.3 Leveraging DevOps and Continuous Integration in a NFV Scenario . . . . .  | 23         |
| 2.4.4 Continuous Integration Tools . . . . .                                    | 23         |
| 2.4.5 Test Automation . . . . .                                                 | 25         |
| 2.4.6 Monitoring Tools . . . . .                                                | 27         |

|          |                                                                  |           |
|----------|------------------------------------------------------------------|-----------|
| 2.5      | Related Work . . . . .                                           | 29        |
| 2.5.1    | 5G-VINNI . . . . .                                               | 29        |
| 2.5.2    | 5GTANGO . . . . .                                                | 31        |
| 2.5.3    | 5G EVE . . . . .                                                 | 32        |
| 2.5.4    | 5GinFIRE . . . . .                                               | 36        |
| 2.5.5    | TMF Specifications . . . . .                                     | 37        |
| 2.6      | Work Proposal . . . . .                                          | 38        |
| 2.7      | Chapter Summary . . . . .                                        | 40        |
| <b>3</b> | <b>Descriptors Validator Service</b>                             | <b>43</b> |
| 3.1      | Problem Statement and Requirements . . . . .                     | 43        |
| 3.2      | Architecture and Specifications . . . . .                        | 45        |
| 3.2.1    | Information Models Parser Module . . . . .                       | 45        |
| 3.2.2    | Descriptors Validator Module . . . . .                           | 46        |
| 3.2.3    | Correction Suggestions Module . . . . .                          | 47        |
| 3.2.4    | REST API and GUI . . . . .                                       | 47        |
| 3.3      | Implementation . . . . .                                         | 49        |
| 3.3.1    | Information Models Parser Module . . . . .                       | 49        |
| 3.3.2    | Descriptors Validator Module . . . . .                           | 53        |
| 3.3.3    | Correction Suggestions Module . . . . .                          | 59        |
| 3.3.4    | REST API and GUI . . . . .                                       | 60        |
| 3.4      | Chapter Summary . . . . .                                        | 62        |
| <b>4</b> | <b>5GASP CI/CD Service</b>                                       | <b>65</b> |
| 4.1      | Problem Statement and Requirements . . . . .                     | 65        |
| 4.2      | Architecture and Specifications . . . . .                        | 67        |
| 4.2.1    | Components . . . . .                                             | 67        |
| 4.2.2    | 5GASP CI/CD Service's interaction with the NODS . . . . .        | 69        |
| 4.2.3    | High-Level Architecture . . . . .                                | 70        |
| 4.2.4    | Selection of the CI tool to implement the CI/CD Agents . . . . . | 71        |
| 4.2.5    | Selection of the TAF . . . . .                                   | 72        |
| 4.2.6    | Tests Execution . . . . .                                        | 72        |
| 4.2.7    | Selection of the Monitoring Mechanisms . . . . .                 | 73        |
| 4.2.8    | Metrics Collection . . . . .                                     | 74        |
| 4.2.9    | Final Architecture and Workflow . . . . .                        | 74        |
| 4.3      | Implementation . . . . .                                         | 77        |
| 4.3.1    | CI/CD Manager . . . . .                                          | 77        |
| 4.3.2    | CI/CD Agents . . . . .                                           | 82        |

|          |                                                                |            |
|----------|----------------------------------------------------------------|------------|
| 4.3.3    | LTR . . . . .                                                  | 84         |
| 4.3.4    | Metrics Collection . . . . .                                   | 88         |
| 4.3.5    | TRVD . . . . .                                                 | 89         |
| 4.3.6    | Testing Descriptors . . . . .                                  | 91         |
| 4.4      | Chapter Summary . . . . .                                      | 95         |
| <b>5</b> | <b>Results and Evaluation</b>                                  | <b>97</b>  |
| 5.1      | Descriptors Validator . . . . .                                | 97         |
| 5.1.1    | Usage Scenarios . . . . .                                      | 97         |
| 5.1.2    | Testing Scenario . . . . .                                     | 98         |
| 5.1.3    | Results . . . . .                                              | 99         |
| 5.2      | 5GASP CI/CD Service . . . . .                                  | 106        |
| 5.2.1    | Usage Scenarios . . . . .                                      | 106        |
| 5.2.2    | Testing Scenario . . . . .                                     | 106        |
| 5.2.3    | Results . . . . .                                              | 107        |
| 5.3      | Chapter Summary . . . . .                                      | 113        |
| <b>6</b> | <b>Conclusions</b>                                             | <b>115</b> |
| 6.1      | Conclusions . . . . .                                          | 115        |
| 6.2      | Future Work . . . . .                                          | 116        |
|          | <b>References</b>                                              | <b>119</b> |
|          | <b>Appendix</b>                                                | <b>125</b> |
|          | Descriptors Validator Service Classification Results . . . . . | 125        |
|          | Descriptors Validator Service Processing Times . . . . .       | 129        |
|          | Correction Suggestions Module Results . . . . .                | 131        |



# List of Figures

|      |                                                                                                           |    |
|------|-----------------------------------------------------------------------------------------------------------|----|
| 2.1  | Transition from the Traditional Networking Paradigm to an NFV-Oriented Paradigm [9].                      | 7  |
| 2.2  | ETSI NFV Architectural Framework [13]. . . . .                                                            | 9  |
| 2.3  | VNFM Scaling-Up a VNF [9]. . . . .                                                                        | 10 |
| 2.4  | ONAP's Honolulu Release - Platform Architecture Diagram [17]. . . . .                                     | 11 |
| 2.5  | "SONATA powered by 5GTANGO"'s architecture. . . . .                                                       | 13 |
| 2.6  | OSM Main Architectural Components [25]. . . . .                                                           | 14 |
| 2.7  | Description of VNFD's Tags. . . . .                                                                       | 16 |
| 2.8  | OSM MON - Internal Architecture. [31] . . . . .                                                           | 17 |
| 2.9  | OSM Release 5 - Architecture. [31] . . . . .                                                              | 17 |
| 2.10 | VNF Proxy Charm Integration with OSM's Workflow. Addapted from [31]. . . . .                              | 18 |
| 2.11 | Phases of the Waterfall Model. . . . .                                                                    | 20 |
| 2.12 | DevOps Stages. . . . .                                                                                    | 21 |
| 2.13 | Relationship between Continuous Integration, Continuous Delivery, and Continuous Deployment [46]. . . . . | 23 |
| 2.14 | Network Slice Instantiation in Openslice [71]. . . . .                                                    | 30 |
| 2.15 | 5GTANGO's V&V Platform [73]. . . . .                                                                      | 31 |
| 2.16 | 5G EVE Platform Architecture [75]. . . . .                                                                | 33 |
| 2.17 | 5G EVE Testing and Validation Approach [76]. . . . .                                                      | 33 |
| 2.18 | 5G EVE Testing and Validation Framework Architecture [76]. . . . .                                        | 34 |
| 2.19 | 5G EVE Test Case Blueprinte. Adapted from [78]. . . . .                                                   | 35 |
| 2.20 | 5G EVE Experiment Blueprint Example. Adapted from [79]. . . . .                                           | 35 |
| 2.21 | Automated Validation of VNFs in 5GinFIRE [82]. . . . .                                                    | 36 |
| 2.22 | Class Diagram for the Service Specification Resource Class, defined by TMF633 [84]. . .                   | 38 |
|      |                                                                                                           |    |
| 3.1  | Information Models Parser Module's Architecture. . . . .                                                  | 46 |
| 3.2  | Descriptors Validator Module's Architecture. . . . .                                                      | 46 |
| 3.3  | Correction Suggestions Module's Architecture. . . . .                                                     | 47 |
| 3.4  | Descriptors Validator Service Complete Architecture. . . . .                                              | 48 |
| 3.5  | Semantic Validation Flow. . . . .                                                                         | 55 |

|      |                                                                                                                 |     |
|------|-----------------------------------------------------------------------------------------------------------------|-----|
| 3.6  | Reference Validation Flow. . . . .                                                                              | 56  |
| 3.7  | Descriptors Validator Database Schema. . . . .                                                                  | 56  |
| 3.8  | Complete Descriptors Validation Flow. . . . .                                                                   | 58  |
| 3.9  | REST API's Endpoints. . . . .                                                                                   | 60  |
| 3.10 | WebUI Before Validating a Descriptor. . . . .                                                                   | 61  |
| 3.11 | WebUI After Validating a Descriptor. . . . .                                                                    | 62  |
|      |                                                                                                                 |     |
| 4.1  | 5GASP Approach on DevOps Experimentation and Certification Lifecycle [87]. . . . .                              | 66  |
| 4.2  | 5GASP's Communication Interfaces [88]. . . . .                                                                  | 69  |
| 4.3  | E2 Interface Utilization [89] . . . . .                                                                         | 70  |
| 4.4  | 5GASP CI/CD Service's High-Level Architecture. . . . .                                                          | 71  |
| 4.5  | CI/CD Service's Architecture. . . . .                                                                           | 75  |
| 4.6  | CI/CD Service's Interaction Diagram. . . . .                                                                    | 76  |
| 4.7  | 5GASP CI/CD Manager API Endpoints. . . . .                                                                      | 78  |
| 4.8  | Workflow of a New Test Submission to the CI/CD Manager. . . . .                                                 | 80  |
| 4.9  | CI/CD Manager's Database Schema. . . . .                                                                        | 82  |
| 4.10 | CI/CD Agent's Pipeline Stages. . . . .                                                                          | 83  |
| 4.11 | Chronograf's Dashboard Example. . . . .                                                                         | 89  |
| 4.12 | TRVD - Test Base Information. . . . .                                                                           | 89  |
| 4.13 | TRVD - Test Stages. . . . .                                                                                     | 90  |
| 4.14 | TRVD - Tests Performed and Collected Metrics. . . . .                                                           | 90  |
| 4.15 | TRVD - Test Outputs and Results. . . . .                                                                        | 91  |
|      |                                                                                                                 |     |
| 5.1  | Relation Between the Descriptor's File Size and its Overall Processing Time - Scatter Plot. 101                 |     |
| 5.2  | Relation Between the Descriptor's File Size and its Validation Time - Scatter Plot. . . . .                     | 102 |
| 5.3  | Relation Between the Descriptor's File Size and its Validation Time - Box Plot. . . . .                         | 102 |
| 5.4  | Descriptor's File Size Distribution. . . . .                                                                    | 103 |
| 5.5  | Relation Between the Descriptor's Error Type and the Time Needed to Generate Correction<br>Suggestions. . . . . | 105 |
| 5.6  | TRVD's Web Interface Portraiting a Successful Validation Process. . . . .                                       | 110 |
| 5.7  | <i>report.html</i> Robot File for the Successful Execution of the Open Ports Test. . . . .                      | 110 |
| 5.8  | TRVD's Web Interface Portraiting an Unsuccessful Validation Process. . . . .                                    | 110 |
| 5.9  | <i>report.html</i> Robot File for the Unsuccessful Execution of the Open Ports Test. . . . .                    | 111 |
| 5.10 | Portion of the Console Log of an Unsuccessful Validation Process Execution. . . . .                             | 111 |
| 5.11 | Metrics Repository Dashboard. . . . .                                                                           | 112 |
| 5.12 | Overall Validation Process Execution Time. . . . .                                                              | 112 |
| 5.13 | Execution Time for Each Individual Test. . . . .                                                                | 113 |



# List of Tables

|     |                                                                                                                |     |
|-----|----------------------------------------------------------------------------------------------------------------|-----|
| 2.1 | Overview of the Most Used CI Tools. . . . .                                                                    | 25  |
| 4.1 | Possible Test Status. . . . .                                                                                  | 81  |
| 5.1 | Subset of the Descriptors Validator Service Classification Results. . . . .                                    | 99  |
| 5.2 | Sample of the Descriptors Validator Service Processing Times. . . . .                                          | 101 |
| 5.3 | Relation Between the Descriptor's File Size and its Median Validation Time. . . . .                            | 103 |
| 5.4 | Sample of the Correction Suggestions Module Results. . . . .                                                   | 104 |
| 5.5 | Relation Between the Descriptor's Error Type and its Median Correction Suggestions<br>Generation Time. . . . . | 105 |
| 5.6 | Median of Each Test Execution Times. . . . .                                                                   | 113 |
| 1   | Descriptors Validator Service Classification Results. . . . .                                                  | 128 |
| 2   | Descriptors Validator Service Processing Times. . . . .                                                        | 131 |
| 3   | Correction Suggestions Module Results. . . . .                                                                 | 134 |



# Acronyms

|               |                                                 |              |                                                                 |
|---------------|-------------------------------------------------|--------------|-----------------------------------------------------------------|
| <b>5G</b>     | 5th Generation Mobile Network                   | <b>MR</b>    | Metrics Repository                                              |
| <b>5G-PPP</b> | 5G Infrastructure Public Private Partnership    | <b>NBI</b>   | Northbound Interface                                            |
| <b>API</b>    | Application Programming Interface               | <b>NEST</b>  | Network Slice Template                                          |
| <b>BSS</b>    | Business Support System                         | <b>NF</b>    | Network Function                                                |
| <b>CD</b>     | Continuous Delivery                             | <b>NFV</b>   | Network Function Virtualization                                 |
| <b>CD</b>     | Continuous Deployment                           | <b>NFVI</b>  | Network Function Virtualization Infrastructure                  |
| <b>CFM</b>    | Continuous Feedback and Monitoring              | <b>NFVO</b>  | Network Function Virtualization Orchestrator                    |
| <b>CFS</b>    | Customer-Facing Services                        | <b>NODS</b>  | NetApp Onboarding and Deployment Service                        |
| <b>CI</b>     | Continuous Integration                          | <b>NS</b>    | Network Service                                                 |
| <b>CI</b>     | Continuous Integration                          | <b>NSaaS</b> | Network Slice-as-a-Service                                      |
| <b>CLI</b>    | Command-Line Interface                          | <b>NSD</b>   | Network Service Descriptor                                      |
| <b>COTS</b>   | Commercial Off-The-Shelf                        | <b>ONAP</b>  | Open Network Automation Platform                                |
| <b>CP</b>     | Continuous Planning                             | <b>OSM</b>   | Open Source MANO                                                |
| <b>CPU</b>    | Central Processing Unit                         | <b>OSS</b>   | Operations Support System                                       |
| <b>CRUD</b>   | Create, Read, Update and Delete                 | <b>PD</b>    | Performance Diagnosis                                           |
| <b>CSS</b>    | Cascading Style Sheets                          | <b>POL</b>   | Policy Manager                                                  |
| <b>CT</b>     | Continuous Testing                              | <b>QA</b>    | Quality Assurance                                               |
| <b>DNS</b>    | Domain Name System                              | <b>RAM</b>   | Random Access Memory                                            |
| <b>E2E</b>    | end-to-end                                      | <b>RAV</b>   | Results Analysis and Validation                                 |
| <b>EEM</b>    | Experiment Execution Manager                    | <b>REST</b>  | Representational State Transfer                                 |
| <b>ETSI</b>   | European Telecommunications Standards Institute | <b>RFS</b>   | Resource-Facing Services                                        |
| <b>EU</b>     | European Union                                  | <b>RO</b>    | Resource Orchestrator                                           |
| <b>FTP</b>    | File Transfer Protocol                          | <b>SCM</b>   | Source Code Management                                          |
| <b>GUI</b>    | Graphical User Interface                        | <b>SDK</b>   | Software Development Kit                                        |
| <b>HTML</b>   | HyperText Markup Language                       | <b>SDLC</b>  | Software Development Lifecycle                                  |
| <b>HTTP</b>   | Hypertext Transfer Protocol                     | <b>SDN</b>   | Software Defined Network                                        |
| <b>IDE</b>    | Integrated Development Environment              | <b>SLA</b>   | Service Layer Agreement                                         |
| <b>IM</b>     | Information Model                               | <b>SME</b>   | Small and Medium-Sized Enterprise                               |
| <b>IoT</b>    | Internet of Things                              | <b>SOAP</b>  | Simple Object Access Protocol                                   |
| <b>ISG</b>    | Industry Specification Group                    | <b>SSH</b>   | Secure Shell                                                    |
| <b>IT</b>     | Information Technology                          | <b>SSH</b>   | Secure Shell                                                    |
| <b>JS</b>     | JavaScript                                      | <b>SUT</b>   | System Under Test                                               |
| <b>JSON</b>   | JavaScript Object Notation                      | <b>TAF</b>   | Test Automation Framework                                       |
| <b>KPI</b>    | Key Performance Indicator                       | <b>TCP</b>   | Transmission Control Protocol                                   |
| <b>LCM</b>    | Lifecycle Manager                               | <b>TEE</b>   | Test Execution Engine                                           |
| <b>LTR</b>    | Local Test Repository                           | <b>TMF</b>   | Tele Management Forum                                           |
| <b>MANO</b>   | Management and Orchestration                    | <b>TOSCA</b> | Topology and Orchestration Specification for Cloud Applications |
| <b>MON</b>    | Monitoring Module                               | <b>TRVD</b>  | Test Results Visualization Dashboard                            |

|                |                                     |              |                                         |
|----------------|-------------------------------------|--------------|-----------------------------------------|
| <b>TSDB</b>    | Time Series Database                | <b>VM</b>    | Virtual Machine                         |
| <b>TSP</b>     | Telecommunication Service Provider  | <b>VNF</b>   | Virtualised Network Function            |
| <b>UI</b>      | Lightweight User Interface          | <b>VNFD</b>  | Virtualised Network Function Descriptor |
| <b>UI</b>      | User Interface                      | <b>VNFM</b>  | Virtual Network Function Manager        |
| <b>URL</b>     | Uniform Resource Locator            | <b>VoLTE</b> | Voice over Long-Term Evolution          |
| <b>V&amp;V</b> | Verification & Validation           | <b>WP</b>    | Work Package                            |
| <b>VCA</b>     | VNF Configuration and Abstraction   | <b>XML</b>   | Extensible Markup Language              |
| <b>vCPE</b>    | virtual Customer Premises Equipment | <b>YAML</b>  | YAML Ain't Markup Language              |
| <b>VIM</b>     | Virtualised Infrastructure Manager  | <b>YANG</b>  | Yet Another Next Generation             |
| <b>VL</b>      | Virtual Link                        |              |                                         |

# Introduction

Traditional Network Systems have always been hardware-based systems. These systems rapidly reach the end of their lifecycle, forcing the Telecommunication Service Providers (TSPs) to acquire new equipment regularly. Besides, these systems are designed to fulfill a specific goal, and it is difficult to adapt them to meet other goals and functionalities.

Due to this, the TSPs are facing huge capital expenditures (CAPEX) and operating expenditures (OPEX) [1]. There are also other problems arising from these architectures: (i) lack of flexibility and of scalability, and (ii) an increase of the time-to-market, resulting in (iii) a loss of revenue[1].

Besides, the rise of Internet of Things (IoT) and streaming services, for instance, highly contributes to an increase in network traffic, which poses an even more significant challenge for Telecommunication Providers.

The necessity of higher bandwidths and lower latencies profoundly impacts the adoption of 5th Generation Mobile Network (5G) networks. These networks have high bandwidth, reduced latency, low power consumption, and are cost-effective, thus enabling better and faster network services [2].

This scenario can be addressed with Network Function Virtualization (NFV), which leverages virtualization technologies and allows the consolidation of several network equipment onto a single server located on-premises or on an external data center. NFV provides the decoupling of the Network Functions (NFs) from the hardware they run on, enabling them to be deployed on Commercial Off-The-Shelf (COTS) servers through Virtualised Network Functions (VNFs), which can easily be adjusted to the client's demands [3][4].

This approach allows the reduction of the equipment cost and of its maintenance. This is also a more agile process, which results in time savings and increased scalability and flexibility, allowing the TSPs to produce new systems and products faster [5].

However, the migration from traditional networks to NFV oriented networks is not a straightforward process, raising many questions concerning software reliability [2].

In fact, one of the biggest challenges of NFV is the validation of softwarized NFs. Thus,

in recent years, the existence of a continuous effort to address this problem. Projects like 5GTANGO<sup>1</sup>, 5G-EVE<sup>2</sup>, and 5GinFIRE<sup>3</sup> tackled the verification and validation of VNFs and Network Services (NSs), although not completely solving the issue at hand. It is still possible to make significant improvements in the validation of NFs, and this is where this dissertation focuses.

## 1.1 MOTIVATION

5GASP<sup>4</sup> - 5G Application & Services Experimentation and Certification Platform - is a Horizon 2020 funded project that aims to shorten the idea-to-market process by creating a fully automated and self-service testbed. This project is heavily focused on providing support tools for Continuous Integration and Continuous Deployment of VNFs in a secure and trusted environment, addressing the DevOps paradigm [6].

5GASP aims to validate both VNFs and the underlying infrastructure on which they are deployed. To achieve this, 5GASP has as its main goal the creation of an automatic validation pipeline that validates the VNFs against some developer-defined tests and collects metrics on their performance [6].

This project is expected to extend the work developed under the 5GTANGO, 5G-EVE, and 5GinFIRE projects, solving some of the issues they present and paving the way for a fully automated testing and validation process.

## 1.2 GOALS

This dissertation aims to create tools and mechanisms to allow a fully automated verification and validation of VNFs. These tools should provide a simple and efficient way for NetApp developers to test them and receive feedback on their behavior and performance.

To do so, the development of an automated Continuous Integration (CI) pipeline is of utmost importance. This pipeline has to be highly modular and technology independent in order to allow the 5G community to extend it in the future.

All this work falls under the scope of 5GASP's Work Package (WP) 5 and will be integrated with its infrastructure and services.

## 1.3 DISSERTATION OUTLINE

To elucidate the reader on the several topics addressed in this document, Chapter 2 presents essential background concepts and the work related to one developed during this dissertation. Then, based on the background concepts presented in Chapter 2 and this dissertation's goals, presented in Section 1.2, a work proposal is formed in Section 2.6, describing to the reader the two outputs of this document.

---

<sup>1</sup><https://www.5gtango.eu/>

<sup>2</sup><https://www.5g-eve.eu/>

<sup>3</sup><https://www.5ginfire.eu/>

<sup>4</sup><https://www.5gasp.eu/>

The first output - the Descriptors Validator Service - is addressed in Chapter 3. There, the requirements definition is presented and mapped to the specifications and architecture of this tool. Its implementation is also addressed.

Chapter 4 addresses the second output of this document - the 5GASP CI/CD Service. This chapter follows the same structure as Chapter 3.

Having presented the architecture and implementation of the addressed tools, Chapter 5 introduces and evaluates the results obtained.

In Chapter 6, a work conclusion is addressed. This chapter also presents a future work proposal.

Finally, the references addressed throughout this document and the appendix are presented.

#### 1.4 CONTRIBUTIONS

This dissertation resulted not only in this document but also in several contributions to the 5GASP project and to the overall NFV community. These contributions are listed below:

- Paper published in the 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom). This paper is entitled *An automated CI/CD process for testing and deployment of Network Applications over 5G infrastructure*;
- I was the responsible and main editor of 5GASP's Deliverable 5.1 - Initial Report on Test Plan Creation and Testing Methodologies;
- Contributions to 5GASP's Deliverable 3.1 - Experimentation Services, Middleware and Multi-Domain facilities Continuous Integration;
- Contributions to 5GASP's Deliverable 2.1 - Architecture, Model Entities Specification and Design;





# Background Concepts and Work Proposal

This chapter introduces all the concepts needed to understand the work developed during this dissertation. Aiming to create an automated pipeline for testing and validating VNFs and NSs, it is important to fully understand these concepts and why they are significant in NFV, a new network paradigm. Thus, this chapter starts by addressing the migration from the traditional network paradigm to NFV.

One of the most relevant components of NFV is the Management and Orchestration (MANO) Framework, a management and orchestration platform for NFV. This platform is heavily addressed in this chapter, and its several implementations are presented, such as Open Network Automation Platform (ONAP), SONATA, and Open Source MANO (OSM). Since the 5GASP project relies on OSM, an extended and detailed description of this platform is provided.

DevOps and CI concepts are also heavily addressed since they are the key enablers of the automated validation pipeline this dissertation aims to create.

During this chapter, several related projects are described in order to understand the work that has already been developed, its problems, and how to extend it.

Lastly, a work proposal is formulated, based on all the background concepts presented.

## 2.1 EVOLUTION OF THE NETWORKING PARADIGMS

Traditional Network Systems have always been hardware-based. These rapidly reach the end of their lifecycle, causing TSPs to acquire new equipment regularly. Besides this, many other problems arose:

- Whenever a new network service had to be deployed, another hardware equipment was needed, so as the space for deploying it [7];
- Several components of these systems had to be chained in a predefined order, according to the network topology and the localization of the service components [8];

- The systems were designed to fulfill a specific goal, being extremely hard to adapt them to fulfill other goals and functions;
- Hardware-based appliances' lifecycles were becoming shorter, resulting in considerable costs to keep the network systems running [3];
- Most hardware equipment only had vendor-specific interfaces, which diffculted its integration with other equipment.

The described problems lead to low flexibility and scalability, which resulted in a higher effort to keep the network systems online. TSPs often had to purchase new physical equipment to handle new service requirements [8]. Besides, the hardware restrictions resulted in a higher time-to-market and more difficulties in the network systems' management.

All this imposed higher CAPEX (cost of buying new hardware) and OPEX (cost of maintenance and operation) investments that were not followed by increased revenue streams.

TSPs then realized they had to find ways to (i) build more dynamic networks, (ii) reduce their time-to-market and capital expenses, and (iii) enable a simpler management of these systems.

NFV was proposed to address these challenges, leveraging virtualization technology to offer new networking services. NFVs would allow the TSPs to create new networking services that are completely decoupled from the hardware they run on, solving many of the problems mentioned before [8].

Section 2.2 introduces NFV, its underlying technologies, and the advantages offered to the TSPs that choose to follow this new networking paradigm.

## 2.2 NETWORK FUNCTION VIRTUALIZATION

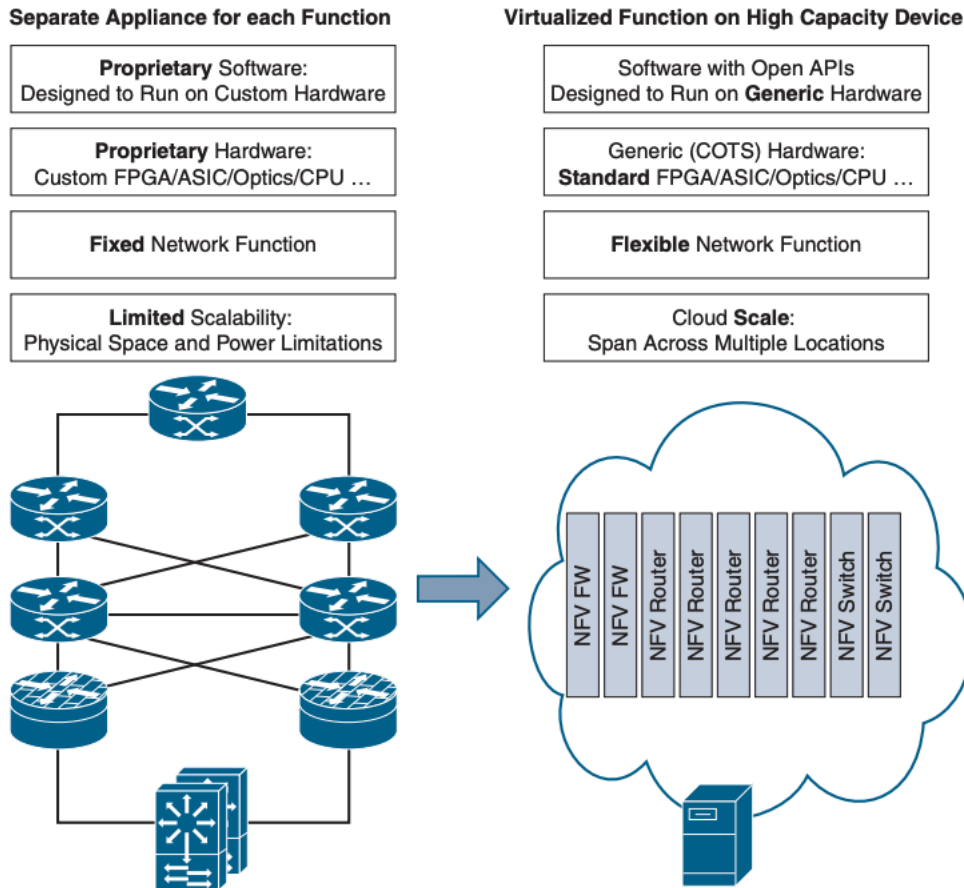
NFV allows the decoupling of the Network Functions from the physical equipment itself. With this level of virtualization it is possible to run several network equipment, such as switches, routers, Domain Name System (DNS) servers, firewalls, and so many more, in COTS [8]. This enables the NFs to be deployed at different locations without installing new hardware, which allows the TSPs to target most of the necessities of the vertical markets without huge added costs.

Besides, NFV allows faster service upgrades, dynamic scalability, better reliability, etc [1]. According to [3] and [4], NFV has the following advantages:

- Use COTS servers to deploy network systems through hyper virtualization;
- Possibility of running several environments in the same infrastructure. The operators can deploy the production and testing scenarios in the same physical infrastructure;
- Decrease of the time-to-market, minimizing the lifecycle of creating a network system;
- Geographical targetted system deployment based on the needs of the operators. Services can be deployed in remote datacenters without the development team being physically present since it will be able to control the infrastructure remotely;
- Automated optimization of the network, based on traffic patterns;
- Reduced energy consumption due to virtualization techniques;

- Improved flexibility and scalability of the networking systems, since they are deployed using software and not hardware;
- Standardized and often open interfaces between several network components;

Given these advantages, the transition from the traditional networking paradigm to an NFV oriented paradigm comes naturally. Fig. 2.1 depicts this transition.



**Figure 2.1:** Transition from the Traditional Networking Paradigm to an NFV-Oriented Paradigm [9].

To summarize, NFV differs from the traditional network paradigm in the following characteristics: (i) decouples hardware and software, making these components independent from each other, and (ii) boosts flexibility and improves the scalability of the networking systems, since the Network Functions are virtual, so their deployment and upgrading is more straightforward when compared to the traditional NFs deployment [1] [4].

In Europe, the entity responsible for the standardization of NFV is the European Telecommunications Standards Institute (ETSI). In 2012 this entity created an Industry Specification Group (ISG) responsible for the research and guidelines on NFV [4].

This group is composed of seven telecommunication operators, and, until this date, produced over one hundred publications that cover everything from pre-standardization studies to detailed specifications for NFV [10]. The last release from ETSI is Release 4, which addresses the work developed in 2019 and 2020. This document is mainly focused on:

- NFV evolution and support for lightweight virtualization, such as OS containers;
- Enhancement of the NFV automation and capabilities;
- Optimization of the NFV-MANO framework.

This release also presents new features, such as [10]:

- NFV-MANO automation
- NFV enhancements for 5G;
- Multy domain enhancements for the NFV-MANO;
- Policy model;
- Continuous VNF integration.

### 2.2.1 Virtual Network Functions, Network Services, and Network Slices

A VNF is the virtual implementation of a Network Function, such as a router, a DNS server, etc. VNFs use Virtual Machines (VMs), provided by a Network Function Virtualization Infrastructure (NFVI), to deploy a softwarized Network Function, replacing a specific hardware component that would be responsible for performing such function. One great advantage of VNFs is that they are not tieded to any specific hardware, being deployed over COTS servers [9], and working seamlessly on different environments.

According to [9], in regards to VNFs, it must be ensured the following:

- The VNF must be independent of the hardware where it is deployed;
- The existence of a lifecycle management service;
- VNFs interconnection.

When more than one VNFs are connected, a NS is created. NSs are composed of multiple VNFs and the connection between them, being defined by their functional and behavioral specifications [11].

Regarding Network Slicing, this concept refers to the partitioning of physical networks into several virtual networks, which can be optimized for specific scenarios and applications and distributed over multiple domains. All this leads to network efficiency gains [12].

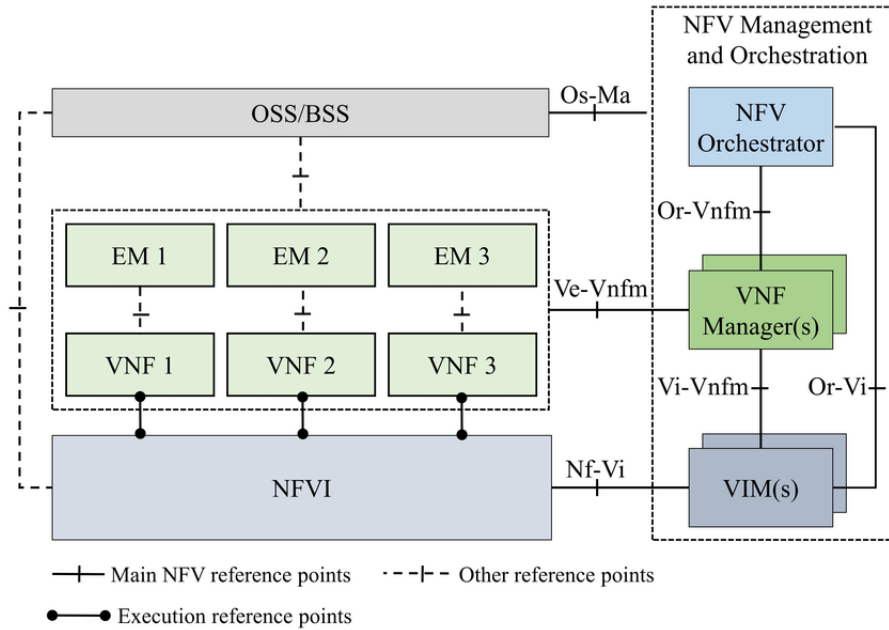
### 2.2.2 ETSI MANO

MANO is one of the most relevant elements of the ETSI NFV architecture, being the architectural framework responsible for the orchestration and lifecycle management of the resources involved in NFV.

MANO, all by itself, is not capable of delivering all the NFV benefits [4]. Hence, the need to integrate MANO with other management entities, using the interfaces provided by these services and providing its own external interfaces so the other entities can communicate with it.

This framework consists of three functional blocks: (i) the Network Function Virtualization Orchestrator (NFVO), (ii) the VNF Manager, and (iii) the Virtualised Infrastructure Manager (VIM), and four data repositories: (i) the NS Catalog, (ii) the VNF Catalog, (iii) the NFV Instances Repository, and the (iv) NFVI Resources Repository [1] [4].

Figure 2.2 depicts the ETSI NFV architectural framework, which includes MANO.



**Figure 2.2:** ETSI NFV Architectural Framework [13].

The VNF Catalog stores the models that describe the deployment and functional features of the VNFs available, while the NS Catalog stores models that detail how to deploy the services and their functions and how to establish communication between them. Regarding the NFVI Repository, it contains information about the availability of the NFVI resources. Lastly, the NFV Instances Repository collects all the data of the VNF and NS instances [1] [4].

A brief description of each of the three main functional blocks of the MANO framework is now presented.

### ETSI NFVO

The NFVO functional block has two crucial roles: (i) resource orchestration, being in charge of the orchestration of infrastructure resources across multiple VIMs, and (ii) network service orchestration, managing the lifecycle of the NSs, and VNFs [11] [14]. To do so, the NFVO interacts with all the repositories mentioned in before.

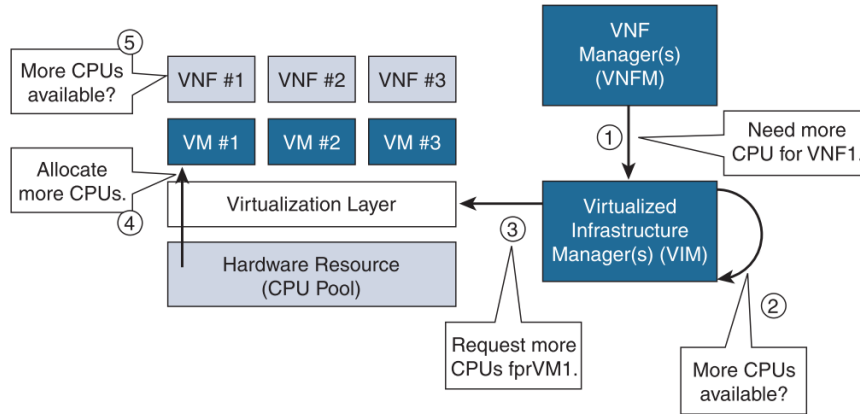
### ETSI VNFM

The VNF Manager is responsible for the lifecycle management of the VNFs. Each VNF must be managed by a Virtual Network Function Manager (VNFM), which, itself, can manage multiple VNFs. According to [14], the main functions of the VNFM are:

- VNF instantiation and termination;
- VNF instance software update;
- VNF instance modification;
- VNF instance scaling;

- VNF instance healing;
- VNF lifecycle management notification;
- Management of the VNFs' integrity.

To perform all these functions, the VNFM continuously communicates with the NFVO and the VIM. Fig. 2.3 presents a scaling-up operation over a VNF managed by a VNFM.



**Figure 2.3:** VNFM Scaling-Up a VNF [9].

### ETSI VIM

VIM is responsible for managing the NFVI network, storage, and compute resources. There may exist several VIMs to support the NFVO, being distributed, or not, over different physical places [9] [14]. According to [14], some of the main functions of the VIM are:

- Orchestration of the NFVI resources;
- Creation and management of virtual links, virtual networks, sub-nets, etc;
- Management of the software images requested by the NFV-MANO;
- Collection of performance information of the NFVI resources;
- Collection of the status of the hardware resources.

### 2.3 MANO PLATFORMS

In the most recent years, more and more Telecommunication Providers are adopting the NFV paradigm.

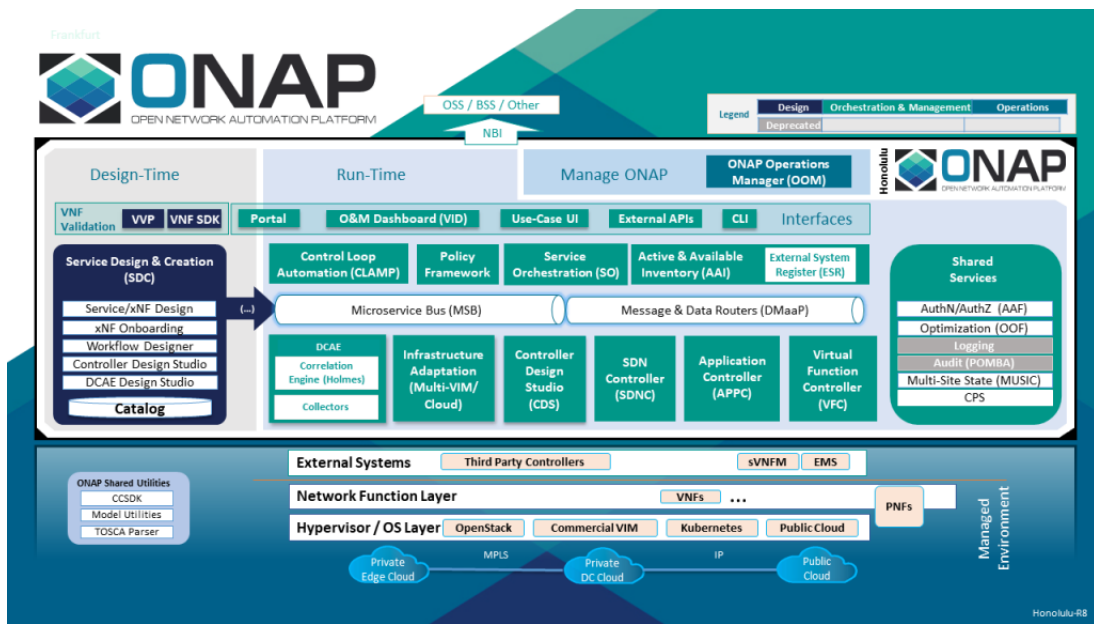
While there is continuous progress on the standardization of NFV, the implementation of the tools that allow this paradigm is still highly heterogeneous [15]. It is possible to notice this if we consider the several NFV MANO implementations that compete against one another. In the most recent years, several projects like ONAP, 5GTANGO, and OSM tried to improve MANO's implementation. These are among the most successful ones and will be addressed in the following sections.

### 2.3.1 ONAP

ONAP is an open-source project that enables the orchestration, management, and automation of network and edge computing services for Network Operators [16]. One of its goals is to create mechanisms that can provide fast and reliable network services without added costs for the operators. This project was founded by AT&T and China Mobile and is currently supported by the Linux Foundation.

ONAP relies on two main frameworks: (i) the design-time framework and (ii) the run-time framework. The first provides a development environment that offers the developers simplified mechanisms to develop resources, services, and products. It includes a vast number of policies for the design and implementation of network function and a Software Development Kit (SDK) with tools for packaging and validate VNFs. The run-time framework executes the rules and policies defined in the design-time environment and is responsible for the controllers that manage the virtual and physical resources [17].

Fig. 2.4 presents the architecture of ONAP, where are listed the components of each framework mentioned before.



**Figure 2.4:** ONAP's Honolulu Release - Platform Architecture Diagram [17].

At the date of this document, ONAP had more than a thousand people working on the project and expanded its scope to over thirty different projects. Besides, it has demonstrated its effectiveness in Voice over Long-Term Evolution (VoLTE) and virtual Customer Premises Equipment (vCPE), hence being used by some of the leading Telecommunication Operators, such as AT&T and China Mobile.

Regarding the VNF and NS Information Models, ONAP relies on Topology and Orchestration Specification for Cloud Applications (TOSCA) to describe them.

### 2.3.2 SONATA and 5GTANGO

SONATA is a framework that provides a development toolchain for the virtualization, management, and orchestration of network elements in NFV environments, being compliant with the ETSI NFV standards. This framework resulted from a Horizon 2020 European Union (EU) funded project that is part of the 5G Infrastructure Public Private Partnership (5G-PPP) initiative [18].

Being developed between 2015 and 2017, SONATA was a joint effort of fifteen partners representing telecommunication operators, service providers, Small and Medium-Sized Enterprises (SMEs), and academic institutes. Given the importance of this project, 5GTANGO was created as a follow up project of SONATA. SONATA was developed considering CI/Continuous Deployment (CD) methodologies focusing on improving software quality while decreasing the development time and the gap between development and operation [19].

The last release of SONATA, already under the scope of 5GTANGO, is named "SONATA powered by 5GTANGO" and presents three principal architectural components: (i) 5GTANGO's SDK, (ii) 5GTANGO's Verification & Validation (V&V) Platform, and (iii) 5GTANGO's Service Platform [20].

The SDK provides tools that empower the developers to better create and validate NFV services, for instance: descriptor validators, tools for the packaging and onboarding of VNFs, and local environments for testing and emulation. The V&V Platform ensures that the services can be tested on the Service Platform, offering advanced mechanisms to validate VNFs and NSs. Lastly, the Service Platform is where the VNFs and NSs are onboarded and instantiated. It offers service monitoring and collects Key Performance Indicators (KPIs) to ensure the Service Layer Agreements (SLAs) defined by the users aren't violated. Fig. 2.5 depicts these three main architectural modules.

According to [20], SONATA accomplished great results that were made possible by various key characteristics:

- SONATA presents a customizable MANO framework;
- SONATA provides platform recursiveness, implementing a MANO layer that runs on top of other platforms SONATA and may delegate the management of services to these platforms;
- SONATA provides network slicing support;
- SONATA is compliant with the ETSI NFV reference architecture.

Since SONATA and 5GTANGO are of utmost importance for this dissertation, they are described in further detail in Section 2.5.



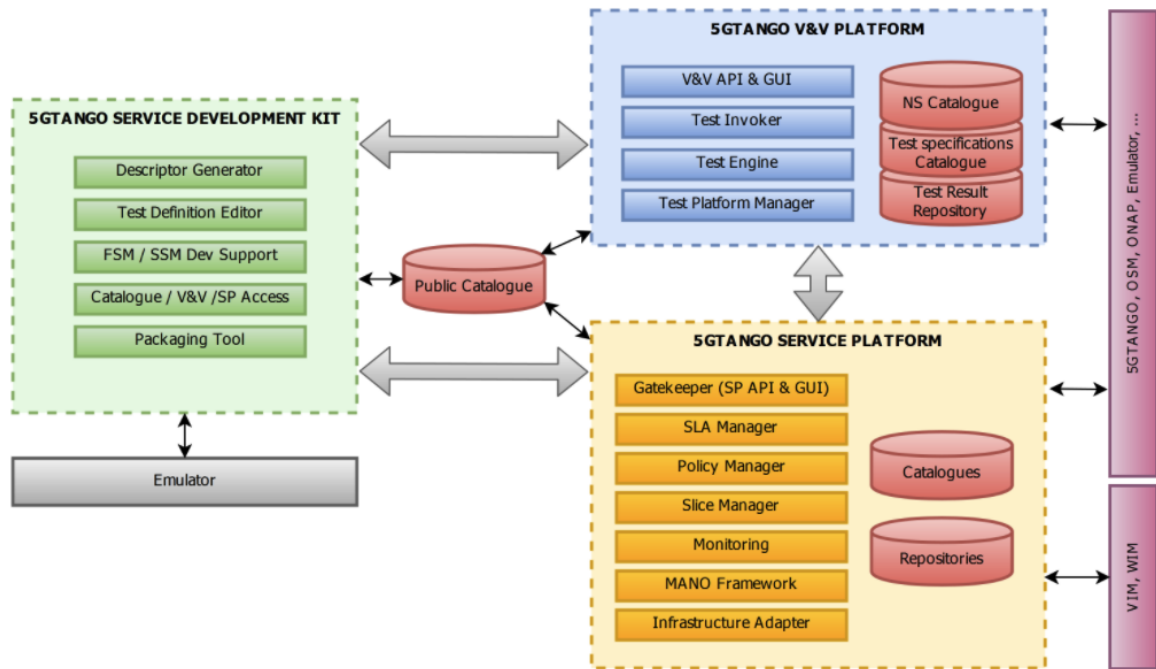


Figure 2.5: "SONATA powered by 5GTANGO"s architecture.

### 2.3.3 OSM

OSM is an ETSI initiative to develop an Open Source NFV Management and Orchestrator, aiming to provide Telecommunication Operators with an ecosystem to deliver NFV solutions rapidly and at a reduced cost. Being an ETSI initiative, OSM is completely aligned with ETSI NFV standards which simplifies its interoperability with other NFV implementations [21].

Currently, OSM is at Release 10, but in the scope of the 5GASP project, Release 9 was the chosen MANO, so this document will mainly address this release. Since Release 4, OSM is supported by the OSM Northbound Application Programming Interface (API), standardized in ETSI-NFV-SOL005. This API allows Operations Support System (OSS) and Business Support System (BSS) to interact with the NFVO through RESTful APIs.

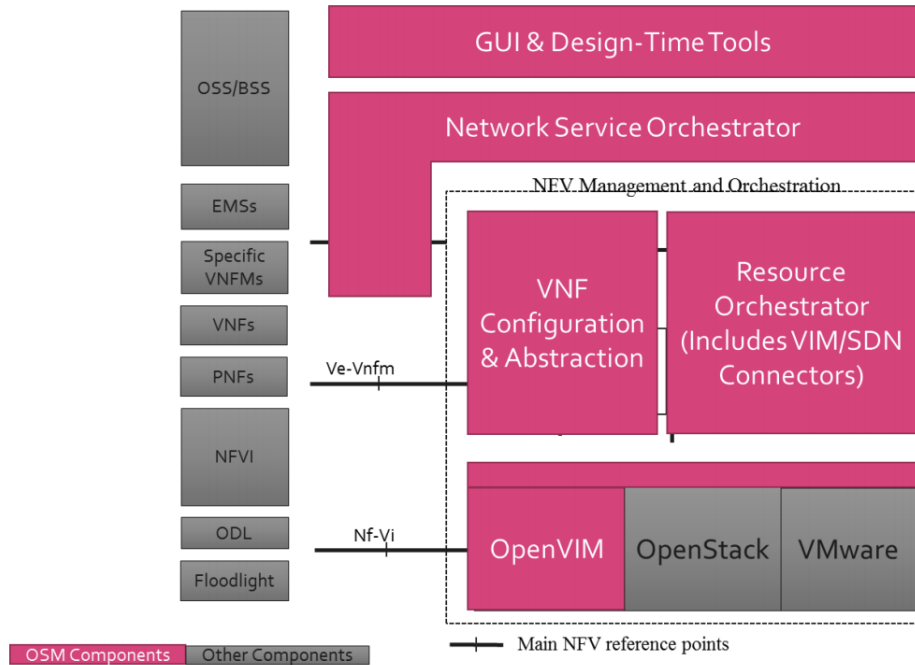
In December 2020, OSM's Release 9 addressed a new standard, ETSI-NFV-SOL006, replacing the syntax of the VNF and NS descriptors. Besides providing legacy support for the descriptors from previous releases, SOL006 allowed a new set of functionalities, achieved via the augmentation of the Yet Another Next Generation (YANG) Information Models (IMs) that describe the syntax of the descriptors [22] [23].

The configuration of VNFs and NSs is not only achieved via their descriptors. Juju Charms also play an essential role in the configuration of NFs, reducing the deployment complexity. These Charms allow the operators to add several specifications and configurations to the NFs, packaging them alongside the VNFs [24].

The implementation of OSM took into account some key characteristics, such as: simplicity, layering, modularity, and abstraction. This enables rapidly and cost-effectively adding new

features to OSM, which explains the ten different releases of OSM between October 2006 and June 2021.

Fig. 2.6 shows the leading architectural components of OSM, according to ETSI NFV MANO logical view.



**Figure 2.6:** OSM Main Architectural Components [25].

In Section 2.2.2 the ETSI MANO main components were already addressed. Further along this Section, the specific components of OSM will be addressed, since the understanding of its architecture is of extreme importance for the work presented in this dissertation.

### Virtualised Network Functions

The OSM’s VNFs follow the definition of ETSI VNFs, relying on virtualization to implement and deploy several NFs. They are packaged into objects that hold information about their functionality [11].

Each VNF package is composed of four main components: (i) the Virtualised Network Function Descriptor (VNFD) file, (ii) Juju Charms, (iii) configuration files and directories, for instance, the *cloud\_init* directory, and (iv) additional metadata, such as a README file.

The VNFD is a YAML Ain’t Markup Language (YAML) file that describes the functionality, specifications, and operational behavior of the VNF [11].

Despite being VNF optional elements, the Juju Charms are widely used to manage, configure and deploy cloud applications. Hence, its importance to NFV. These Charms will be further explored in this section.

Lastly, the *cloud\_init* directory might contain one or more *cloud\_init* files that enable the developer to upload metadata, user data, and vendor data to the VNFs. The *cloud\_init* files will be read during the boot of the cloud application and initialize the system accordingly [26].

## Network Services

OSM's NSs, resembling OSM's VNFs, follow ETSI NFV definition. They are distributed as packages and contain information about how the VNFs are interconnected.

NSs contribute to the behavior of the higher layer services, resulting from the combination of the behavior of the VNFs with the behavior of the network infrastructure defined by the NS [11].

Similar to VNFs, NSs are composed of: (i) a Network Service Descriptor (NSD) file, (ii) configuration files and directories, and (iii) additional metadata. The descriptor file defines the network topology of its constituent VNFs, for instance, through Virtual Links (VLs) [11].

## Information Models

OSM's IMs define how the VNFs and NSs are described, enabling the lifecycle modeling of these entities. Since ETSI NFV defines infrastructure agnostic NFs, its IMs follow the same premise. Thus, allowing the definition of VNFDs and NSDs that are not coupled to the underlying infrastructure [27].

YANG is a data modeling language widely used to describe data structures, hence, being chosen by ETSI to create OSM's IMs. OSM's IM is defined via several YANG models and their augmentations and, since release 9 of OSM, is strictly aligned with ETSI-NFV-SOL006, being aligned with SOL005 in the previous releases.

The OSM's IMs are defined via a tree structure, where each node is defined as a tag. A tag can have two different designations: a container or a leaf. A container defines a new level on the tree, and can encapsulate new containers or leaves, thus not having a defined data type. Contrastingly, leaf nodes contemplate concrete information that defines a value in the context of the descriptor [28]. Hence, leaf nodes have defined data types: *boolean*, *integer*, *string*, *enum*, or reference. A reference leaf, as the name suggests, references another leaf. For instance, it is very common to define a VNF connection point and then reference it in other tags.

A container can be defined by the following schemas: container, list, case, and choice. A container defined with the schema container is a "pure container" that simply creates another tree level. A list container defines a list of other entities, indexed by an id.

Regarding the leaf nodes, they can be mandatory or optional.

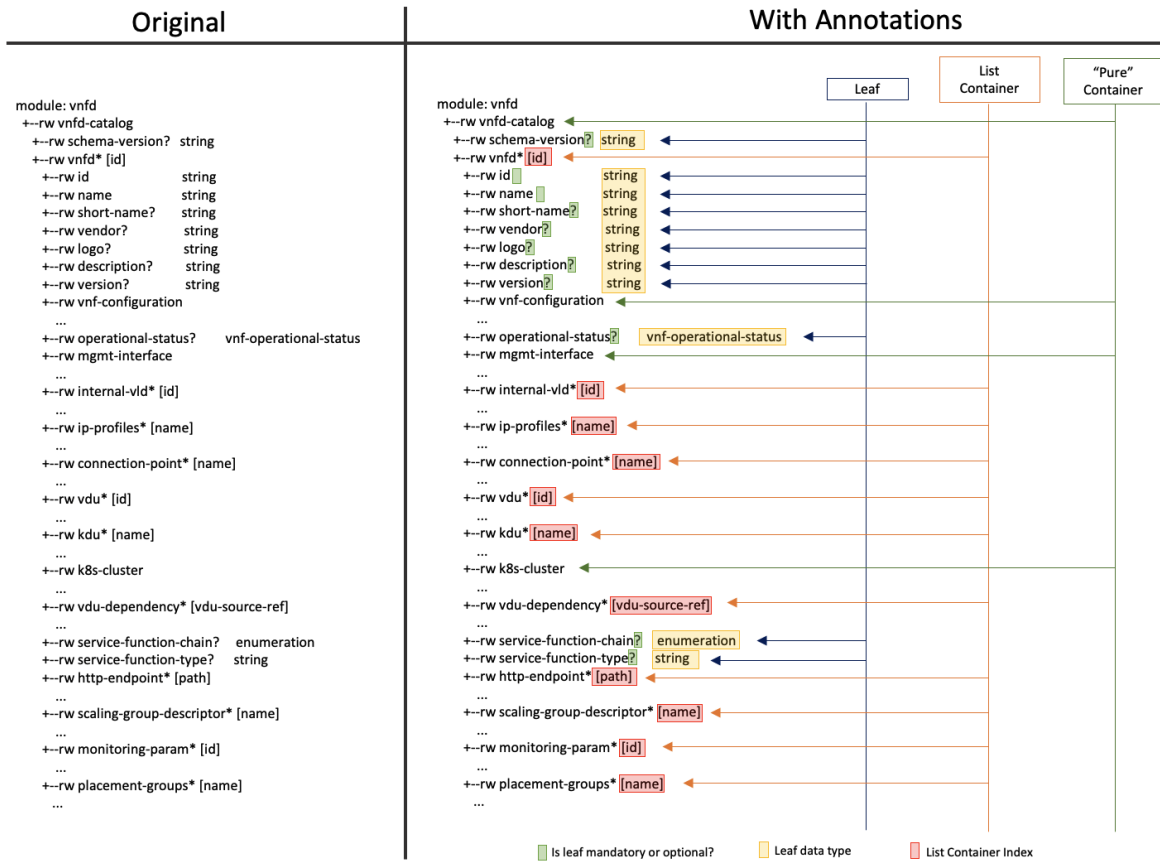
Fig. 2.7 presents the first three levels of a VNF descriptor, such as some annotations explaining each tag.

## Architecture and Components

OSM aims to achieve a modular architecture, thus being implemented as a collection of components, or services, that communicate with each other via a Kafka<sup>1</sup> bus. Besides these components, OSM also uses common databases to store information about the deployed NF and the system itself.

---

<sup>1</sup><https://kafka.apache.org/>



**Figure 2.7:** Description of VNFD's Tags.

OSM is composed of the following components [29] [30]:

- **Lightweight User Interface (UI)** - Provides an access point to OSM users, offering simplified management of the VNFs and NSs lifecycle. To interact with other OSM components, the UI relies on the Northbound Interface (NBI);
- **Northbound Interface (NBI)** - Provides a Representational State Transfer (REST) API, based on ETSI-NFV-SOL005, to interact with OSM, allowing, for instance, the onboarding and lifecycle management of VNFs and NSs;
- **Lifecycle Manager (LCM)** - Is responsible, as the name suggests, for the lifecycle management of multiple VNFs. LCM is involved in the onboarding, instantiation, scaling, and migration of VNFs and NSs;
- **VNF Configuration and Abstraction (VCA)** - This component acts as a VNFM, being responsible for enabling the configuration of VNFs. Alongside Juju, one of VCA's responsibilities is to signal the VNFs to perform specific actions;
- **Resource Orchestrator (RO)** - Manages and orchestrates the resources needed for the OSM NFV environment. The RO component controls and allocates resources through VIMs and Software Defined Network (SDN) controllers, deploying the necessary VMs and orchestrating the network used by the VNFs;

- **Monitoring Module (MON)** - Collects metrics through VIM and VCA and exposes them to be consumed by Prometheus<sup>2</sup>, a Time Series Database (TSDB). Besides, it iterates through the alarms defined by the Policy Manager (POL) module and evaluates them. This way, MON allows the continuous monitoring of the entities that are deployed in OSM [31]. Its architecture can be observed in Fig. 2.8.
- **Policy Manager (POL)** - Enables the creation, management, and triggering of alarms based on VNF and infrastructure events and metrics.

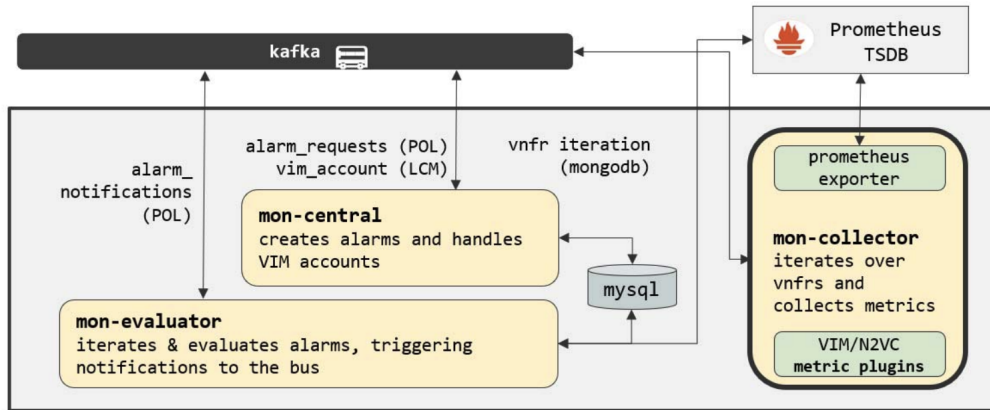


Figure 2.8: OSM MON - Internal Architecture. [31]

Fig. 2.9 presents the architecture of OSM Release 5. Until the last release of OSM - Release 10 - this architecture has fundamentally remained the same, with just some enhancements of the individual components.

From OSM Release 5 to OSM Release 10, these enhancements mainly address (i) the scalability of VNFs and Kubernetes deployments, (ii) improvements in Juju's functionalities, (iii) support for Microsoft Azure to be used as VIM, and (iv) high-availability scenarios [32][33][34][35][35].

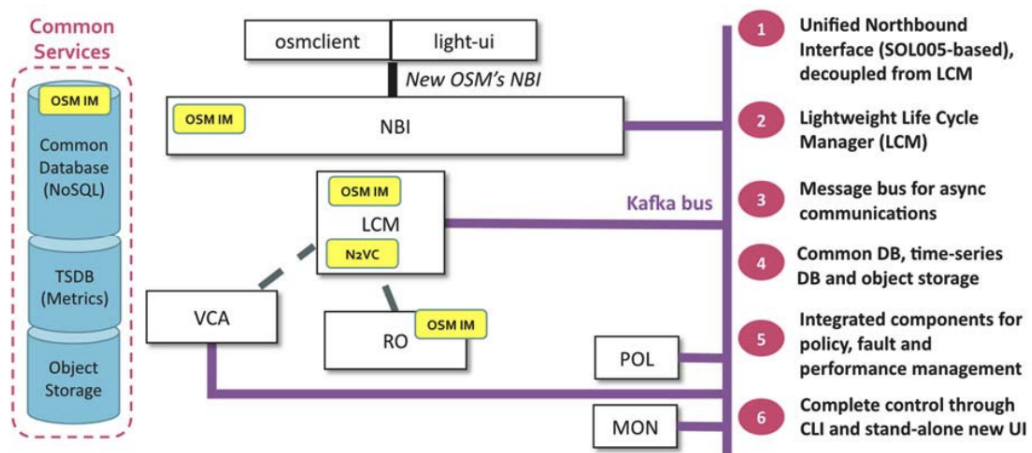


Figure 2.9: OSM Release 5 - Architecture. [31]

<sup>2</sup><https://prometheus.io/>

## Juju Charms

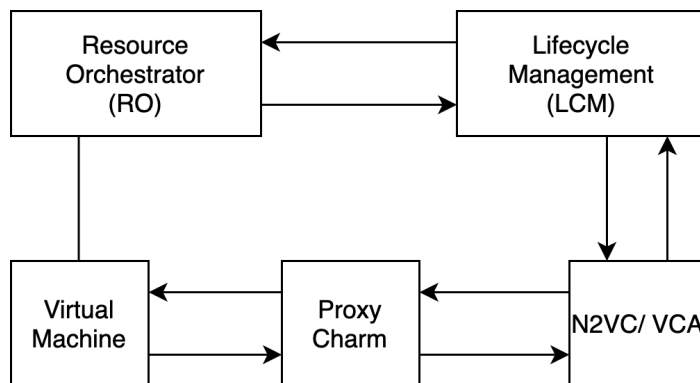
Juju<sup>3</sup> is an open-source tool that enables simplified management of cloud software services, allowing a quick and efficient deployment, configuration, management, maintenance, and scaling of cloud applications [36].

Juju encapsulates all the code that enables these operations in what is described as Charms. Juju Charms are collections of YAML files and scripts that allow the installation of software, service management, and so much more, and it is through Charms that Juju provides all the information to configure and monitor virtual instances [37].

Charms are defined by three main concepts: (i) actions, which are the programs that will be executed, (ii) hooks, being the signals that will trigger an action, and (iii) layers. Juju Charms are composed of several layers, each one composed of actions and hooks. This architecture allows a simplified modification of Charms, increasing their reusability and modularity.

OSM allows the VNF developers to upload their Charms in order to simplify the configuration and management of the VNFs. These Charms have two mandatory layers: the basic layer and the VNF proxy layer. The basic layer imports the reactive framework that contains all the code needed for the other layers to work, while the VNF proxy layer imports the required code to run actions via Secure Shell (SSH) in the VM. Besides these two layers, VNF developers can also add more layers to their Charms. The most used layers in OSM's context are: (i) the metrics layer, which allows the gathering of VNF metrics, (ii) the REST API layer, that imports the required code to run actions via REST on the VNF, and (iii) the Netconf layer, that enables to run VNF actions via Netconf primitives [38].

Figure 2.10 presents how the VNF proxy charm fits in the OSM workflow. First, a VNF package is instantiated through the LCM; then, the LCM requests a VM from the RO, that will instantiate it. After the VNF is instantiated, the LCM will order the VCA to deploy the VNF proxy charm, informing it on how to access the VM [38].



**Figure 2.10:** VNF Proxy Charm Integration with OSM's Workflow. Adapted from [31].

Juju will then perform day 0 and day 1 configurations on the VM. Day 0 configuration avoids the usage of pre-built images. During this phase, the basic configurations, such as SSH keys generation and setting the hostname, are performed. These configurations ultimately

<sup>3</sup><https://juju.is/>

```

|-- my_charm
  |-- README
  |-- config.yaml
  |-- actions.yaml
  |-- icon.svg
  |-- layer.yaml
  |-- metadata.yaml
  |-- actions
  |   --- my_action
  |-- reactive
  |   --- my_action.py
  |-- tests
  --- copyright

```

**Code Block 1:** Structure of a New Juju Charm Layer.

allow communication and access to the VM. Day 1 configurations are performed by the actions and scripts defined in the Charms. These mostly consist of the downloading of software packages and execution of services and, when these are over, it is expected that the VM is fully working [37].

To further comprehend Juju Charms, it is crucial to analyze their structure. Code Block 1 presents the base structure of a new Juju Charm layer.

New layers are composed of the following files and directories:

- **README** - file that contains useful information for the Charm's users. This document should be the first one that the charm's user reads;
- **config.yaml** - file that defines the configuration of the Charm, containing a list of options where each setting is defined;
- **actions.yaml** - file that presents the description of the actions of the Charm;
- **icon.svg** - the picture that is associated with the charm;
- **layer.yaml** - file that presents the layers on which this new layer is based;
- **metadata.yaml** - file that includes several high-level information, such as the charm's creator name, Charm's description, etc;
- **actions** - directory that contains the actions that will be performed by the Charm. Each action is defined via a script, and it is the connection point between the reactive platform and the signal to perform an action;
- **reactive** - directory that contains several reactive handlers. These handlers are written in Python<sup>4</sup> and contain the actual implementation of the actions;
- **tests** - directory that contains several tests to validate the Charm's behavior;
- **copyright** - file that contains the Charm's license or copyright.

## 2.4 DEVOPS AND CONTINUOUS INTEGRATION

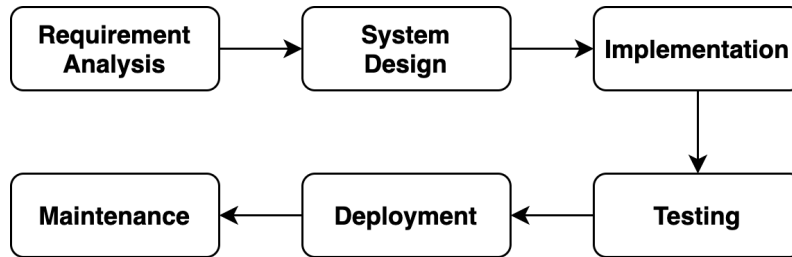
The Software Development Lifecycle (SDLC) is very extensive and involves multiple teams. These teams often do not have good communication policies, thus increasing the development time.

---

<sup>4</sup><https://www.python.org/>

The Waterfall model was the first widely adopted model for the SDLC, dividing it into several separate phases executed sequentially. Here, the output of a phase is used as the input for the next phase.

Fig. 2.11 illustrates the different phases of the Waterfall Model.



**Figure 2.11:** Phases of the Waterfall Model.

This model presented several disadvantages: (i) not good enough model for complex projects, (ii) not suitable for projects where the requirements may change during the course of SDLC, (iii) difficulty in making adjustments after the initial stages, among others.

Given these problems, the Agile methodology was introduced. It aimed to improve the interactions between individual teams, thus improving and paving the way for better and smoother collaboration [39].

Agile uses iterative processes that result in incremental delivery. Hence, a project can be divided into multiple releases, that gather the output of several iterations [40]. This way, the stakeholders can closely and continuously monitor the quality of the software that is being developed and give feedback to the development team.

Every iteration is composed of several user stories that map the software requirements. These stories are translated to tasks that are then passed to the development teams, and will, ultimately, guide the SDLC [40].

Although Agile presented an optimization on software development, it did not offer guidelines for the software operation tasks [41] since it only focused on improving the development processes. While the development teams focus on software creation, thus being change-oriented, the operation teams seek the stability of the software. This misalignment of goals led to the introduction of methodologies that contemplated both software development and its operation. The most known methodology is DevOps.

Section 2.4.1 addresses this methodology, while Section 2.4.2 focuses on Continuous Integration, one of the most crucial elements of DevOps.

### 2.4.1 DevOps Methodology

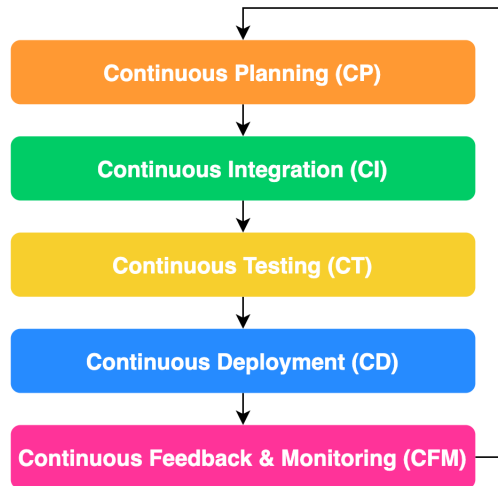
DevOps was firstly introduced in 2009 and it is derived from the combination of Development (Dev) and Operations (Ops) [42][43]. It enables better collaboration between the development and operation teams and recognizes the need to continuously integrate software development with operational deployment [44], thus extending Agile [42].

DevOps is defined by four dimensions: (i) collaboration, (ii) automation, (iii) measurement, and (iv) monitoring [39]. Regarding the collaboration component, DevOps states that both



the development and the operation teams have to make an effort to be aware of the other team's tasks, which demands continuous communication between the teams. The automation component enables gains in efficiency, while the measurement and monitoring components lead to collecting logs and metrics, not only of the deployment but also from the teams that are working on the software [39].

Fig. 2.12 depicts the phases of the DevOps methodologies [45].



**Figure 2.12:** DevOps Stages.

Continuous Planning (CP) aims to bring together all the developers, operations team members, testers, and stakeholders, in order to gather all the requirements and prepare a release plan.

Continuous Integration (CI) addresses the integration of new code in the code-base. Using CI tools, the new code can be automatically tested before it is integrated, thus preventing the insertion of errors in the code-base and avoiding errors in future builds. With CI, the developers can quickly receive feedback about the code they are producing, and act accordingly.

Continuous Testing (CT) can also be addressed as a stage of the CI process. It aims to automate all testing stages, ensuring that the software being produced respects the quality standards previously defined. One of the most crucial challenges in CT is the creation of testing environments that replicate the production environments. If the testing and production environments are heterogeneous, the CT phase will not have the desired output.

Continuous Deployment (CD) enables automated deployment of the software in the production environment. If CI and CT phases are successful, the code is ready to be deployed. Here is where CD comes into action. CD reduces the deployment time and the complexity of deploying an application.

Lastly, Continuous Feedback and Monitoring (CFM) addresses the automated monitoring of the deployed software. With CFM tools, it is possible to continuously collect metrics and logs that allow the operation teams to understand if the software behavior is different from what is expected. This way, the operations team can anticipate errors, leading to a higher user satisfaction rate.

### 2.4.2 Continuous Integration

Continuous Integration is a software development practice that allows that whenever a new code change is committed to a code repository, an automated build is triggered. This build will be validated against several requirements and tests, which will be performed automatically [46]. After the building and testing phase, the code developers will be informed if the newly committed code follows the standards needed to follow through to the integration phase.

This concept was introduced in 1991 by Grady Booch, and although he initially not suggested for this practice to be used as an integral part of a software project, the increased complexity of business procedures and software development lead to CI being a fundamental component of the SDLC [47].

In Continuous Integration, the code repository acts as a central hub for source code, keeping track of all the changes committed by all the developers. It allows the versioning of the code-base, which is a significant advantage. In a CI scenario, the repository will detect whenever new code is submitted and inform the CI server [48], which will gather all the latest code changes and start a pipeline, automatically, that will test the new code, build it, and, if all is according to what is expected, persist the code changes in the code repository.

The usage of a CI server enables the configuration of the environment where the code will be tested. The developers can create scripts to configure a testing environment, recreating the deployment conditions, which adds more trust to the process.

CI servers also provide a set of additional features that might interest the developers, like scheduling and visualization features.

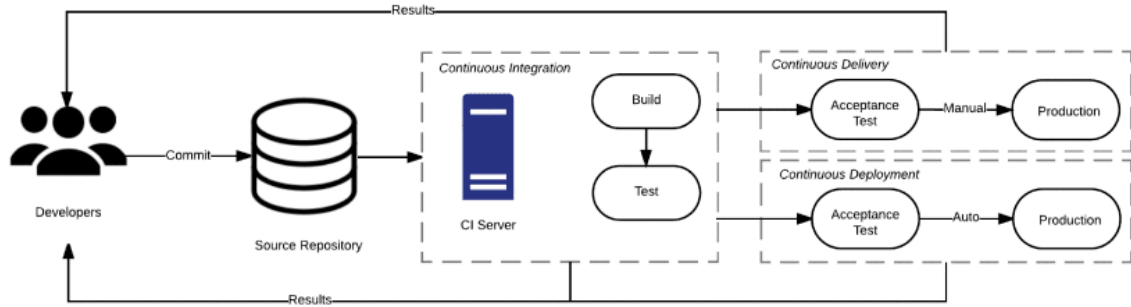
Section 2.4.4 further explores the tools that enable CI.

Regarding the benefits of using CI, the following are some of the utmost important ones:

- Early detection of bugs, which simplifies the process of fixing them;
- With Continuous Integration, if the developers wish to roll back to a previous version, less code will be lost since new code is continuously being added to the code repository;
- The current build is constantly available for testing, demo, or release purposes;
- The process of continuously integrating new code leads the developers to create modular and less complex code;
- Faster releases;
- Increase in customer satisfaction;
- Cost reduction.

Since CI abruptly changes the SDLC, some challenges may arise during its implementation in software development teams. These challenges may be motivated by (i) the lack of investment in CI tools and during the implementation of this methodology, (ii) developers' resistance to the adoption of the CI paradigm, (iii) difficulty in changing old organizational culture and policies, and (iv) lack of proper testing strategies [46].

As mentioned in Section 2.4.1, Continuous Integration is not an isolated process, being the starting point to achieve, for instance, Continuous Delivery and Continuous Deployment. Fig. 2.13 depicts the relationship between these practices [46].



**Figure 2.13:** Relationship between Continuous Integration, Continuous Delivery, and Continuous Deployment [46].

With all the benefits presented above, it comes naturally that this practice is being widely adopted as an integral part of the SDLC. Nowadays, Continuous Integration is being used in software development companies and for telecommunication providers, for instance, in NFV. So, it comes naturally to adopt CI practices in the 5GASP project.

### 2.4.3 Leveraging DevOps and Continuous Integration in a NFV Scenario

As stated in Sections 2.1 and 2.2, it is expected that the physical network devices will be replaced by VNFs, with analysts predicting millions of additional devices on the network. Nowadays, the deployment of network hardware is one of the major bottlenecks in obtaining a faster network speed and reliability. Besides, the usage of hardware-based network systems also contributes to long recovery times from failures.

With the symbiosis of NFV and DevOps, these problems can be easily addressed. The DevOps paradigm is the best way to guarantee a smooth transition between physical network devices and NFV/SDNs, providing better services to the end-users. It enables the collaboration between network operators and developers, allowing the automated deployment of new NFs [49].

Using DevOps and NFV, the number of configurations needed to deploy new devices can be easily reduced. Besides, upgrading these virtual network equipment will also be facilitated, taking less time and costing less money. The Network Functions can be deployed on container-based solutions, over COTS, and scaled according to the needs of the network users, which results in time-to-market advantages, and a better service provided to the users[49].

The DevOps methodology also enables a better testing phase of the Network Services. Leveraging the automation provided by this paradigm, the NFs can be tested with almost no human intervention and deployed to the network in a matter of minutes, which is a clear advantage over the current network paradigm.

### 2.4.4 Continuous Integration Tools

Regarding Continuous Integration tools, several aspects can heavily impact their usability, thus the need to study each available option before choosing the most viable for a specific use case. These aspects are (i) customization, (ii) the existence of APIs to interact with the CI

tool, (iii) the possibility of installing the tool on-premises, (iv) the need for integration with Source Code Managements (SCMs) repositories, and (v) cost.

Table 2.1 presents an overview of some of the most used CI tools.

| <b>Tool</b>              | <b>Interaction via API</b> | <b>Possibility of self-hosting?</b> | <b>Requires SCM repository integration?</b> | <b>Customization</b>                                                          | <b>Cost</b>                                                                                                         |
|--------------------------|----------------------------|-------------------------------------|---------------------------------------------|-------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <b>Jenkins</b><br>[50]   | Yes                        | Yes                                 | No                                          | Wide variety of plugins                                                       | Free                                                                                                                |
| <b>Circle CI</b><br>[51] | Yes                        | Yes                                 | Yes                                         | Does not offer plugins, but claims to have all needed customizations built-in | Free on the cloud, but with limited builds (Freemium)                                                               |
| <b>TeamCity</b><br>[52]  | Yes                        | Yes                                 | No                                          | Wide variety of plugins                                                       | The free self-hosted version only allows for 100 different build configurations (Freemium)                          |
| <b>GitlabCI</b><br>[53]  | Yes                        | Yes                                 | Yes                                         | Offers a limited number of plugins                                            | Free on the cloud, but with time restrictions and free self-hosted option, but with feature restrictions (Freemium) |
| <b>Travis CI</b><br>[54] | Yes                        | Yes                                 | Yes                                         | Offers a limited number of plugins                                            | Free but with build restrictions (Freemium)                                                                         |

|                         |     |     |     |                                    |                                                                                                                                         |
|-------------------------|-----|-----|-----|------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>Drone CI</b><br>[55] | Yes | Yes | No  | Wide variety of plugins            | Free on the cloud, but with some restrictions. Offers a free version for on-premises installations but with limited features (Freemium) |
| <b>Bamboo</b><br>[56]   | Yes | Yes | Yes | Offers a limited number of plugins | No free options                                                                                                                         |

**Table 2.1:** Overview of the Most Used CI Tools.

#### 2.4.5 Test Automation

Test Automation is an Information Technology (IT) practice that automates testing and validation processes so that human intervention is as reduced as possible. It relies on a collection of automation tools to validate the behavior of an application and output comprehensive results which the application developers can consult.

This methodology allows to automatically perform code analysis, unit tests, integration tests, and so many other types of tests, relying on Test Automation Frameworks (TAFs) to define a testing flow and execute it.

##### Test Automation Frameworks

Testing Frameworks create a collection of rules for developing and testing application test tools, comprising a set of tools combined with Quality Assurance (QA) policies and practices. They aim to increase the application's test coverage, ensuring that the developed tests are reusable, easy to maintain, and stable, thus simplifying the testing and validation processes.

These tools enable better testing efficiency and accuracy, reducing the costs of testing applications and providing the following benefits: (i) reduced human intervention, (ii) testing code reusability, (iii) lower maintenance and test development costs, and (iv) improved test efficiency.

Currently, several TAFs are widely used. Some of the most relevant ones are now addressed.

##### Katalon Studio

Katalon Studio allows test performing on (i) web applications, (ii) APIs, (iii) mobile applications, and (iv) desktop applications. It provides an Integrated Development Environ-

ment (IDE) for test development so that less technical users can easily create tests. Besides this, it also includes test scripting using Java or Groovy [57].

This tool is easily integrated within CI/CD pipelines and provides good visualization reports of the testing process.

Katalon Studio provides hundreds of built-in keywords for the development of tests and can be extended via plugins, available in the Katalon Store [57].

Although, Katalon is shipped as a freemium tool, with limited features on its free tier.

### Robot Framework

Robot Framework mainly targets the testing of web applications and APIs, although, in conjunction with other tools, it may be used to test mobile and desktop applications [58].

To create test cases, Robot relies on human-readable keywords and straightforward syntax.

Robot Framework is a free and open-source tool with a vast community that continuously contributes to extending its capabilities. These can easily be extended through Java and Python implemented libraries [58]. One other great advantage is the easier installation of this Testing Automation Framework and the fact that it is very lightweight.

Regarding the testing process output, Robot Framework produces Extensible Markup Language (XML) and HyperText Markup Language (HTML) interactive report files, which can be integrated, for instance, with Jenkins' dashboard.

### Selenium

Selenium is an open-source TAF that mainly targets web applications and APIs, with its tests being written in various programming languages, such as Java, Python, or Ruby, for instance. Besides this, Selenium provides several modules to augment its testing capabilities [59].

From these components, the Selenium WebDriver, which executes testing scripts using browser-specific drivers, and Selenium Grid, which parallelizes tests execution, are among the most important ones [60].

Selenium is easily integrated with CI/CD tools. Although, it lacks good visual test reports, which can be generated, but require the usage of other tools.

### Eggplant

Eggplant provides machine learning models to help application developers to automate the testing process.

For test creation, Eggplant provides a Graphical User Interface (GUI) and a proprietary language named Sense Talk. Although not being commonly known, this language is English-like which makes it easy to use [61].

Regarding the System Under Test (SUT), Eggplant mainly addresses the testing of GUIs and can be easily integrated with CI/CD tools such as Jenkins or Bamboo [61].

One of Eggplant's most significant disadvantages is that it doesn't provide a free tier, which might lead to some users and companies not choosing this tool.

## Citrus

Citrus is a testing framework that enables the creation of fully automated test cases for applications following a service-oriented architecture. This framework is written in Java and is open-source, currently available on Github [62].

To test applications, Citrus provides support for several protocols, such as Hypertext Transfer Protocol (HTTP), Simple Object Access Protocol (SOAP), Transmission Control Protocol (TCP), and SSH. This enables Citrus to test end-end systems [62].

Although not providing visual reports by default, the Citrus configuration can be tweaked to produce HTML interactive reports, simplifying the test results analysis process.

### **2.4.6 Monitoring Tools**

Monitoring tools enable monitoring and collecting metrics and logs from infrastructure components and services. These metrics, for instance, can be system, network, or storage-related and provide useful information about the availability of components and services, simplifying its management.

Through monitoring, it is possible to gather the behavior of a system and act accordingly. For instance, when considering a web application, if monitoring tools capture that more clients are using it, it is possible to scale some application's components, thus improving its performance and ultimately offering a better user experience.

Regarding the DevOps methodology, these tools are part of the Continuous Feedback and Monitoring stage.

Some of the most used monitoring tools are introduced in this section.

## TICK Stack

TICK Stack is composed of four components: (i) Telegraf, (ii) InfluxBD, (iii) Chronograf, and (iv) Kapacitor. These components capture and monitor several metrics and offer a visualization dashboard, so its understanding is straightforward [63].

Telegraf is a collection metrics agent, which uses plugins to scan for the intended metrics and store them in InfluxDB. This agent can easily be deployed in a VM, for instance, and almost instantly start the metrics collection process. One of its significant advantages is its straightforward configuration [63].

InfluxDB is a TSDB built to handle high query loads. In TICK Stack, this component is responsible for storing the metrics collected by Telegraf and making them available to Chronograf [63].

Chronograf is a visualization interface, allowing the visualization of the metrics stored in InfluxDB. Its usage is effortless since it includes several templates, which enable a faster and simpler dashboard development process. Besides, the configuration of this tool can be accomplished using only its web interface, which simplifies its configuration process [63].

Lastly, Kapacitor processes the InfluxDB metrics and provides alarmistic mechanisms. Besides, it is capable of performing several actions according to the raised alarms [63].

TICK Stack is open-source and free, which makes it highly desirable among several IT companies.

### Prometheus and Grafana

Prometheus is an open-source monitoring tool. Besides providing monitoring mechanisms, Prometheus also includes a built-in rules engine, which provides alarmistic [64].

To collect metrics, Prometheus has several exporters available. These are deployed in the SUT, collect metrics, and make them available by creating HTTP endpoints that can then be scraped by Prometheus[64].

To store the collected metrics, Prometheus can use its internal TSDB or use remote storage points.

Prometheus, although very powerful, has a complex installation and configuration process. However, most IT companies still choose to use this tool to monitor their infrastructure and services, not being discouraged by the complexity of Prometheus' configuration.

On the other hand, Grafana is a powerful dashboard and visualization tool, which can be integrated with several TSDBs. When deployed alongside Prometheus, Grafana enables several visualizations on the data collected by Prometheus, thus offering a comprehensive dashboard for analyzing the collected metrics [65].

### ELK Stack

In contrast with TICK Stack and Prometheus, ELK Stack doesn't focus on metrics monitoring but rather on textual logs monitoring.

Regarding its components, ELK Stack is comprised of three components: (i) Elasticsearch, (ii) Logstash, and (iii) Kibana [66].

Elasticsearch is a NoSQL database that offers ample volume data storage, indexes it, and allows advanced queries on the stored data. It relies on the Apache Lucene search engine and provides several REST APIs so that other components can interact with it [66].

Logstash is responsible for collecting data, selecting useful information from it, and storing it in Elasticsearch. It enables uniforming the data collected from different sources and cleansing it [66].

Lastly, Kibana provides visualization dashboards to enable users to visualize the data stored in Elasticsearch and have insights into this information. Configuring and using Kibana is very simple since it provides, for instance, built-in dashboard templates and a straightforward process of connecting this service to Elasticsearch [66].

In 2015, ELK introduced Filebeat, a logs collection mechanism that can easily be configured in the SUT to ship logs to Logstash, storing them in Elasticsearch. Currently, Filebeat counts with several tailor-made modules developed to collect service's specific logs, simplifying the logs collection process [66].



## 2.5 RELATED WORK

In recent years, several 5G European projects arose, such as (i) 5G-VINNI<sup>5</sup>, (ii) 5GTANGO, (iii) 5GEVE, and (iv) 5GinFIRE. The following sections focus on four different projects with significant contributions to establishing 5G environments and networks. 5G-VINNI presented an open-source OSS for delivering Network Slice-as-a-Service (NSaaS), while 5GTANGO, described in Section 2.5.2, was one of the pioneer projects regarding the testing and validation of NFs [67]. Section 2.5.3 describes the 5G EVE project, which aimed to implement and test advanced 5G infrastructures in Europe [68]. Lastly, in Section 2.5.4, the 5GinFIRE is addressed. This project aimed to create an experimental playground where new components and NFs could be tested before being deployed in production environments [69].

### 2.5.1 5G-VINNI

5G-VINNI - 5G Verticals INNOvation Infrastructure - is a Horizon 2020 European project that strives to provide an end-to-end (E2E) facility to validate the performance of the 5G technologies. This project had its kickoff in July 2018 and will last 36 months [70].

It aims to (i) design an advanced E2E facility, (ii) provide zero-touch orchestration, operation, and management of the previously mentioned facility, (iii) validate 5G KPIs and support the execution of vertical use cases, and (iv) point out the relevant standards and open source communities in the 5G community [70].

5G-VINNI accomplished good results regarding Network Slicing, developing internal NSaaS mechanisms. Besides this, it also advanced the state of the art in regards to KPIs Validation.

In the context of this dissertation, regarding the contributions of 5G-VINNI, only Openslice<sup>6</sup> is addressed in detail, since it presents a good platform for orchestrating and managing network slices and is a viable candidate to be used during the 5GASP project.

#### Openslice

Openslice is an open-source OSS for delivering NSaaS. It supports the onboarding of VNFs and NSs to OSM, so as the management of these entities. Openslice presents several services that follow the Tele Management Forum (TMF) standards, for instance, the Service Catalog, implemented according to the TMF633 specification [71] [72]. Since TMF standards are an integral part of the Openslice architecture, it is of interest to describe them in further detail. This is accomplished in Section 2.5.5. These standards are crucial to enable third-party applications to interact with Openslice.

To achieve an NSaaS paradigm, Openslice follows three main phases [71]:

1. **Slice Ordering** - This phase starts by ordering Service Specifications related to the Network Services previously onboarded. The Service Specifications can be (i) Customer-Facing Services (CFS) or (ii) Resource-Facing Services (RFS). After ordering the Service Specifications, a Network Slice Template (NEST) is modeled as a TMF Service

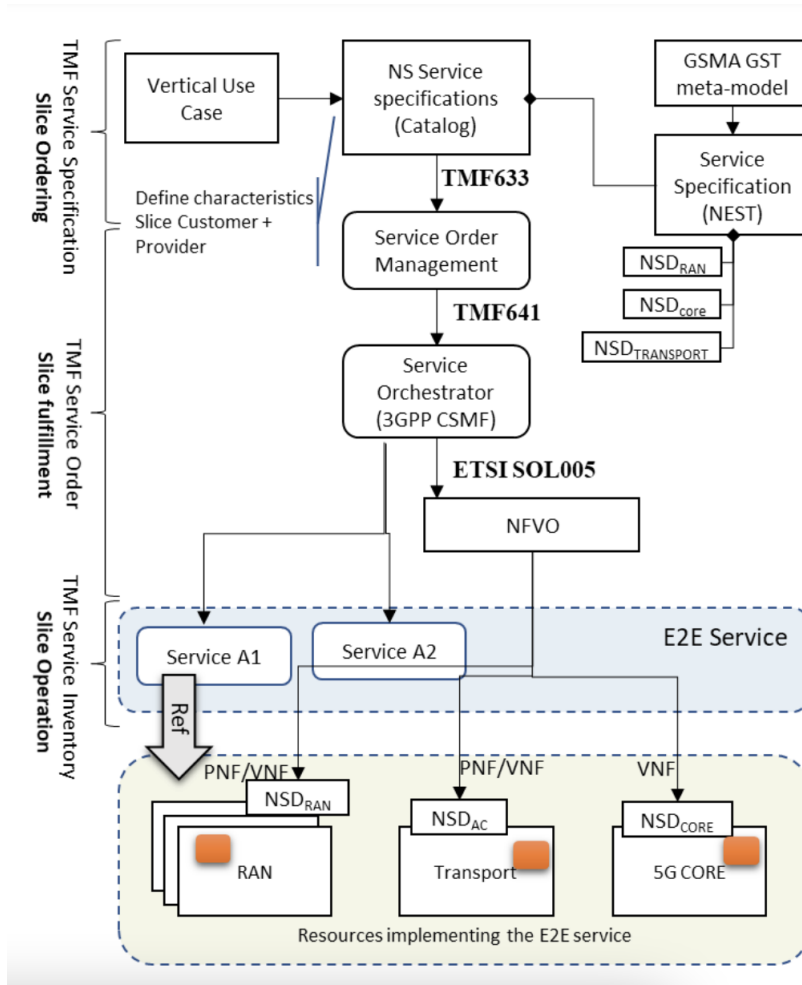
---

<sup>5</sup><https://www.5g-vinni.eu/>

<sup>6</sup><https://openslice.readthedocs.io/en/stable/>

- Specification. This concludes the Slice Ordering phase, where a customer requests a Service Specification in order to create a Service Order;
2. **Slice Fulfillment** - This phase starts with the Service Order's triggering. When this happens, the Service Order will be translated to a collection of resource requirements that will be sent to the NFVO using ETSI-NFV-SOL005. Then, the NFVO will allocate a slice instance.
  3. **Slice Operation** - During this phase, the customer can track the status of the slice and get run-time information about it. To do so, the customer uses the Service Inventory API, defined according to the TMF638 specification.

Fig. 2.14 depicts Openslice's approach to NSaaS and all the TMF specifications used.



**Figure 2.14:** Network Slice Instantiation in Openslice [71].

Openslice can also be used in multidomain scenarios through federation. To achieve this, Openslice exchanges Service Specifications and Catalogs and makes Service Orders between different Organizations.

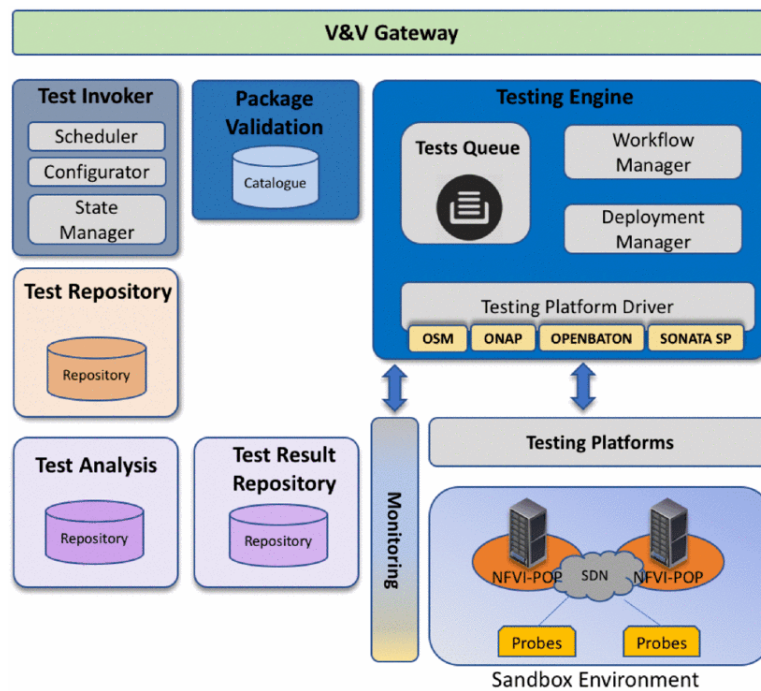
### 2.5.2 5GTANGO

5GTANGO is a 5G-PPP Phase 2 Initiative that extended the SONATA project. It was a collaborative effort between several partners, mainly Telecommunication Operators and research and academic institutes, taking place between June 2017 and the beginning of 2020 [67].

The main goals of this project were: (i) to reduce the time-to-market of network services, (ii) reduce the needed effort of third party developers when creating VNFs, (iii) enable better customization and adaptation of network verticals, and (iv) accelerate the industry’s adoption of NFV, by adopting a DevOps model to validate Network Services [67].

One of the most significant outcomes of 5GTANGO was its V&V platform, which offered advanced mechanisms to validate the VNFs and NSs. The main components of this platform are: (i) the V&V Gateway, which exposes an API to interact with this service, (ii) the Test Invoker, responsible for test configuration and scheduling, (iii) the Test Repository, that stores the Testing Descriptors, (iv) the Package Validator, for the validation of the submitted packages (descriptors, for instance), (v) the Testing Engine, that is responsible for deploying the tests, (vi) the Test Results Repository, that stores the outputs of the testing phase, and (vii) the Test Analysis Component, responsible for the results processing and visualization [73].

Fig. 2.15 depicts the components mentioned before.



**Figure 2.15:** 5GTANGO’s V&V Platform [73].

Another significant outcome of the 5GTANGO project is the development of an SDK that allows: (i) the generation of descriptors, (ii) validation of the descriptors, (iii) emulation of the services and components, (iv) development of tests, and (v) benchmarking and analysis of

the performance of the services [74].

The V&V Platform and the SDK enabled a well-structured testing process, which vastly contributed to a better and more extensive validation of NFs, consequently reducing their time-to-market.

### Verification and Validation Process

The verification and validation process is composed of three stages: (i) structural validation of the descriptor, (ii) functional testing, and (iii) performance evaluation.

During the structural validation of the descriptors, four tests are performed: (i) syntax testing, where the descriptor is validated against the 5GTANGO schemas, (ii) integrity testing, which checks if all tags of the descriptor have the correct values, (iii) topology testing, where the network topology of the NSs is validated, and (iv) custom rules testing, that validates the descriptor against custom rules defined by the developer, and provided via a YAML file.

If the descriptors are structurally correct, the second testing phase may start. Along with the onboarding of the VNFs and NSs, the developer also onboards some test plans. These are YAML files with the definition of the tests that will be performed. The second phase starts with the deployment of the VNFs and NSs in an emulated environment. Then, the test plans are executed.

After the test execution, the third phase occurs, where several results and metrics are gathered and processed. After this, the outputs of the V&V process are displayed via a GUI.

### **2.5.3 5G EVE**

The 5G European Validation Platform for Extensive Trials, abbreviated to 5G EVE, is a European project that started in July 2018, aiming to implement and test 5G infrastructures in Europe. To do so, during this project's scope, several European 5G sites will be interconnected to create a unique 5G E2E facility [68]. According to 5G EVE's goals, this unique facility will also live outside the project's scope, being made available to other EU-funded projects [75].

To achieve such ambitious goals, 5G EVE presents an extended and complex architecture. Its main components are (i) the Portal, (ii) the Interworking Layer, and (iii) the Site-Specific Layer. Fig. 2.16 depicts 5G EVE's architecture [75].

To validate 5G services, 5G EVE focuses on metrics and KPIs. Regarding metrics, 5G EVE assumes the existence of application and infrastructure metrics. The first are metrics generated by the vertical services, whereas the second refer to the underlying infrastructure where the services are deployed. KPIs are used to validate if specific criteria are achieved and are applied on the collected metrics. For instance, a KPI might be a threshold for a specific metric [75].

### Testing and Validation Framework

On 5G EVE, the entity responsible for the testing and validating the 5G services is the Testing and Validation Framework. This framework follows the 5G EVE Testing and Validation Approach, depicted in Fig. 2.17. Its main components are: (i) the Experiment

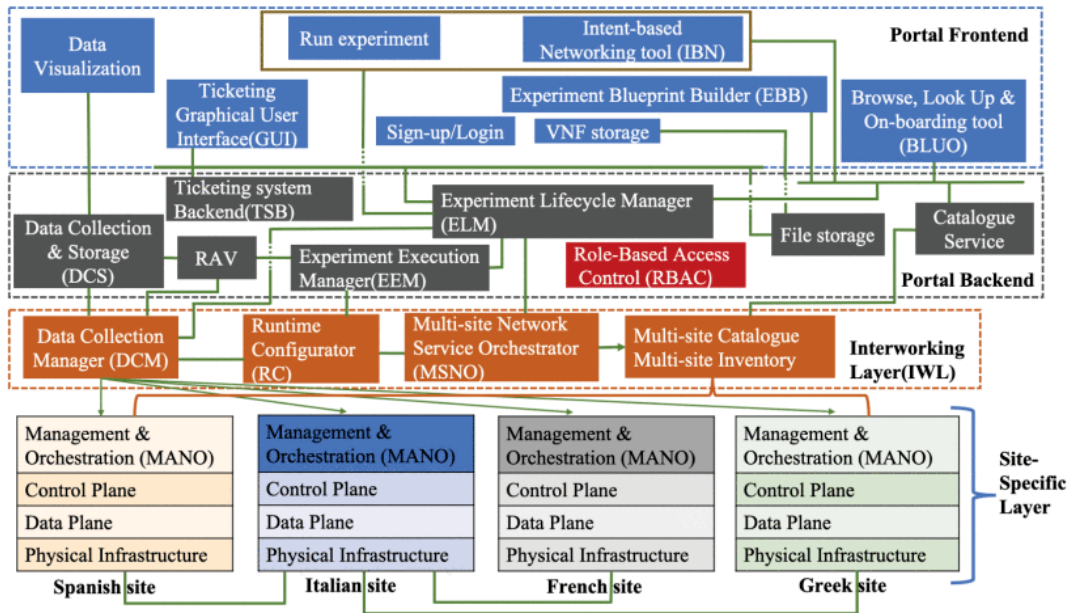


Figure 2.16: 5G EVE Platform Architecture [75].

Execution Manager (EEM), (ii) the Results Analysis and Validation (RAV) component, and (iii) the Performance Diagnosis (PD) module. Alongside these components, a Test Script Inventory stores the scripts associated with each test case [76] [77]. Fig. 2.18 presents the architecture of 5G EVE Testing and Validation Framework.

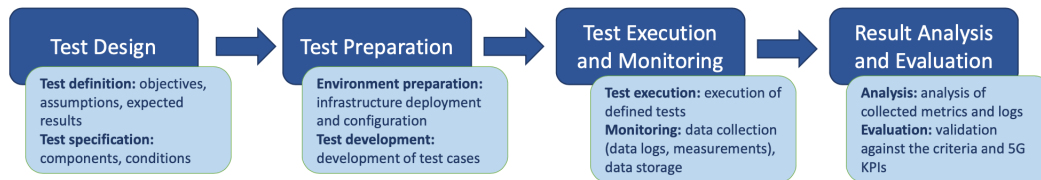


Figure 2.17: 5G EVE Testing and Validation Approach [76].

The EEM is responsible for the execution of the tests performed on the 5G services and has the following functionalities: (i) trigger day-2 configurations, (ii) trigger the execution of test cases that interact with the infrastructure components, (iii) trigger the validation of the test cases through RAV, and (iv) reset the infrastructure conditions to the ones before the test case execution [76].

The RAV module evaluates the obtained results, performing the following functions: (i) calculates KPIs from raw metrics, (ii) validates the test results, and (iii) generates a test report with the outcomes of the testing phase [76].

Lastly, the PD, is responsible for: (i) performing a deeper KPI analysis, (ii) performing anomaly detection on the collected metrics, (iii) performing Root Cause Analysis, and (iv) generating insights that allow the alleviation of the performance degradation [76].

### Testing Process

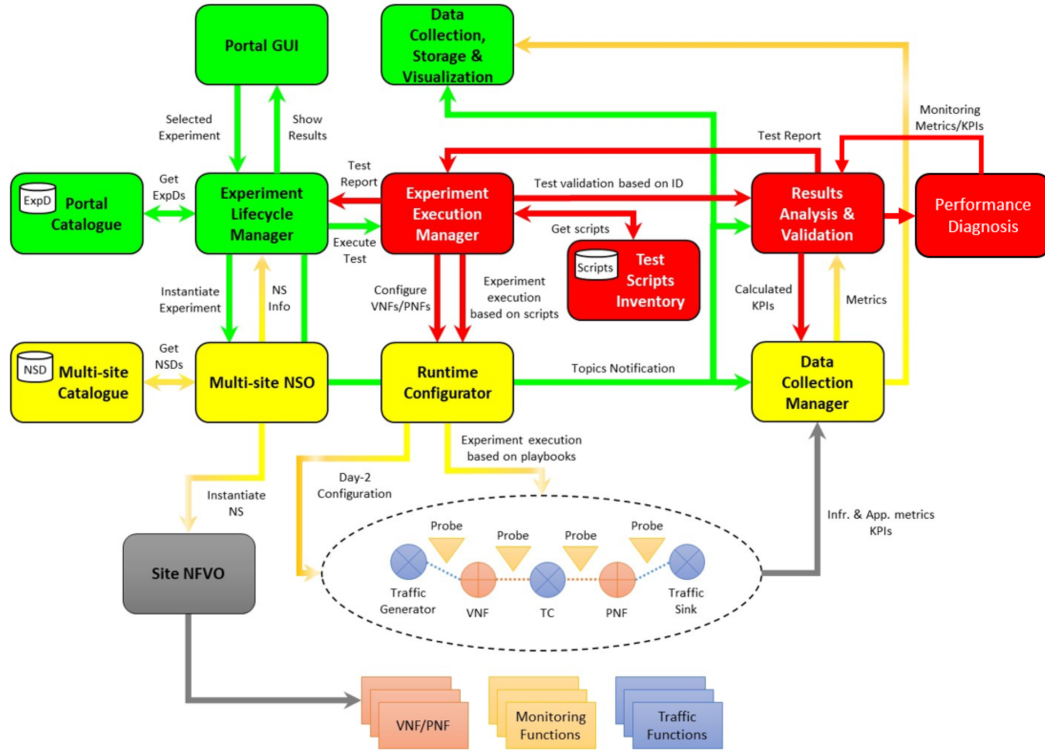


Figure 2.18: 5G EVE Testing and Validation Framework Architecture [76].

Creating a test experiment on the 5G EVE platform requires four different blueprints. These blueprints address the experiment’s information, ultimately defining it, and have to be designed as a joint effort between the experiment’s developer and the manager of the vertical where the experiment will take place [77].

These blueprints are the following [77]:

1. **Vertical Service Blueprint** - This blueprint defines the main features of the experiment, as (i) its components, (ii) the service’s connection endpoints, (iii) the virtual links associated with the connection points, (iv) the metrics that will be collected during the experiment, and (v) the facility where the experiment will take place;
2. **Context Blueprint** - This blueprint is used to generate network impairments, such as network delay and packet loss;
3. **Test Case Blueprint** - This blueprint is used to test the functionality of the 5G service;
4. **Experiment Blueprint** - This blueprint combines all the previously mentioned blueprints and defines the KPI thresholds to be used in the experiment.

To better understand the design of these blueprints, an annotated Test Case Blueprint is presented in Fig. 2.19, while Fig. 2.20 presents an annotated Experiment Blueprint.



Figure 2.19: 5G EVE Test Case Blueprinte. Adapted from [78].

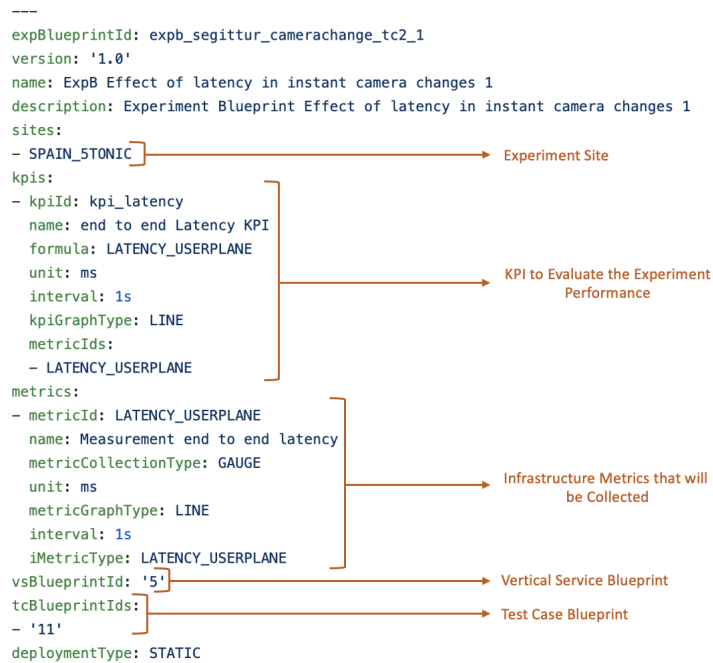


Figure 2.20: 5G EVE Experiment Blueprint Example. Adapted from [79].



### 2.5.4 5GinFIRE

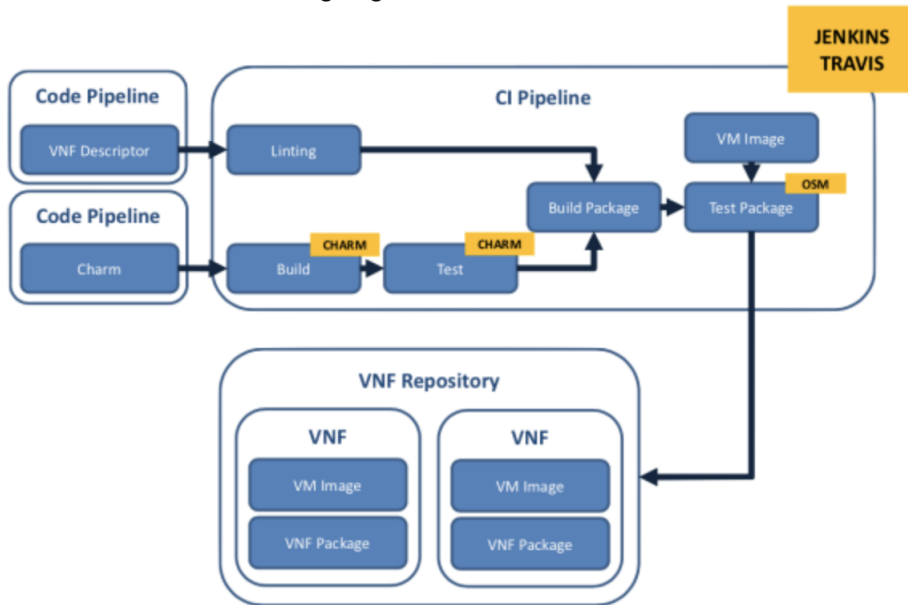
5GinFIRE is a Horizon 2020 project that aimed to create and operate an open and extensible 5G NFV-based ecosystem focused mainly on vertical industries [69]. This project started in January 2017 and ended in December 2019 [80].

It intended to create a distributed platform where new 5G components and architectures could be deployed and tested, before reaching production. To achieve this, 5GinFIRE relied on ETSI NFV standards and followed an open-source approach [81].

For this dissertation, the most crucial contribution of 5GinFIRE is its Continuous Integration Pipeline, which enabled some automation in the 5G services validation process. This validation and verification process is further described.

#### Automated Validation Process

The validation process is triggered once a VNF is uploaded to the 5GinFIRE Portal. During this process, two independent validations occur: (i) validation of the VNFD and (ii) validation of the VNF Juju Charms. Automation is achieved using Jenkins, a CI server that is responsible for managing the validation of the VNFDs and Charms and can even deploy the submitted VNF in a staging testbed, through OSM [82]. Fig. 2.21 illustrates this process.



**Figure 2.21:** Automated Validation of VNFs in 5GinFIRE [82].

On the onboarding of the VNF package, Jenkins configures a testing environment creating a Docker container from a Dockerfile containing all the needed configurations. This allows setting up an environment with all the tools necessary to run the tests and send the results back to the Portal successfully.

Jenkins Pipeline is composed of the following stages;

- Fetch the VNF Package that needed to be tested;
- Extract the VNFD;



- Syntactic, Semantic and Reference Testing;
- Obtain the results and send them to the 5GinFIRE Portal;
- Clean Jenkins workspace, removing the files that may have been created during the execution phase.

The first step of the testing process is to evaluate if the VNFD is a JavaScript Object Notation (JSON) or a YAML file, since the Release 5 of OSM, the MANO framework used in 5GinFIRE, only supports these files. Then, the content of the file is loaded, and the syntactic testing phase starts. During all the tests, it is necessary to iterate over all the tags, recursively getting new tags from the containers.

The syntactic testing phase ensures that all the tags used in the VNFD exist in the OSM IM, and are correctly placed in the descriptor. After this phase is completed, semantic validation takes place. This validation verifies if the content of each tag is according to the data types defined for that tag. If the tag is a container, the data type validation will not take place, and the validator moves on to the following tag.

During the last testing phase - reference testing - the path references are evaluated. If there are references to paths that do not exist, it is during this phase that those errors are identified.

While the syntactic and semantic tests are performed when navigating the VNFD, the reference validation can only be performed after the full navigation of the VNFD.

If the tests described previously are successful, the 5GinFIRE Portal is notified, and the VNF is deployed.

### 2.5.5 TMF Specifications

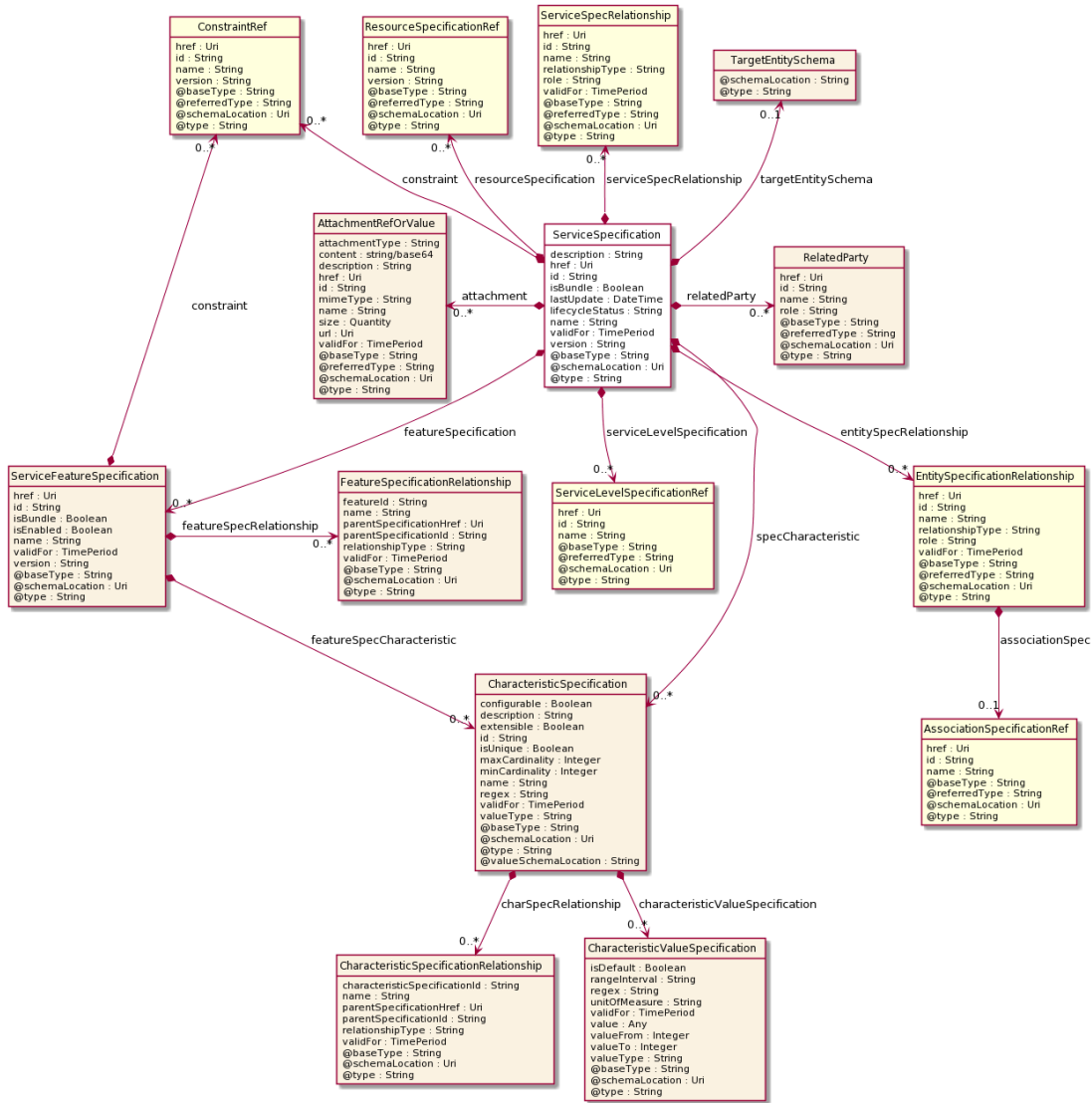
TMF is a global association that provides an open and collaborative environment to help its members in their digital transformation initiatives [83].

By signing the TMF Manifesto [83], the members commit to using and contributing to a suite of standard Open APIs, moving towards increasing efficiency and providing common interfaces that will simplify the integration of services and components from different vendors. Thus, reducing the costs, risk, and time-to-market associated with traditional integrations [83].

This initiative combats the usage of proprietary interfaces, which ultimately pose difficulties in advancing the current networking paradigm.

Until July of 2021, ninety-one of the world's leading telecommunication providers have signed the Open API Manifesto, and this number is expected to increase [83].

Openslice, presented in Section 2.5.1, follows several specifications from the TMF (for instance, TMF633 and TMF641), which highly increases the possibility of this project's expansion [71]. As an example of a TMF specification, Fig. 2.22 is presented, depicting the Class Diagram for the Service Specification Resource Class, presented in the TMF633 specification [84].



**Figure 2.22:** Class Diagram for the Service Specification Resource Class, defined by TMF633 [84].

## 2.6 WORK PROPOSAL

Having the background concepts being presented, a work proposal must be formulated. Thus, this section introduces the problems addressed by this dissertation and presents a work proposal to solve them.

As mentioned before, this dissertation's goals intimately follow the ones of the 5GASP Project. Thus, it is vital to understand the intended outcomes of 5GASP and their contribution to improve NFV.

5GASP aims to provide a fully automated and self-service testbed, which shortens the idea-to-market time. Besides, this testbed should provide Continuous Integration mechanisms that enable a faster and automatic validation of NetApps, hence increasing the network operators' trust in NetApps created by third-party developers.

To do so, 5GASP aims to validate NetApps and the underlying infrastructure where they

are deployed, through the creation of an automated pipeline that performs these validations.

Although the core of the validation pipeline resides on validating the behavior and performance of NetApps, a pre-flight validation must also be addressed. As the name states, this validation process occurs before the NetApps are deployed and mainly addresses the VNF and NS packages. This validation must approach aspects like: (i) the VNF and NS descriptors structure, (ii) the validation of the VNF Juju Charms, the security validation of the VNF Charms and Scripts, etc.

Thus, this dissertation addresses two problems: (i) the validation of the NetApp's packages and (ii) the validation of the NetApps' behavior through infrastructure, functional, and performance tests. Although the development of a NetApp Package Validator would be a very challenging and interesting outcome of this dissertation, unfortunately, this validator is a rather complex tool and, if fully developed, it might have to be the only focus of this document.

Since the main goal of this dissertation is the creation of validation mechanisms for the behavior and performance of NetApps, the development of a NetApp Package Validator will be considered a secondary goal of this dissertation, and only its Descriptors Validator module will be addressed in this document.

Consequently, this document will achieve two different outcomes: (i) a descriptor's validation mechanism and (ii) an automated pipeline to validate the NetApps, which is implemented according to a Continuous Integration paradigm.

Regarding the Descriptors Validator, it will validate the descriptors on their syntax, semantics, and internal references. Syntactical tests verify if all the descriptor's tags are correctly defined. Semantics tests validate if all the leaf values' data types are according to what is defined in the information models. Lastly, reference tests validate if the internal dependencies between descriptor elements are correct.

Given the fact that 5GASP will support two different MANO frameworks (ONAP and OSM), the Descriptors Validator will have to address, at least, TOSCA and YANG models. It is intended the validator can be extended to also support other models. Although, since the 5GASP project kicked off in January 2021, at the date of writing this dissertation, only OSM is being used as MANO, thus a stronger focus on validating OSM's VNF and NS descriptors.

On the other hand, the on-flight validation pipeline will have to address several validation scenarios, thus being as dynamic and modular as possible. This pipeline will be responsible for performing two different types of tests: (i) pre-defined tests, which will be available on all the individual testbeds, and developer-defined tests, which are specifically created to validate a NetApp. Besides, the validation pipeline must also allow the deployment of testing VNFs, which are part of the validation process. For instance, the deployment of a VNF that generates network noise. Besides this, the pipeline should also collect VNF-related metrics.

Chapter 3 addresses the requirements, architecture, and implementation of the Descriptors Validator, while Chapter 4 addresses the same topics regarding the testing and validation pipeline.

## 2.7 CHAPTER SUMMARY

This chapter starts by exposing the problems of the traditional network systems, in Section 2.1. As explained in this section, TSPs faced many problems with hardware-based systems, being the most relevant ones: (i) the lack of flexibility, (ii) difficulties in adapting existing systems to new purposes, and (iii) reduced scalability. These issues resulted in a higher time-to-market and higher CAPEX and OPEX costs, which ultimately lead to market losses.

Section 2.2 introduces a new networking paradigm - NFV. NFV decouples Network Functions from the physical equipment itself, boosting flexibility, efficiency, and scalability. NFV, in Europe, is standardized by ETSI, which created an ISG responsible for the research and guidelines establishment regarding it. All these efforts uplifted NFV, which became one of the most prominent technologies of 5G networks.

To enable an NFV paradigm, ETSI released MANO, a framework responsible for managing and orchestrating NFs, whose main components are the NFVO, VNFM, and VIM. MANO got widely adopted, and numerous implementations of this framework arose. Sections 2.3.1, 2.3.2, and 2.3.3 present the most successful ones, although OSM is the one that stands out. As a result, Section 2.3.3 presents an extended description of this initiative, addressing in detail OSM's architecture and Information Models.

OSM's architecture is highly modular, with several components and services performing different functions. On the other hand, OSM's IM is extensive and complex, offering an impressive collection of configurations, which provides NFV developers with tools to implement diverse NFs.

Sections 2.4.1, 2.4.2, and 2.4.5 introduce DevOps, Continuous Integration, and testing automation concepts, which were recently adopted in NFV environments. These concepts provide a smoother transition to NFV environments, reducing the time-to-market of new Network Services.

Section 2.5 presents the most relevant work developed in the context of this dissertation. Section 2.5.1 presents the 5G-VINNI European project, which resulted in the development of an OSS open-source NSaaS platform - Openslice. Section 2.5.2 presents 5GTANGO, which introduced an SDK to create and validate VNFs and NSs. This tool offered structural, functional, and performance validation of the NFs. Although, this validation process lacked automation, which poses a disadvantage when compared to the 5GinFIRE validation process. This European project, described in Section 2.5.4, presented an automatic validation pipeline for the VNF descriptors. When the VNFs were onboarded to the 5GinFIRE Portal, this pipeline would be triggered, performing syntactical, semantic, and references validation. The automation process in 5GinFIRE was powered by a Jenkins CI server that first configured a testing environment, using a Docker container, performed the mentioned tests, and then sent the results back to the Portal.

Besides these projects, 5G EVE is also addressed in this chapter. This project presented crucial contributions through its validation blueprints. The 5G EVE testing process was based on the following blueprints: (i) Vertical Service Blueprint, (ii) Context Blueprint, (iii) Test

Case Blueprint, and (iv) Experiment Blueprint. Section 2.5.3 explains in detail the role of each blueprint in the testing and validation process.

Section 2.5.5 presents the TMF Specifications, which introduce standardization for APIs. Through standardization, TMF aims to create universal Open APIs, used mainly by Telecommunication Providers, to ensure a faster and easier integration of network components and services.

Lastly, in Section 2.6, a work proposal is presented. This work proposal will guide all the development process, and presents the two outcomes of this dissertation: (i) a descriptors validation mechanism and (ii) an automated pipeline to validate the NetApps.



# Descriptors Validator Service

Section 2.6 presented the work proposal for this dissertation. In that chapter, two different goals were defined: (i) the creation of a mechanism to validate VNF and NS descriptors and (ii) the creation of CI/CD Service to validate NetApps.

This chapter aims to define and present the architecture and implementation of the Descriptors Validator Service. Before addressing these topics, some contextualization is needed. Thus, Section 3.1 presents the requirements of the Descriptors Validator Service.

After the requirements and specification definition, the next phase is to define the architecture of this Service. Since this Service is composed of several independent modules, in Section 3.2, which presents the architecture of the Descriptors Validator Service, each module is addressed independently.

With the architecture defined, it is possible to move on to the implementation, described in Section 3.3. Once again, each component is addressed individually. To further elucidate the reader, several code blocks and figures are presented. With this, it is expected that the reader is able to fully understand how the Descriptors Validator Service was designed and implemented.

## 3.1 PROBLEM STATEMENT AND REQUIREMENTS

To validate VNFs and NSs two types of tests can be addressed: (i) structural tests, where the descriptors are validated, and (ii) functional tests, which validate their behavior. The latest are highly complex, thus the creation of a testing pipeline to fulfill this validation.

When performing a structural validation of VNFs, several aspects must be approached: (i) the structural validation of the descriptor, (ii) validation of the charms packaged with the VNFs, (iii) the existence of the referenced VM images, and so many other. Although, in the scope of the 5GASP Project, the validation of the charms and of the referenced VM images is already addressed in the NetApp Onboarding and Deployment Service (NODS). Thus, this dissertation will mainly address the validation of the VNFs and NSs descriptors.

```

typedef scaling-trigger {
    type enumeration {
        enum pre-scale-in {
            value 1;
        }
        enum post-scale-in {
            value 2;
        }
        enum pre-scale-out {
            value 3;
        }
        enum post-scale-out {
            value 4;
        }
    }
}

typedef scaling-policy-type {
    type enumeration {
        enum manual {
            value 1;
        }
        enum automatic {
            value 2;
        }
    }
}

```

**Code Block 2:** *Enums* example in OSM.

To validate the structure of these descriptors, syntactical, semantical, and reference tests have to be performed. The syntactic tests must validate if all the tags of the descriptors are correctly defined. To achieve this, the name of each tag must be correctly defined according to the Information Models. The semantic tests, on the other hand, validate the content of each tag. A tag can have a limited data type, such as *string*, *integer*, or *float*, or a specific range of accepted values. When occurring the last case, the accepted values are defined in the Information Models. Code Block 2, presents some of the *enums* defined in OSM's IMs.

Finally, the reference tests validate the dependencies between a descriptor's tags. Since a tag can reference another tag, the Descriptors Validator Service must verify if the referenced tag is defined in the descriptor and if the defined value is according to the one of the referenced tag.

The validation of the descriptors also poses another complex problem: different MANO frameworks may use different IMs. This happens, for instance, in ONAP and OSM, the addressed MANO frameworks in 5GASP. While ONAP relies on TOSCA Information Models, OSM relies on YANG Information Models.

To cope with this, two alternatives arise: (i) create a different validator for each Information Model or (ii) create a singular structure that maps all the different Information Models and use this structure in the syntactical, semantical, and reference validation.

When considering modularity and decoupling, the second alternative presents the best approach, thus being the one chosen to implement the Descriptors Validator Service. This



way, the Descriptors Validator Service will be mainly composed of two individual modules. The first is the Information Models Parser, and the second is the actual Descriptors Validator module.

Besides, another module can also be implemented. More than validating the descriptors, it is beneficial if correction suggestions are provided. Thus, a Correction Suggestions module will also be created. This module will provide recommendations based on previous valid descriptors. When reference errors occur, this approach does not solve the problem at hand, thus, in this scenario, the suggestion will reside in the value of the referenced tags.

## 3.2 ARCHITECTURE AND SPECIFICATIONS

The Descriptors Validator Service will be composed of three individual modules: (i) the Information Models Parser module, (ii) the Descriptors Validator module, and the Correction Suggestions module. Together, these modules create a pipeline that validates VNF and NS descriptors.

### 3.2.1 Information Models Parser Module

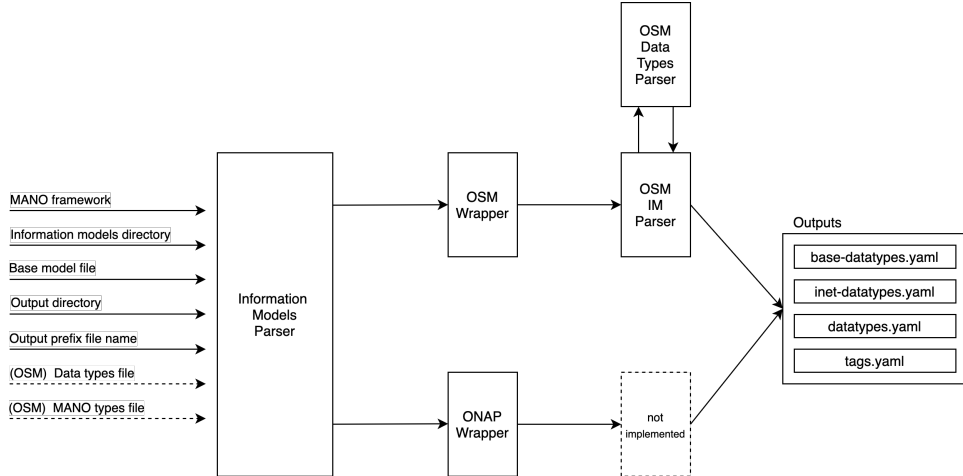
The Information Models Parser module aims to parse different information modules (YANG, or TOSCA, for instance) to a singular data structure, which the Descriptors Validator module will then consume. This module comprises two wrappers: (i) OSM's Wrapper and (ii) ONAP's Wrapper, but more wrappers can be added since it is highly modular and decoupled.

The Information Models Parser module receives as input: (i) the MANO framework relative to the Information Models to be parsed, (ii) the directory containing the Information Models, (iii) the specific model file that should be parsed, (iv) the directory where the outputted files should be stored, and (v) the prefix that should be added to the name of each outputted file.

Since 5GASP only uses OSM as the main MANO framework at the time of writing this dissertation, only its wrapper was implemented. This wrapper required two more inputs: (i) the file containing the OSM's MANO types specification, and (ii) the file containing OSM's data types.

Regarding the outputs of the Information Models Parser module, using the OSM's Wrapper, four files will be created: (i) a file containing the description of each data type, (ii) a file containing the inet *enumerables*, (iii) a file with the *enumerables* regarding specific OSM's tags, and (iii) a file with the description of all the tags of the Information Model. The last presents the following information: (i) tag name, (ii) tag path, (iii) tag data types, (iv) tag accepted values, (v) tag type (list, container, leaf, etc), (vi) reference path, if the tag is supposed to reference another tag, and (vii) additional information.

Fig. 3.1 presents an architectural diagram of this module.



**Figure 3.1:** Information Models Parser Module's Architecture.

### 3.2.2 Descriptors Validator Module

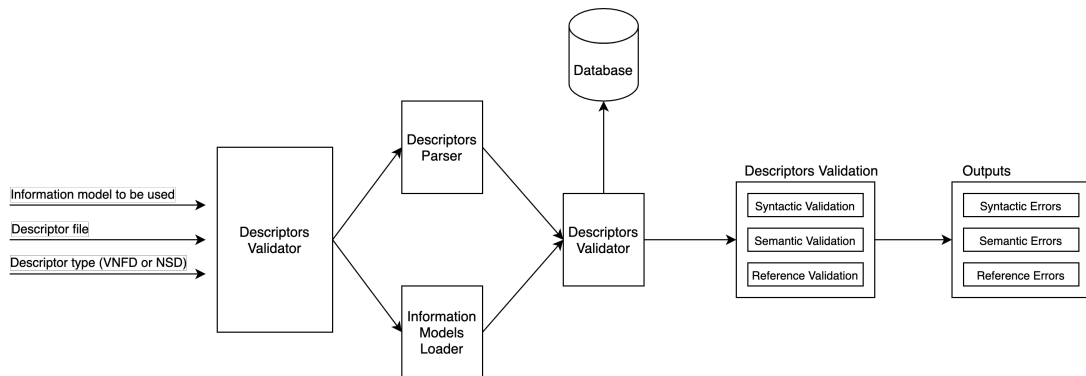
The Descriptors Validator module is responsible for validating a VNF or NS descriptor, using the outputs of the Information Models Parser module.

This module requires the following inputs: (i) the descriptor file to be validated, (ii) the type of the descriptor (VNFD or NSD), and (iii) which IMs should be used to validate the descriptor. Since the parsed Information Models are already loaded in the Descriptors Validator module, when running it, it is only needed to provide the Information Model key as input and not the directory containing all the outputs of the Information Models Parser module.

After validating the descriptors, this module will output a list of all the errors found, indexed by the type of error - syntactic, semantical, or reference. This information will then be passed to the Correction Suggestions module.

To allow context-based suggestions, this module uses a database to store descriptors-related data. Everytime a VNF or NS descriptor is validated, the Descriptors Validator module stores its tags, and their value, in the database, which will provide some context for the Correction Suggestions module.

Fig. 3.2 presents the high-level architecture of the Descriptors Validator module.



**Figure 3.2:** Descriptors Validator Module's Architecture.

### 3.2.3 Correction Suggestions Module

As mentioned before, the Correction Suggestions module receives as input a list of the errors found in a descriptor, indexed by the type of error - syntactic, semantical, or reference.

The process behind generating suggestions differs for each type of error. For syntactic errors, this module will compute the Jaro Distance to all the tags described in the Information Models, suggesting the three tags with the lowest distance to wrongly defined tag. For semantic errors, this module will query the Descriptors Validator database and return the three most used values for a specific tag. Lastly, for reference errors, if the referenced tag exists, the Correction Suggestions module will suggest the value defined for that tag.

The output of this module is an enhancement of the list of the errors it received as input, where on each error will be added a list of correction suggestions.

Fig. 3.3 presents the high-level architecture of the Correction Suggestions module.

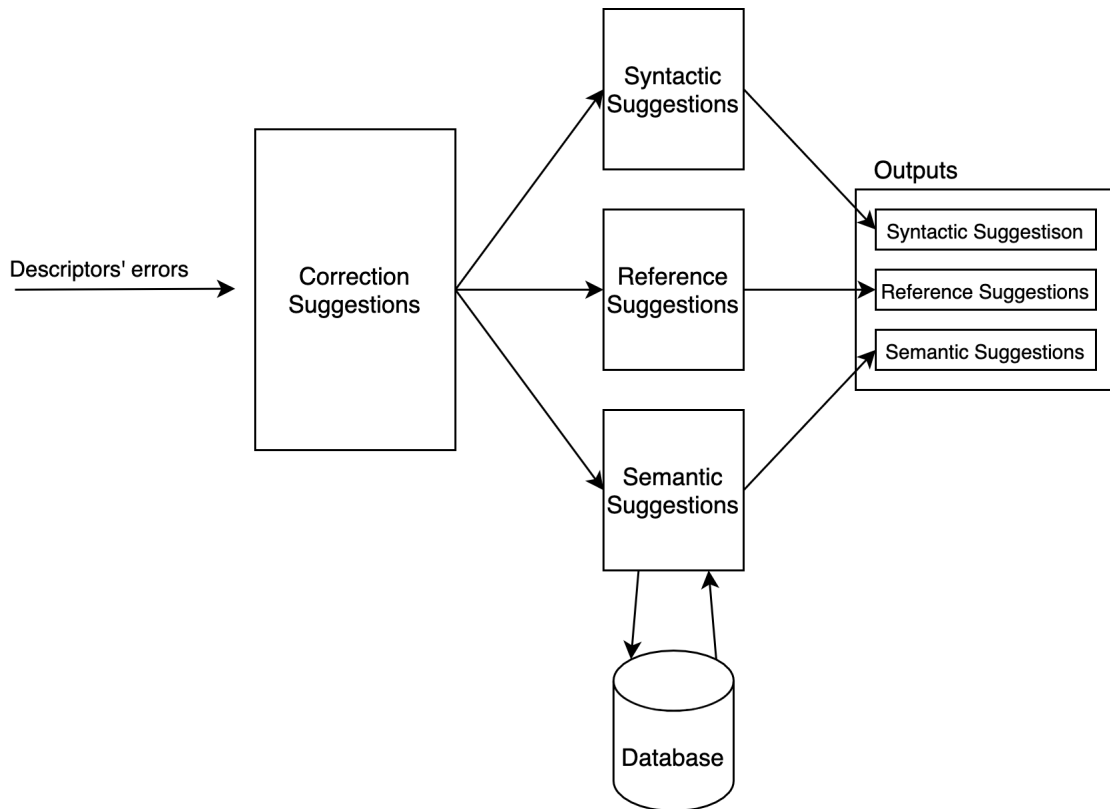


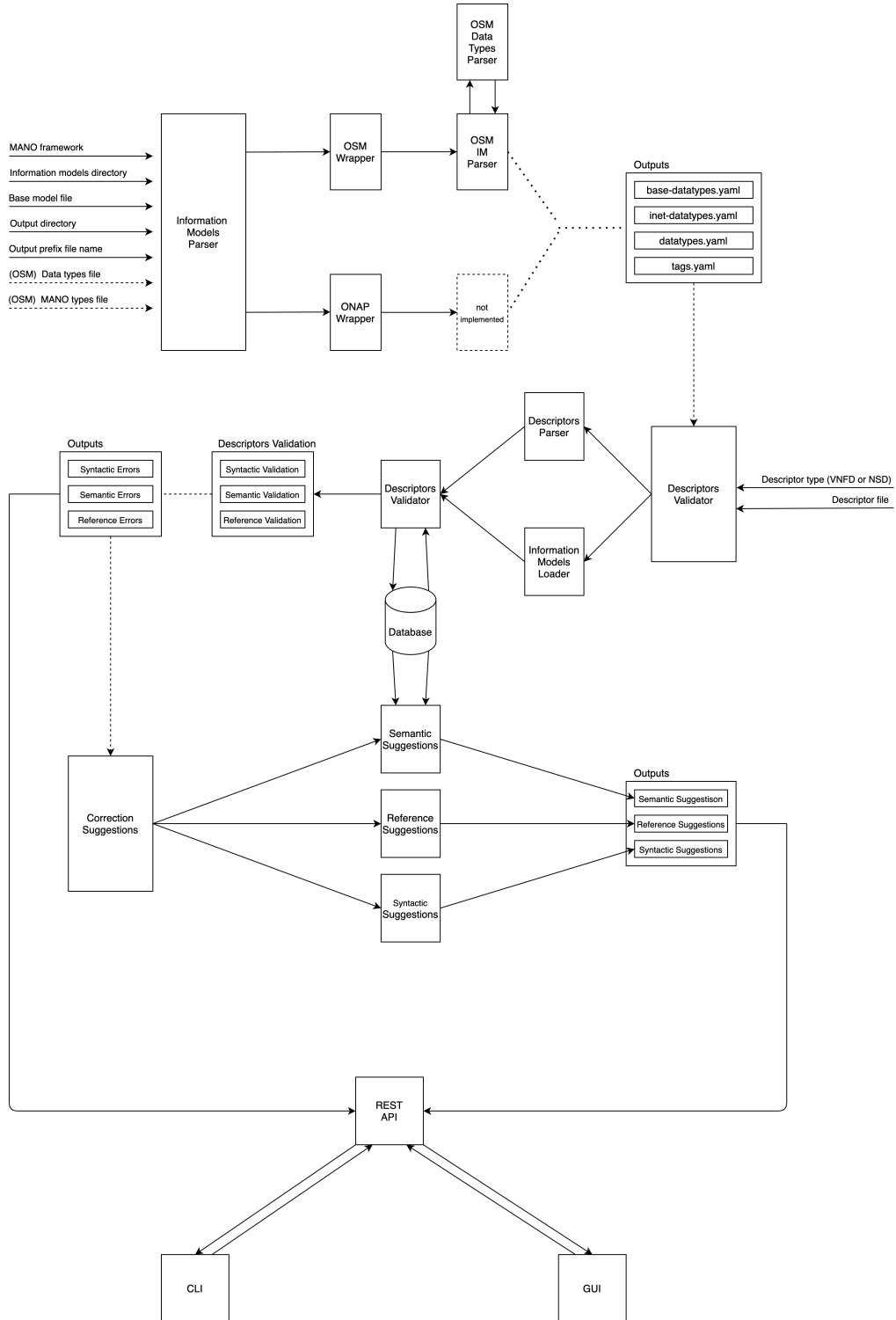
Figure 3.3: Correction Suggestions Module's Architecture.

### 3.2.4 REST API and GUI

To make the Descriptor Validator Service available for NetApps' developers, an API and a GUI were implemented. Besides, this program is also available through a Command-Line Interface (CLI) that makes requests to the API.

The GUI is a simple HTML Bootstrap website that, such as the CLI, makes requests to the API.

Having all the modules and components of the Descriptors Validator Service being described, and their architecture presented, it still is missing to present the overall architecture for this service. Thus, in Fig. 3.4 is possible to observe the complete Descriptors Validator Service architecture.



**Figure 3.4:** Descriptors Validator Service Complete Architecture.

### 3.3 IMPLEMENTATION

#### 3.3.1 Information Models Parser Module

The goal of this module is to represent different Information Models through a singular and global data structure. Thus, it was defined that the model would be parsed and their content converted and stored in YAML files. These files allow the validation of VNF and NS descriptors without having different validators for each MANO framework, since they present a layer of abstraction that enables the existence of a single Descriptors Validator module.

Since 5GASP, until the time of writing this dissertation, only addressed OSM as their MANO framework, only the OSM Wrapper was implemented. Although, in the future, it is expected that 5GASP also supports ONAP, which will lead to the implementation of the ONAP Wrapper, that will parse TOSCA Information Models.

OSM's Information Models are described using YANG, a data modeling language. This language differs from traditional structural languages, such as YAML and JSON. Thus, the complexity in directly converting YANG to one of these formats. Besides, the possibility of YANG modules referencing and being augmented by other modules also contributes to this complexity since a module would have to be rendered before parsing it to JSON or YAML.

Due to this complexity, there is still no solution that can perfectly parse YANG modules to YAML or JSON, which is the reason why the Information Models Parser module was implemented.

Since, as stated before, YANG modules can be augmented by other modules, before starting to parse these models, it is needed to render them. To do so, `pyang`<sup>1</sup> was used. `Pyang` is a YANG validator and parser written in Python [85]. One of the features provided by this tool is the possibility of generating trees that map the YANG modules' information. Besides being easier to interpret the models' information, when mapped in a tree, these information trees are also simpler to parse. Due to this, the first stage of the Information Models Parser is the generation of information trees. Code Block 3 presents an example of a tree mapping a VNF related information model.

The next stage is the parsing of the file containing the information tree to a tags tree. To do so, a `Tag` class was created. Code Block 4 shows the structure of this Python class.

An OSM tag can be: (i) a container, (ii) a leaf, (iii) a list, (iv) a choice, or an (v) option. This information is stored in the parameter `tag_type`. The `reference_path` parameter is only relevant if the value of the tag that is being parsed references another tag. If so, this parameter stores the path of the referenced tag. `optional_info`, as the name states, stores optional information, and can be used to store all the information that is not directly mapped to the other tag parameters, for instance, the key of a tag that is of type `list`. The other parameters are self-explanatory, so they won't be described in detail.

If a tag's value is an enumeration, it is needed to gather the optional values directly from the YANG models since the tree file does not provide this information.

---

<sup>1</sup><https://github.com/mbj4668/pyang>

```

module: vnfd
+--rw vnfd-catalog
  +--rw schema-version?  string
  +--rw vnfd* [id]
    +--rw id              string
    +--rw name            string
    +--rw short-name?    string
    +--rw vendor?        string
    +--rw logo?          string
    +--rw description?   string
    +--rw version?       string
    +--rw vnf-configuration
      | +--rw (config-method)?
      | | +--:(script)
      | | | +--rw script
      | | |   +--rw script-type?  enumeration
      | | +--:(juju)
      | | | +--rw juju
      | | |   +--rw charm?  string
      | | |   +--rw proxy?  boolean
      | | |   +--rw cloud?  enumeration

```

**Code Block 3:** OSM 8 - Information Tree.

```

class Tag:
  def __init__(self, name=None, tag_type=None, path=None, datatype=None, mandatory=None,
    possible_values=None, reference_path=None, optional_info=None):
    self.name = name
    self.tag_type = tag_type
    self.path = path
    self.datatype = datatype
    self.mandatory = mandatory
    self.possible_values = possible_values
    self.reference_path = reference_path
    self.optional_info = optional_info

```

**Code Block 4:** *Tag* Class.

The parsing of the information model tree is very simple. As it is possible to observe in Code Block 3, each tag in the tree file follows the same structure: `<tag_name><tag_data_type>`. The tag name can be succeeded by a '?', which indicates that the tag is optional. On the other hand, if the tag maps a list, the tag name is succeeded by an '\*' and the list id. This id is used to assure there are no repeated elements in the list.

Besides the parsing of the Information Models tree, the OSM Wrapper also parses the accepted data types. This information is stored in YAML files that, besides listing the accepted data types, also provide information on how to validate them, for instance, through regexes.

The whole parsing process is encapsulated by the `parse_model` function, presented in Code Block 5.

The complete output of this module is a collection of 4 different files: (i), `tags.yaml`, (ii) `datatypes.yaml`, (iii) `base-datatypes.yaml`, and (iv) `inet-datatypes.yaml`. Code Blocks 6, 7, 8, 9 present a portion of each of these files, respectively.

```

def parse_model(self):
    # generate yang tree and load it
    yang_tree_generator = YANG_Tree_Generator(self.osm_model_directory_path,
        self.osm_model_yang_filename)
    self.osm_tree_filepath = yang_tree_generator.generate_tree()

    # parse datatypes
    self.osm_datatypes_parser = OSM_Datatypes_Parser(self.osm_tree_filepath,
        self.osm_base_model_filepath, self.osm_model_mano_datatypes_filepath)
    self.osm_base_datatypes, self.osm_inet_datatypes = self.__read_osm_datatypes()

    # create tags dict
    self.tags_dic = {}

    return (
        self.__parse_im_tree_data(),
        self.__parse_tree_datatypes(),
        self.osm_base_datatypes,
        self.osm_inet_datatypes
    )

```

Code Block 5: *Parse\_model* Function.

```

tags:
  vnfd:vnfd-catalog:
    datatype: null
    mandatory: null
    name: vnfd-catalog
    optional_info: {}
    path:
      - vnfd-catalog
    possible_values: null
    reference_path: null
    tag_type: CONTAINER
  vnfd:vnfd-catalog/schema-version:
    datatype: STRING
    mandatory: false
    name: schema-version
    optional_info: {}
    path:
      - vnfd-catalog
      - schema-version
    possible_values: null
    reference_path: null
    tag_type: LEAF
  vnfd:vnfd-catalog/vnfd:
    datatype: null
    mandatory: null
    name: vnfd
    optional_info:
      list_key: '[id]'
    path:
      - vnfd-catalog
      - vnfd
    ...

```

Code Block 6: Portion of the *tags.yaml* File.

```

AGGREGATION_TYPE:
  options:
    - AVERAGE
    - MINIMUM
    - MAXIMUM
    - COUNT
    - SUM
  type: aggregation-type
ALARM_SEVERITY_TYPE:
  options:
    - LOW
    - MODERATE
    - CRITICAL
  type: alarm-severity-type
ALARM_STATISTIC_TYPE:
  options:
    - AVERAGE
    - MINIMUM
    - MAXIMUM
    - COUNT
    - SUM
  type: alarm-statistic-type
CLOUD:
  options:
    - lxd
    - k8s
    - lxd
    - k8s
    - aws
    - cloudsim
    - cloudsim_proxy
    - mock
    - openmano
    - openstack
    - vsphere
    - openvim
  type: cloud
...

```

Code Block 7: Portion of the *datatypes.yaml* File.

```

AGGREGATION_TYPE:
BOOLEAN:
  options:
    - 'True'
    - 'False'
    - 'true'
    - 'false'
  type: boolean
DECIMAL64:
  options:
    - ''
  type: decimal64
STRING:
  options:
    - ''
  regex_validation: '[a-zA-Z][a-zA-Z0-9\-\_\.]*|\.\.\.|[\^xX]\. *|[\^mM]\. *|[\^lL]\. *'
  type: string
...

```

Code Block 8: Portion of the *base-datatypes.yaml* File.



```

IPV6_PREFIX:
  options:
  - ''
  regex_validation: ((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}((([0-9a-fA-F]{0,4})?
(:|[0-9a-fA-F]{0,4}))|(((25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])\.){3}
(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])))/((([0-9])|([0-9]{2})|(1[0-1]
[0-9])|(12[0-8])))|((([:]+:){6}((([:]+:[:]+)|(.*\..*)))|((([:]+:)*
[:]+)?::(([:]+:)*[:]+)?)/.+)
  type: ipv6_prefix
IP_ADDRESS:
  options:
  - ''
  regex_validation: ((([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|
1[0-9][0-9]|2[0-4][0-9]|25[0-5])|(\%[\p{N}\p{L}]+)?|((:|[0-9a-fA-F]{0,4}):)
([0-9a-fA-F]{0,4}:){0,5}((([0-9a-fA-F]{0,4})?:)(:|[0-9a-fA-F]{0,4}))|
(((25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|
[01]?[0-9]?[0-9]))|(\%[\p{N}\p{L}]+)?|((([:]+:){6}((([:]+:[:]+)|(.*\..*)))|
((([:]+:)*[:]+)?::(([:]+:)*[:]+)?)(\%.+)?
  type: ip_address
IP_ADDRESS_NO_ZONE:
  options:
  - ''
  regex_validation: '[0-9\.\.]*|[0-9a-fA-F:\.]*'
  type: ip_address_no_zone
IP_PREFIX:
  options:
  - ''
  regex_validation: ((([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|
1[0-9][0-9]|2[0-4][0-9]|25[0-5])/((([0-9])|([1-2][0-9])|(3[0-2])))|
((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}((([0-9a-fA-F]{0,4})?:)(:|
[0-9a-fA-F]{0,4}))|(((25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])\.){3}(25[0-5]|
2[0-4][0-9]|[01]?[0-9]?[0-9])))/((([0-9])|([0-9]{2})|(1[0-1][0-9])|
(12[0-8])))|((([:]+:){6}((([:]+:[:]+)|(.*\..*)))|((([:]+:)*[:]+)?::(([:]+:)*[:]+)?)/.+)
  type: ip_prefix

```

**Code Block 9:** Portion of the *inet-datatypes.yaml* File.

These files are all stored in a single directory that will be used as input for the Descriptors Validator module.

### 3.3.2 Descriptors Validator Module

The Descriptors Validator module uses the outputs of the Information Models Parser module to validate a descriptor. The MANO agnostic Information Models are already loaded in this module, so, as input, besides the VNF or NS descriptor and type of descriptor (VNFD/NSD), it must receive the Information Models' version to access the pre-loaded models.

The first stage of validating a descriptor is to verify (i) if the descriptor provided follows a YAML or JSON structure and (ii) if the selected information model is available. The information model used to validate the descriptor is filtered using two parameters: (i) the information model version, for instance: *osm\_7*, *osm\_8*, or *osm\_9*, and (ii) the descriptor's type - VNFD or NSD.

After the pre-flight validations, the descriptors' validation process may start. The first step is to load the descriptor's tags as instances of a *Descriptor\_Tag* class. Code Block 10

```

class Descriptor_Tag:
    def __init__(self, tag_type, path, value=None):
        self.tag_type = tag_type
        self.path = path
        self.value = value

    def __str__(self):
        return "Tag={tag_type={}, path={}, value={}}".format(self.tag_type, self.path,
            self.value)

```

**Code Block 10:** *Descriptor\_Tag* Class.

```

Starting Validation for model osm_8 ...
Syntactic Test Results:
  Invalid Tag vnfd:vnfd-catalog/vnfd/connectionpoint.
  Invalid Tag vnfd:vnfd-catalog/vnfd/connectionpoint/name.
Semantic Test Results:
Can't perform semantic validation. The VNFD did not pass the syntactic validation!
Reference Test Results:
Can't perform reference validation. The VNFD did not pass the syntactic validation!
Ended Validation for model osm_8.

```

**Code Block 11:** Example of a Unsuccessful Syntactic Validation.

presents the structure of this class.

Since each tag's path is unique, the tags are stored in a dictionary, indexed by the tag's path. After this, the Information Models are loaded into memory, and the descriptor validation process can start.

The first stage of the descriptors validation is the syntactic validation. To do so, it is needed to verify if the descriptor's tags exist in the information model, which is achieved by a lookup in the dictionary that maps the information model's tags.

If a descriptor's tag does not exist in the model's tag dictionary, than the tag is wrongly defined. After all the tags are validated, the validation phase stops if errors are encountered since it would be impossible to validate the data type for the wrongly defined tags. Code Block 11, presents an example of a syntactic validation where errors were found.

If all the tags are correct, then the Descriptors Validator module can proceed to perform the semantic validation. This phase is the most complex one since the values of the tag differ a lot.

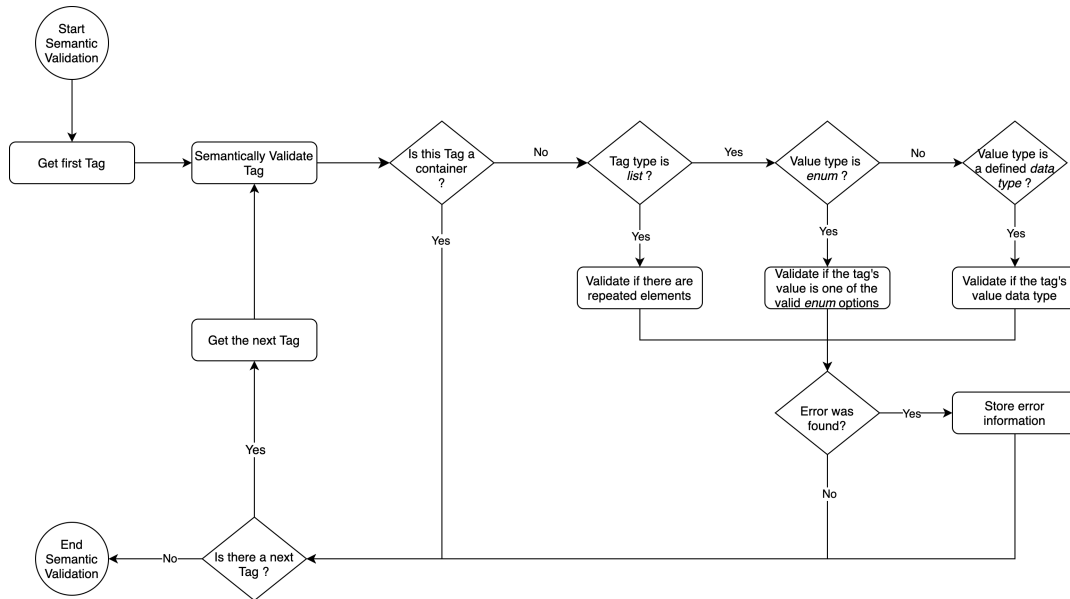
If the tag is defined as a reference one, it will not be semantically validated. On the other hand, if the tag's value is an enumerable, it must be verified if the tag's value is one of the possible values defined for that enumerable. If this value has a defined data type, such as a string or an integer, this value is casted to the defined data type, and if this operation is successful, the value is correctly defined.

Some data types are defined via a regex, so, in this scenario, besides the casting process, it is also needed to validate if the tag's value is according to the validation regex. Lastly, if the tag's type is *list*, the semantic validation phase verifies if there are no repeated elements.

Code Block 12, presents an example of a semantic validation where errors were found, and Fig. 3.5 presents the semantic validation flow.

Starting Validation for model osm\_8 ...  
 Syntactic Test Results:  
 All Syntactic tests passed!  
 Semantic Test Results:  
 The value '1a' on vnf:vnfd-catalog/vnfd/vdu/vm-flavor/vcpu-count doesn't match it's datatype (uint16).  
 Reference Test Results:  
 All Reference tests passed!  
 Ended Validation for model osm\_8.

**Code Block 12:** Example of an Unsuccessful Semantic Validation.



**Figure 3.5:** Semantic Validation Flow.

Starting Validation for model osm\_8 ...  
 Syntactic Test Results:  
 All Syntactic tests passed!  
 Semantic Test Results:  
 All Semantic tests passed!  
 Reference Test Results:  
 The value 'vnf-cp1' on vnf:vnfd-catalog/vnfd/vdu/interface/external-connection-point-ref doesn't match it's reference (vnf:vnfd-catalog/vnfd/connection-point/name) value of 'vnf-cp0'  
 Ended Validation for model osm\_8.

**Code Block 13:** Example of an Unsuccessful Reference Validation.

Finally, the reference validation phase occurs. For a reference tag to be valid, two requirements must be achieved: (i) the referenced tag must be defined in the descriptor and (ii) the value of the tag that is being validated must match the referenced tag's value.

Code Block 13, presents an example of a reference validation where errors were found, and Fig. 3.6 presents the reference validation flow. A complete validation workflow is also presented in Fig. 3.8.

If the descriptor is valid, its tags and values must be stored in a database so that this

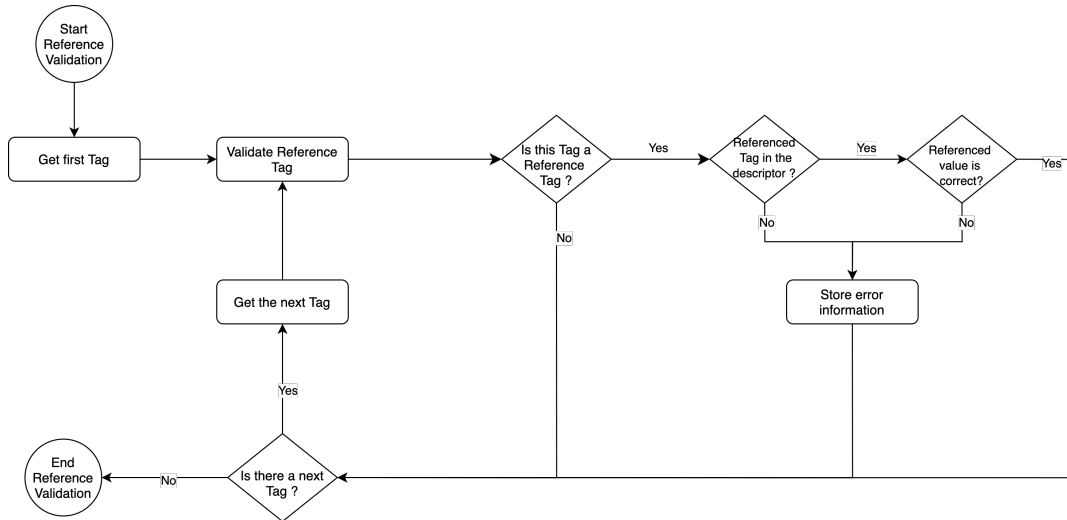


Figure 3.6: Reference Validation Flow.

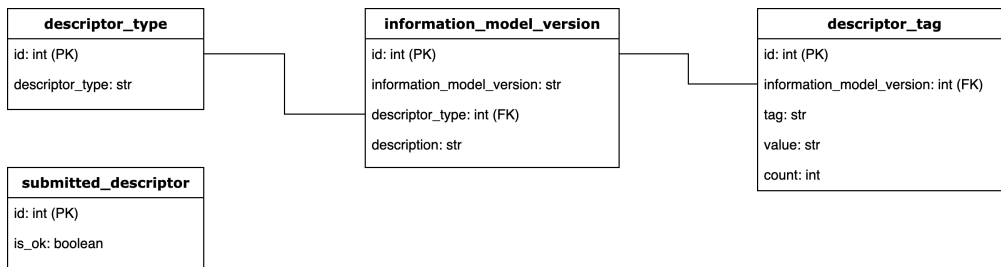


Figure 3.7: Descriptors Validator Database Schema.

information can be used in the Correction Suggestions module. The Descriptors Validator Service uses a PostgreSQL<sup>2</sup> database to store the number of times a specific value was defined for a tag. Its schema is presented in Fig. 3.7.

After the validation of a descriptor, a listing of errors, in form of a dictionary, is created. This is the output of this module and will be the primary input of the Correction Suggestions module. Code Block 14, presents an example of the errors dictionary.

```

{
  "syntactic_errors": [],
  "semantic_errors": [
    {
      "incorrect_value": "'1a'",
      "incorrect_tag": "vnfd:vnfd-catalog/vnfd/vdu/vm-flavor/vcpu-count",
      "validation_regex": None,
      "valid_datatype": "uint16",
      "possible_values": None,
      "message": "The value '1a' on vnfd:vnfd-catalog/vnfd/vdu/vm-flavor/vcpu-count
        ↪ doesn't match it's datatype (uint16).",
    }
  ],
  "reference_errors": [

```

<sup>2</sup><https://www.postgresql.org/>

```

{
  "incorrect_value": "vnf-cp1",
  "incorrect_tag": [
    "vnfd:vnfd-catalog",
    "vnfd",
    "vdu",
    "interface",
    "external-connection-point-ref",
  ],
  "referenced_tag": "vnfd:vnfd-catalog/vnfd/connection-point/name",
  "possible_values": ["vnf-cp0"],
  "message": "The value 'vnf-cp1' on
  ⇨ vnfd:vnfd-catalog/vnfd/vdu/interface/external-connection-point-ref doesn't
  ⇨ match it's reference (vnfd:vnfd-catalog/vnfd/connection-point/name) value of
  ⇨ vnf-cp0",
}
],
}

```

**Code Block 14:** Example of an Output of the Descriptors Validator module.

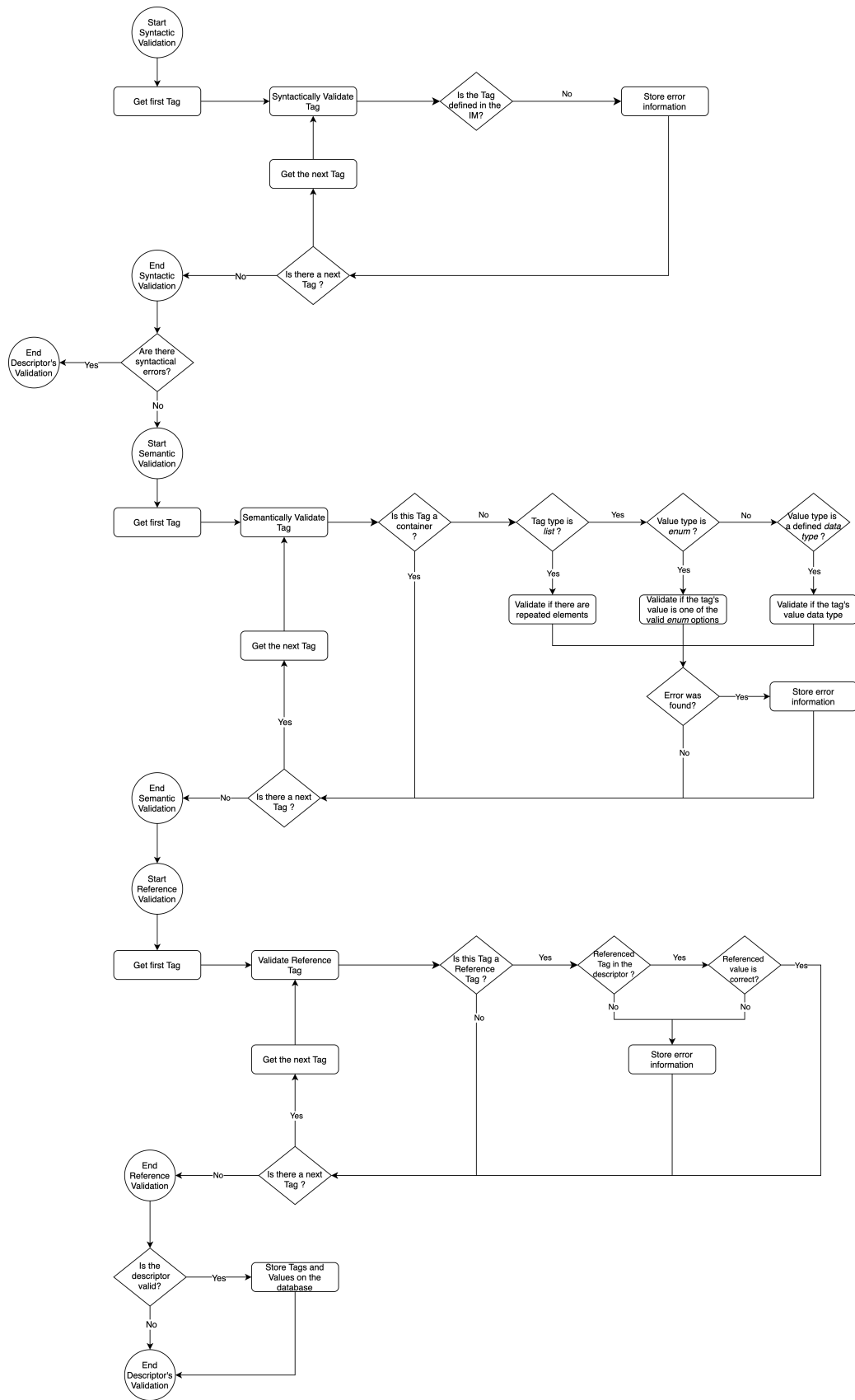


Figure 3.8: Complete Descriptors Validation Flow.

### 3.3.3 Correction Suggestions Module

The Correction Suggestions module receives as input (i) the Information Models used to validate the descriptor and (ii) the errors outputted by the Descriptors Validator module.

There are three functions involved in the correction suggestions generation. Each function generates suggestions for a type of error.

The *syntactic\_suggestions* function, presented in Code Block 15, uses the Jaro distance to compute the distance between a wrongly defined tag and all the tags defined in the Information Models. Then, it returns the three tags of the information model with the highest similarity.

```
def syntactic_suggestions(self):
    suggestions = []
    for error in self.syntactic_errors:
        distances = []
        for tag in self.im_info['tags']["tags_ordered"]:
            distance = jellyfish.jaro_distance(u'{}'.format(error["incorrect_tag"]),
            ↪ u'{}'.format(tag))
            distances.append((tag, distance))

        distances = sorted(distances, key=lambda tup: -tup[1])
        suggestions.append([tup[0] for tup in distances[:3]])
    return suggestions
```

**Code Block 15:** *Syntactic\_suggestions* Function.

The *semantic\_suggestions* function, displayed in Code Block 16, uses the context provided by the previously validated descriptors to generate suggestions. To do so, it queries the Descriptors Validator database for the most used values for a specific tag. Such as the *syntactic\_suggestions* function, it also returns the three best suggestions.

```
def semantic_suggestions(self):
    suggestions = []
    for error in self.semantic_errors:
        error_suggestions = crud.get_tag_values(self.db_connection, self.ims_version,
        ↪ self.descriptor_type, error["incorrect_tag"])
        suggestions.append(error_suggestions)
    return suggestions
```

**Code Block 16:** *Semantic\_suggestions* Function.

Lastly, the *reference\_suggestions* function, presented in Code Block 17, returns the value of the referenced tag. If the referenced tag is not defined in the descriptor, this function will not return any suggestion.

The output of this module can be observed in Code Block 18.

```

def reference_suggestions(self):
    suggestions = []
    for error in self.reference_errors:
        # 1 - there are possible values
        if error["possible_values"] is not None:
            suggestions.append(error["possible_values"])

        # 2 - if there are no possible values (when a referenced tag is not declared)
        else:
            suggestions.append([])
    return suggestions

```

Code Block 17: *Reference\_suggestions* Function.

```

[ERROR CORRECTION SUGGESTIONS]
-----
Error: The value '1a' on vnf:vnfd-catalog/vnfd/vdu/vm-flavor/vcpu-count doesn't match
its datatype (uint16).
Correction Suggestions: 1, 2, 4
-----
Error: The value 'vnf-cp1' on vnf:vnfd-catalog/vnfd/vdu/interface/external-connection-point
-ref doesn't match its reference (vnfd:vnfd-catalog/vnfd/connection-point/name) value of
vnf-cp0
Correction Suggestions: vnf-cp0

```

Code Block 18: Example of an Output of the Correction Suggestions Module.

### 3.3.4 REST API and GUI

All the features previously described are offered to the NetApp developers via a REST API. This API was implemented using the FastAPI<sup>3</sup> framework. FastAPI is considered the fastest Python web framework, outperforming, for instance, Flask, by more than 100%. Besides, FastAPI offers lower latencies and higher throughputs than most of the other existing web frameworks [86].

The developed REST API is extremely simple, only having two endpoints, presented in Fig. 3.9.

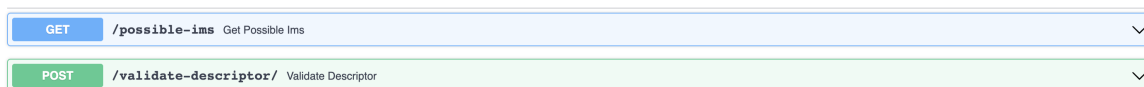


Figure 3.9: REST API's Endpoints.

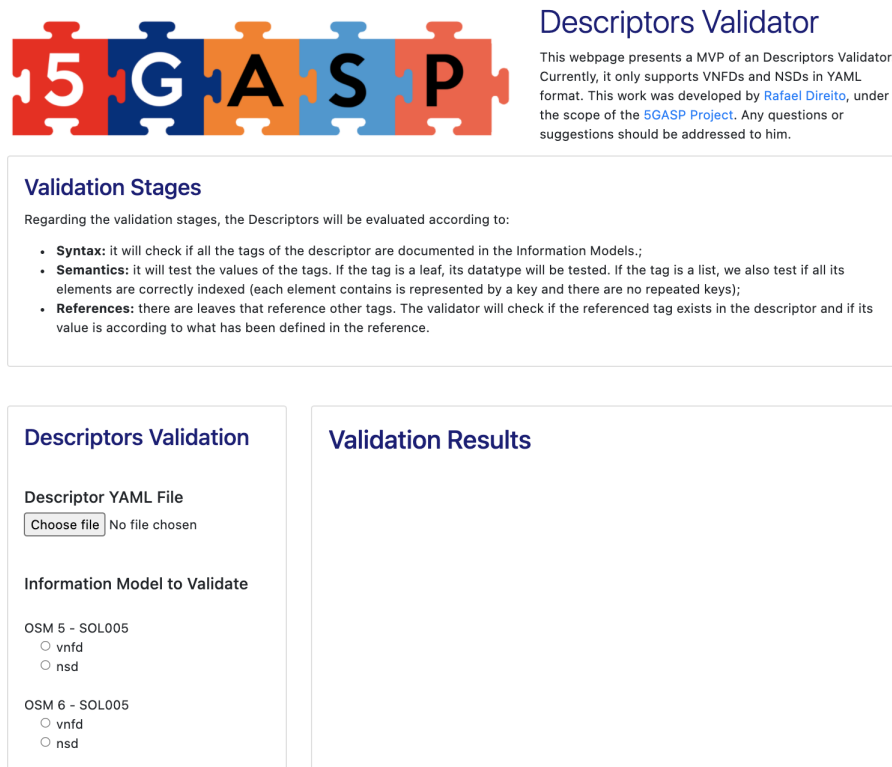
Besides the API, the Descriptors Validator Service is also available via a CLI and a WebUI. The CLI performs requests on the API, thus being also a simple module.

Lastly, the WebUI was implemented using HTML Bootstrap, Cascading Style Sheets (CSS), JavaScript (JS), and jQuery. A simple technological stack was used since the User Interface (UI) is very simplistic and is not expected to extend it in the future. JS and jQuery were used to make requests to the REST API and process its responses.

<sup>3</sup><https://fastapi.tiangolo.com/>



Fig. 3.10 displays the webUI before a descriptor is validated, while Fig. 3.11 displays the GUI after the validation of a descriptor.



The screenshot shows the web interface for the Descriptors Validator. At the top, there is a logo consisting of five interlocking puzzle pieces in red, blue, orange, light blue, and red, with the characters '5', 'G', 'A', 'S', and 'P' on them. To the right of the logo is the title 'Descriptors Validator' and a short introductory paragraph. Below this is a section titled 'Validation Stages' which lists three criteria: Syntax, Semantics, and References. At the bottom, there are two main panels: 'Descriptors Validation' on the left and 'Validation Results' on the right. The 'Descriptors Validation' panel contains a file upload section for the 'Descriptor YAML File' and a section for selecting the 'Information Model to Validate' with radio buttons for 'vnfd' and 'nsd' under two different OSM versions.

## Descriptors Validator

This webpage presents a MVP of an Descriptors Validator. Currently, it only supports VNFs and NSDs in YAML format. This work was developed by [Rafael Direito](#), under the scope of the [5GASP Project](#). Any questions or suggestions should be addressed to him.

### Validation Stages

Regarding the validation stages, the Descriptors will be evaluated according to:

- **Syntax:** it will check if all the tags of the descriptor are documented in the Information Models.;
- **Semantics:** it will test the values of the tags. If the tag is a leaf, its datatype will be tested. If the tag is a list, we also test if all its elements are correctly indexed (each element contains is represented by a key and there are no repeated keys);
- **References:** there are leaves that reference other tags. The validator will check if the referenced tag exists in the descriptor and if its value is according to what has been defined in the reference.

### Descriptors Validation

Descriptor YAML File

No file chosen

Information Model to Validate

OSM 5 - SOL005

vnfd

nsd

OSM 6 - SOL005

vnfd

nsd

### Validation Results

**Figure 3.10:** WebUI Before Validating a Descriptor.

Lastly, both the API and WebUI were packaged into Docker Images, and a Docker Compose was created to deploy this service.

### Descriptors Validation

**Descriptor YAML File**  
 hackfest-ba...e\_error.yaml

**Information Model to Validate**

OSM 5 - SOL005  
 vnfd  
 nsd

OSM 6 - SOL005  
 vnfd  
 nsd

OSM 7 - SOL005  
 vnfd  
 nsd

OSM 8 - SOL005  
 vnfd  
 nsd

OSM 9 - SOL006  
 vnfd  
 nsd

### Validation Results

| Errors                      |                                                                                                                                                                                                                                         |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntactic Validation</b> | Test successful.                                                                                                                                                                                                                        |
| <b>Semantics Validation</b> | <ul style="list-style-type: none"> <li>The value '1a' on vnfd:vnfd-catalog/vnfd/vdu/vm-flavor/vcpu-count doesn't match its datatype (uint16).</li> </ul>                                                                                |
| <b>Reference Validation</b> | <ul style="list-style-type: none"> <li>The value 'vnf-cp1' on vnfd:vnfd-catalog/vnfd/vdu/interface/external-connection-point-ref doesn't match its reference (vnfd:vnfd-catalog/vnfd/connection-point/name) value of vnf-cp0</li> </ul> |

### Correction Suggestions

| Possible Correction Suggestions        |                                                                                                                                                                                                                                                                                   |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntactic Correction References</b> | No need for suggestions.                                                                                                                                                                                                                                                          |
| <b>Semantics Correction References</b> | <p>The value '1a' on vnfd:vnfd-catalog/vnfd/vdu/vm-flavor/vcpu-count doesn't match its datatype (uint16).</p> <p><b>Suggestions:</b></p> <ul style="list-style-type: none"> <li>1</li> </ul>                                                                                      |
| <b>Reference Correction References</b> | <p>The value 'vnf-cp1' on vnfd:vnfd-catalog/vnfd/vdu/interface/external-connection-point-ref doesn't match its reference (vnfd:vnfd-catalog/vnfd/connection-point/name) value of vnf-cp0</p> <p><b>Suggestions:</b></p> <ul style="list-style-type: none"> <li>vnf-cp0</li> </ul> |

Figure 3.11: WebUI After Validating a Descriptor.

### 3.4 CHAPTER SUMMARY

In this Chapter, three crucial phases of the implementation of the Descriptors Validator Service have been addressed.

Section 3.1, fully describes the requirements of this Service. Given these, a collection of specifications has been addressed: (i) the Descriptors Validator Service should provide support for validating both VNFDs and NSDs, (ii) the Service should allow the validation of descriptors relative to different MANO frameworks, and (iii) the Descriptors Validator Service should perform syntactic, semantic, and reference tests. Besides this, Section 3.1 also presents the individual modules of the Descriptors Validator Service: (i) the Information Models Parser module, (ii) the Descriptors Validator module, and (iii) the Correction Suggestions module.

Section 3.2 presents the overall architecture of the Service, defining the inputs and outputs of each individual module and how they are coordinated to create a pipeline for the validation of VNFDs and NSDs. Besides, this section also introduces two new entities: (i) a REST API and (ii) a WebUI. It is through these entities that the NetApp developers interact with the Descriptors Validator Service. Fig. 3.4 presents the complete architecture of this Service, addressing the architecture of each module and how they are inter-connected.

The first module of the Descriptors Validator Service is the Information Models Parser module, which is responsible for converting Information Models from different MANO frameworks to a technology-agnostic structure. As explained before, since, at the time of writing this

dissertation, 5GASP only addressed OSM as its MANO framework, only an OSM Wrapper was implemented. This Python class receives YANG models as input and maps them to a global information structure, which is then stored in 4 YAML files: *tags.yaml*, (ii) *datatypes.yaml*, (iii) *base-datatypes.yaml* , and (iv) *inet-datatypes.yaml*.

The next module, the Descriptors Validator module, uses these files to validate the descriptors. The validation phase results in an errors listing, which is then passed to the Correction Suggestions module. To validate the descriptors, three types of tests are performed: (i) syntactic tests, (ii) semantic tests, and (iii) reference tests, which are fully described in Section 3.1, and their implementation addressed in Section 3.3.

Finally, the Correction Suggestions module is responsible for, given the descriptor's errors and the Information Models used to validate it, generating a list of correction suggestions. If syntactical errors are found, this module will compute the Jaro distance of the wrongly defined tag to the tags defined in the Information Models and suggest the three most similar tags. If semantic errors are found, the Correction Suggestions module will query the Descriptors Validator database and return the three most used values for a tag. These suggestions are based on previously validated descriptors. Lastly, in the presence of a reference error, if the referenced tag is defined in the descriptor, that tag's value will be returned as a correction suggestion.



## 5GASP CI/CD Service

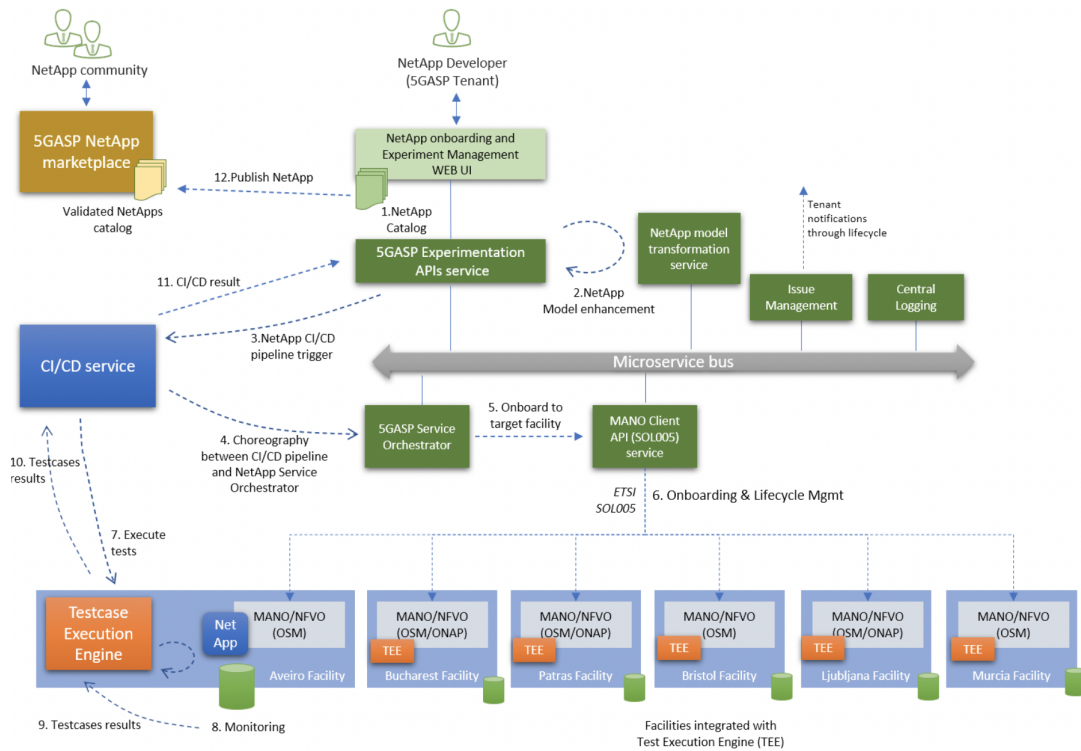
Chapter 3 already presented the architecture and implementation of one of the outputs of this dissertation. This chapter presents the second and most important output of this document: the 5GASP CI/CD Service. Section 4.1 starts by addressing the requirements of this service, which will then be translated into specifications and an architectural design in Section 4.2. The CI/CD Service’s specifications definition is one of the utmost crucial undertakings of this dissertation since this will guide the entire development process of this Service, which aims to present novelty contributions to the testing and validation of 5G NetApps. The implementation of this Service is addressed in Section 4.3, demonstrating how each component was implemented and how it contributes to achieving the initially proposed goals. This chapter also presents information on the Testing Descriptors, which, although not a CI/CD Service component, are the starting point of the validation and testing process.

### 4.1 PROBLEM STATEMENT AND REQUIREMENTS

As already stated in Section 1.1, the 5GASP CI/CD Service is one of the utmost important components of this project. This service enables the NetApps’ developers to automatically test and validate their NetApps, and will, further along the road, enable the certification of NetApps according to the current standards and certification guidelines.

The high-level architecture of 5GASP is presented in Fig. 4.1 to better understand how the CI/CD Service is integrated into this project. This service will coordinate the execution of tests by continuously communicating with the NODS and the Local Test Execution Engines (TEEs), responsible for performing the tests on the NetApps after they are deployed.

To enable a simplified and straightforward validation of the NetApps, the CI/CD Service will have to be fully automated so that, after the beginning of the validation stage, the developers won’t need to have an active participation in the process. Thus, this service will have to make available a simplified way for the developers to provide the specifications of the tests once they onboard their NetApps to the 5GASP NODS. After this, the validation process



**Figure 4.1:** 5GASP Approach on DevOps Experimentation and Certification Lifecycle [87].

won't require any participation from the developers. To enable the described automation, the 5GASP CI/CD Service will adopt CI/CD tools.

To allow the developers to provide the specifications of the tests they wish to perform on their NetApps, 5GASP will enable the onboarding of a bundle containing a NetApp and a Testing Descriptor file, which details all the stages of the validation process. All this information will be onboarded to the NODS, which is the entity responsible for triggering the CI/CD Service.

Since 5GASP is a four-year project that heavily relies on open-source software and aims to achieve high modularity and technological independence, the CI/CD Service can't depend on one CI/CD tool. Instead, it will rely on an abstraction layer that will allow this service to integrate several CI/CD tools. Thus, as its primary component, this service will have an entity named CI/CD Manager that acts as an abstraction layer to the CI/CD tools used to perform the validation stage.

Besides, since one of the goals of the 5GASP project is to enable a globally transparent and distributed testbed, the CI/CD Service will have to deal with all the technical specifications of each individual testbed that is part of the global one. To achieve this, several CI/CD Agents will be deployed in the individual testbeds. These will deal with the singular specifications of the testbed where they are deployed so that the CI/CD Manager won't have to address the specifications of all the testbeds. The CI/CD Manager will be 'testbed agnostic', since it will only have to forward validation jobs to the CI/CD Agents.

Regarding the tests performed on the NetApps, two types of tests will have to be defined:

(i) predefined-tests and (ii) developer-defined tests. The first will already be onboarded in the 5GASP ecosystem so that all the developers can use them to validate global aspects, such as communication latencies or the bandwidth of the testbed. The second will be onboarded by the NetApp developers and address the behavior and singular aspects of the NetApp they developed. Given the maturity of the 5GASP project, for now, only predefined-tests were addressed.

As previously stated, the predefined-tests will already be onboarded in each testbed before the validation stage starts. Thus, an entity will have to be created to store them. This entity is the Local Test Repository (LTR), which provides storage for the tests. During the validation process, the CI/CD Agents will have to gather the tests from the testbed's LTR and then perform them.

All these mechanisms will be defined via a configuration file, which will be dynamically created on the CI/CD Manager, and onboarded on the CI/CD Agents. The Testing Descriptors, already onboarded to the NODS, will be the starting point for creating this configuration file.

Several performance metrics will be collected during the validation phase, such as the Central Processing Unit (CPU) usage or the consumed Random Access Memory (RAM) of a VNF. These metrics will allow the developers to understand the behavior of the VNFs that compose their NetApps.

After the validation stage is completed, the CI/CD Service will have to provide its outputs to the developers, giving them feedback regarding this stage. It is a fact that most of the CI/CD tools already have a mechanism for the developers to get the results and outputs of a testing and validation process. Although, in 5GASP, the developers won't directly interact with these tools. Besides, they won't even know which tools are being used because they will only provide the Testing Descriptors, which are technology agnostic.

So, visualization mechanisms will have to be provided to the developers. These mechanisms will have to provide not only the results of the validation phase but also an overview of this process. For instance, through the listing of all the validation stages, their result, the collected metrics, etc.

## 4.2 ARCHITECTURE AND SPECIFICATIONS

The previous section already listed some of the core components of the CI/CD Service. To summarize, these components are the following: (i) the CI/CD Manager, (ii) the CI/CD Agents, (iii) the LTRs, (iv) the Test Results Visualization Dashboard (TRVD), and (v) the Metrics Repository (MR). A description of each component is presented in the following subsection.

### 4.2.1 Components

#### CI/CD Manager

The CI/CD Manager is the entry-point for the CI/CD Service. It is available via a REST API, implemented using FastAPI, and is triggered by the NODS. After the NetApps are

deployed, the NODS will submit a new test request to the CI/CD Manager, which will create a configuration file to be submitted to the CI/CD Agents located in the testbeds where the NetApp was deployed. After the submission of the configuration file to the Agents, the testing process begins. The CI/CD Agents will then perform the tests and constantly update the test status via the CI/CD Manager's API. To store this information, the CI/CD Manager is also composed of a PostgreSQL database and an File Transfer Protocol (FTP) server to store testing artifacts - the Results Repository. Once the validation process reaches its end, the CI/CD Manager will inform the NODS so the developers can get the results and outputs of the tests.

### CI/CD Agents

The CI/CD Agents, also denominated TEEs, are responsible for performing the tests on the NetApps. Each testbed has a CI/CD Agent deployed on the Network Slice where the NetApps will be deployed, and this Agent is registered in the CI/CD Manager, so it knows how to communicate with it. These Agents, as mentioned before, receive a configuration file from the CI/CD Manager, detailing the testing process. They are responsible for: (i) triggering the metrics collection mechanisms in the NetApps, (ii) gathering the tests from the LTR, (iii) performing them on the NetApps, and (iv) informing the CI/CD Manager of the results of the tests. Besides this, the CI/CD Agents also store the testing artifacts in the CI/CD Manager's Results Repository.

### LTR

The LTRs, implemented via FTP servers, store all the tests that can be performed in a facility and make them available for the CI/CD Agents to gather and execute them on the NetApps. These tests will be implemented via a TAF and validate the behavior of the NetApps and their impact on the underlying infrastructure.

### TRVD

The Test Results Visualization Dashboard enables the NetApp developers to get the outcomes and outputs of the testing and validation phase. After all the tests are performed, the CI/CD Agents will store the outputs and results in the CI/CD Manager's Results Repository. The Test Results Visualization Dashboard will present these outputs so that the developers can check the outcome of the testing phase and provide an Uniform Resource Locator (URL) to access the MR visualization dashboard, so that the developers can observe the metrics collected during the validation process.

### MR

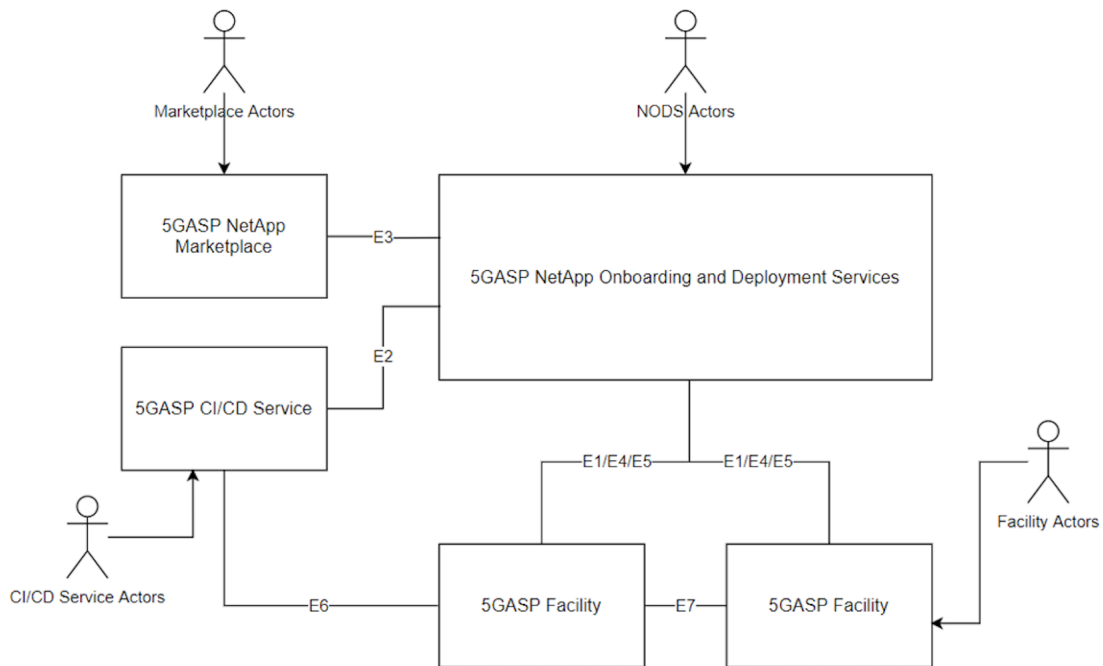
The Metrics Repository is the entity responsible for storing and presenting VNF metrics, such as (i) CPU usage, (ii) RAM consumption, (iii) packets send, etc. Besides providing metrics storage, this entity also provides a metrics visualization dashboard, allowing the developers to visualize the collected metrics.



#### 4.2.2 5GASP CI/CD Service's interaction with the NODS

Given that the NODS is responsible for triggering the CI/CD Service, it is essential to describe the interaction between these two components.

Before describing this interaction, it must be disclosed that the 5GASP CI/CD Service relies on two well-defined communication interfaces: (i) interface E2 - Interface for CI/CD Communication, and (ii) interface E6 - Interface for Facility Interaction with CI/CD. These communication interfaces, presented in Fig. 4.2, were defined in the overall architecture of the 5GASP project and are explained in further detail in [88].



#### Legend

- E1: Interface for communication to the NFVO (SOL005 , etc)
- E2: Interface for CI/CD communication
- E3: Interface for NetApp Marketplace interactions
- E4: Interface for Cross Domain Network Orchestration
- E5: Interface for facility and testing services management
- E6: Interface for facility interaction with CI/CD
- E7: Inter-facility Interface connectivity

**Figure 4.2:** 5GASP's Communication Interfaces [88].

The NetApp developers start by submitting their NetApp and the needed testing information to the NODS, then deploying the NetApp in 5GASP testbeds. It is possible that the NetApps' VNFs have to be augmented by the NODS, in order to install the tools that will be used during the validation process. The NODS will also synchronize the VNF clocks so they follow the same time configuration as the 5GASP ecosystem.

After the deployment of the NetApp, the NODS will render and augment the Testing Descriptor provided by the NetApp developer, adding information such as the IPs of the

deployed VNFs, for instance. After this, the flow is transferred to CI/CD Service through interface E2.

After the CI/CD Manager creates the Agents' configuration files and the validation process begins, the CI/CD Manager will send the NODS information about how the developers can access the TRVD, which enables a continuous feedback chain, allowing the developer to precisely know what the status of the validation process is.

Once the testing and validation process ends, the CI/CD Manager will communicate, once again, with the NODS, informing it of this event.

Fig. 4.3 presents the interaction diagram relative to the communication of the CI/CD Manager with the NODS.

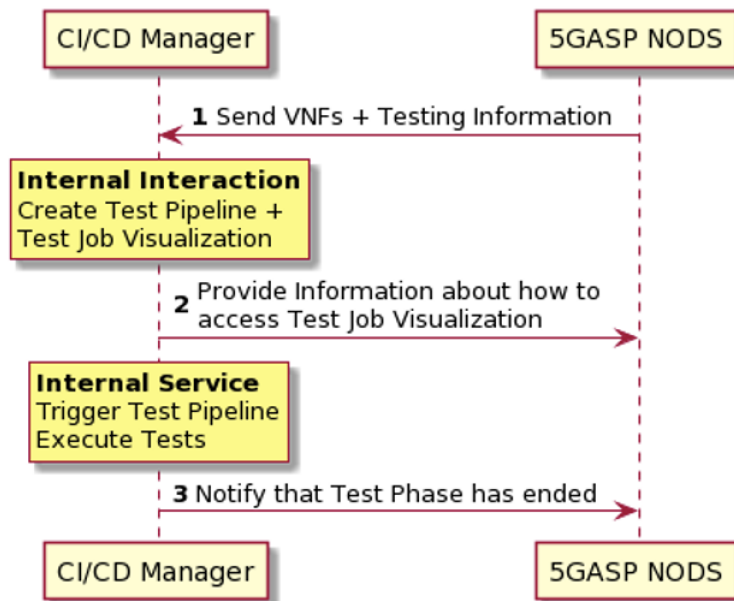


Figure 4.3: E2 Interface Utilization [89]

Although the communication between the CI/CD Service and the NODS is already well-defined, given the lack of maturity of the 5GASP project, the integration between these entities has not been implemented yet. So, in the context of this dissertation, the focus is only on the validation process after the triggering of the CI/CD Service by the NODS.

### 4.2.3 High-Level Architecture

The previous subsections presented the CI/CD Service components and the interaction of this service with the 5GASP NODS. Thus, it is already possible to reveal the high-level architecture of the CI/CD Service. Fig. 4.4 displays this architecture.

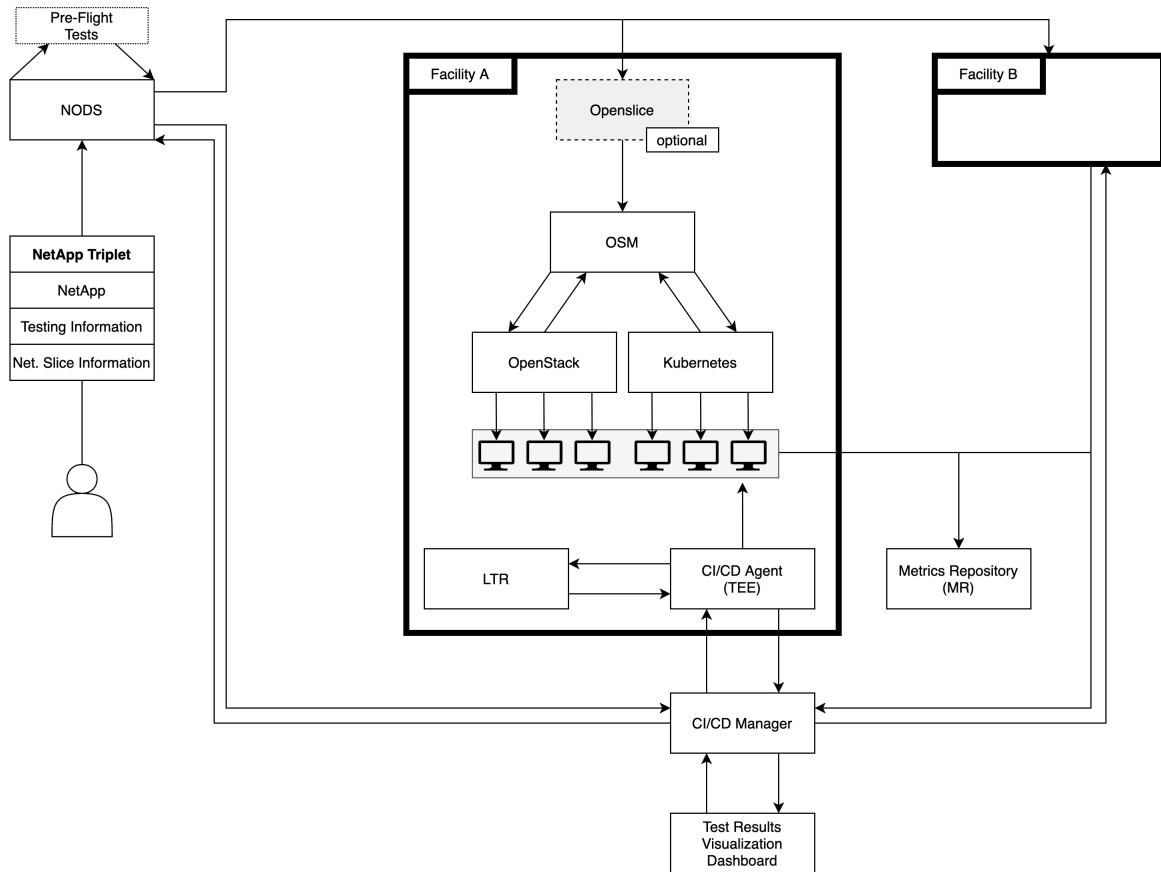


Figure 4.4: 5GASP CI/CD Service’s High-Level Architecture.

#### 4.2.4 Selection of the CI tool to implement the CI/CD Agents

Section 2.4.4 presented and described some Continuous Integration tools. Now, a CI tool has to be chosen to implement the CI/CD Agents.

Given that the 5GASP NODS will store the NetApps, the elected CI/CD tool must not require integration with SCMs, which leaves three available options: (i) Jenkins, (ii) Team City, and (iii) Drone IO. All these tools offer an interaction API and the possibility of self-hosting them, being compliant with the architectural view described before. Thus, the choice of the CI/CD tool will rely on the cost of each tool and its customization capabilities. Team City is excluded when considering the cost since its free version only allows a hundred build configurations, posing severe scalability risks. This leaves Jenkins and DroneIO.

DroneIO, when deployed in self-hosting mode, has restricted features, which might pose some challenges. This does not happen with Jenkins, an open-source CI/CD tool, that can be installed on-premises without added costs and not blocking any of its features. Hence, the final decision is to use Jenkins to implement the CI/CD Agents.

##### Jenkins

Jenkins can be configured using configuration Jenkins Files, which create and configure Jenkins Pipelines. These pipelines are composed of a set of operations that will be performed

during the Continuous Integration workflow and can be of two different types: (i) declarative or (ii) scripted. The scripted pipelines allow Groovy<sup>1</sup> expressions, such as *if/else*, which enables better tailoring of the CI/CD process, thus, being the ideal pipelines for the job at hand.

The scripted pipeline's configuration will be generated in the CI/CD Manager, which will then onboard this configuration on the CI/CD Agents, implemented using Jenkins.

Besides this, the CI/CD Manager will use Jenkins Python Wrapper to interact with the CI/CD Agents, to create credentials, for instance. The existence of (i) several Jenkins Wrappers, (ii) the Jenkins REST API, and its (iii) Visualization Dashboard provides a straightforward configuration and utilization of this tool, which simplifies the implementation of the CI/CD Service.

#### 4.2.5 Selection of the TAF

Given that 5GASP aims to rely on open-source tools, the TAF should also follow this methodology. Besides this, the selected TAF must also be as lightweight as possible and have an easy installation since it will be deployed in all the testbeds. If the configuration of the TAF is a very complex process, delays may occur, which is not desirable.

Besides, the TAF should allow a straightforward test scripting process, thus simplifying test case development.

Lastly, since 5GASP automated testing will rely on CI/CD tools, the TAF must be easily and seamlessly integrated with these.

Given all these requirements, the chosen TAF is the Robot Framework. This framework not only attends all the defined conditions but is already being used by 5GASP consortium members to test their NetApps, which heavily confirms that this framework can be used for 5GASP usage scenario.

#### 4.2.6 Tests Execution

The CI/CD Service offers two different types of tests: (i) predefined-tests that are already onboarded in the 5GASP ecosystem and might access the connectivity and the performance of the communication between the NetApp's components, for instance, and (ii) developer-defined tests that the NetApp developers will onboard, via the NODS, and will test NetApp specific behavior and KPIs.

As previously mentioned, this dissertation will only focus on the predefined-tests, since the lack of maturity of the 5GASP project creates several challenges in dealing with developer-defined tests.

The predefined-tests are developed using the Robot Framework and are already onboarded on the LTRs before the validation process begins. To allow more widespread tests, several environment variables compose these tests, which enables the NetApp developers to configure them straightforwardly.

---

<sup>1</sup><https://groovy-lang.org/>

The configuration of the predefined-tests is achieved via the Testing Descriptor onboarded to the NODS by the NetApp developers. This descriptor is technology-agnostic since it will be the CI/CD Manager that will deal with the implementation of the CI/CD Agents.

When submitted to the NODS, the Testing Descriptor will have several empty fields, which the NODS will then fill after deploying the NetApp and gathering its deployment information. These fields can be, for instance, the IP of the VNFs, the login username and password, etc.

The Testing Descriptor was developed as a joint effort between *Instituto de Telecomunicações - Aveiro* and *OdinS*, a Spanish SME that is part of the 5GASP consortium. Regarding the contributions from *Instituto de Telecomunicações - Aveiro*, they were made solely by me. The descriptor is divided into three different elements that represent the three phases of the testing process: (i) setup, (ii) execution, and (iii) validation. In the scope of this dissertation, the most crucial part is the first one, where the test cases will be defined. Each test case will have a list of parameters, defined using key-value pairs. These will then be added to the CI/CD Agents' configuration file by the CI/CD Manager, which will allow these parameters to perform some configurations on the predefined-tests already onboarded.

After the Jenkins configuration files are submitted to the CI/CD Agents, the testing phase begins. Jenkins will start the metrics collection process and gather the Robot tests from the testbed's LTR, using Python to execute them. Each test will generate three output files: (i) *log.html*, (ii) *report.html*, and (iii) *output.xml*. The first file details the executed test cases in an HTML interactive format. It is composed of a description of the test suite and test cases and describes the keywords used to configure the test. The second file, *report.html*, contains an overview of the results of the execution of the test suite, presenting a summary of the test results. Finally, the *output.xml* file contains all the information about the test execution in an XML format, used internally by Robot to generate the HTML report and log files.

After all the tests are performed, the metrics collection process is stopped, and the tests are then submitted to the CI/CD Manager's Results Repository to make them available to the TRVD. This dashboard will provide a mechanism for the developers to access the results and outputs of the validation phase, allowing the developers to understand why their NetApp might not have passed the 5GASP validation. Besides this, the TRVD will also provide access to the Metrics Repository dashboard.

The TRVD is implemented using elementary technologies such as HTML, Bootstrap, JS, jQuery, and CSS, and communicates with the CI/CD Manager via its REST API to gather the results of the validation process.

#### 4.2.7 Selection of the Monitoring Mechanisms

As stated in Section 2.3.3, OSM already provides a monitoring component. This module collects infrastructure metrics and ships them to Prometheus, making them available to Grafana.

Although the most straightforward process would be to use OSM's monitoring component, this module is not entirely aligned with all the possible VIMs. Regarding OpenStack<sup>2</sup>, for

---

<sup>2</sup><https://www.openstack.org/>

instance, the monitoring component of OSM still does not address its most recent implementations, which might pose a severe risk to the metrics collection process.

Thus, the OSM's built-in monitoring mechanisms will not be used to provide metrics collection and analysis in 5GASP's DevOps methodology.

To enable this, TICK Stack is the chosen monitoring tool. This technological stack is straightforward to install and configure. Besides, the simplified usage of Telegraf to collect metrics is highly appealing, in contrast to Prometheus' exporters, which are quite heterogeneous and do not offer the same simplified configuration as Telegraf does.

Given this, to monitor VNF metrics, a Telegraf binary and a configuration file will be injected into the VNFs and used to gather internal metrics during the validation process, which will provide helpful information to NetApp developers.

#### **4.2.8 Metrics Collection**

The metrics collection process is triggered by the CI/CD Agents when starting the validation process. The collected metrics will be stored in the Metrics Repository InfluxDB and made available via the Chronograf.

To collect VNF-related metrics, the CI/CD Agents injects a Telegraf binary and a Telegraf configuration file in the VNFs. Then, using SSH commands, the Agents will execute the Telegraf binary, thus starting the metrics collection process.

During this process, metrics regarding the CPU, RAM, network, and disk will be collected and sent to the Metrics Repository InfluxDB.

#### **4.2.9 Final Architecture and Workflow**

Since the previous sections already presented all the CI/CD Service components and their implementation aspects, it is now possible to present the detailed architecture and workflow of this Service. Thus, the final architecture is shown in Fig. 4.5, and an interaction diagram is presented in Fig. 4.6.

With these two figures, the reader is better elucidated on the mechanisms implemented in the CI/CD Service, which allows a better understanding of it, and how it achieves its proposed objectives, detailed in Section 2.6.

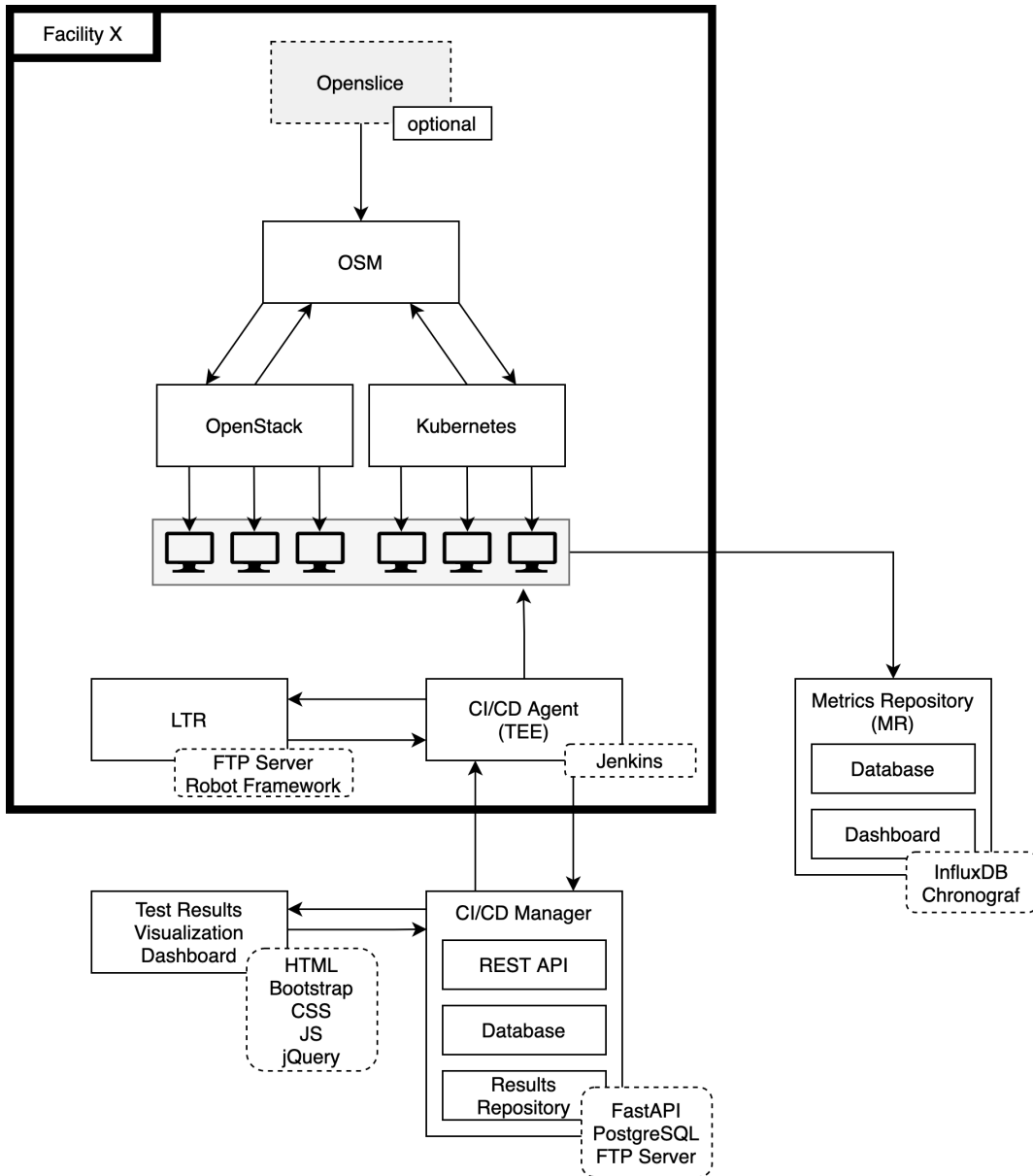


Figure 4.5: CI/CD Service's Architecture.

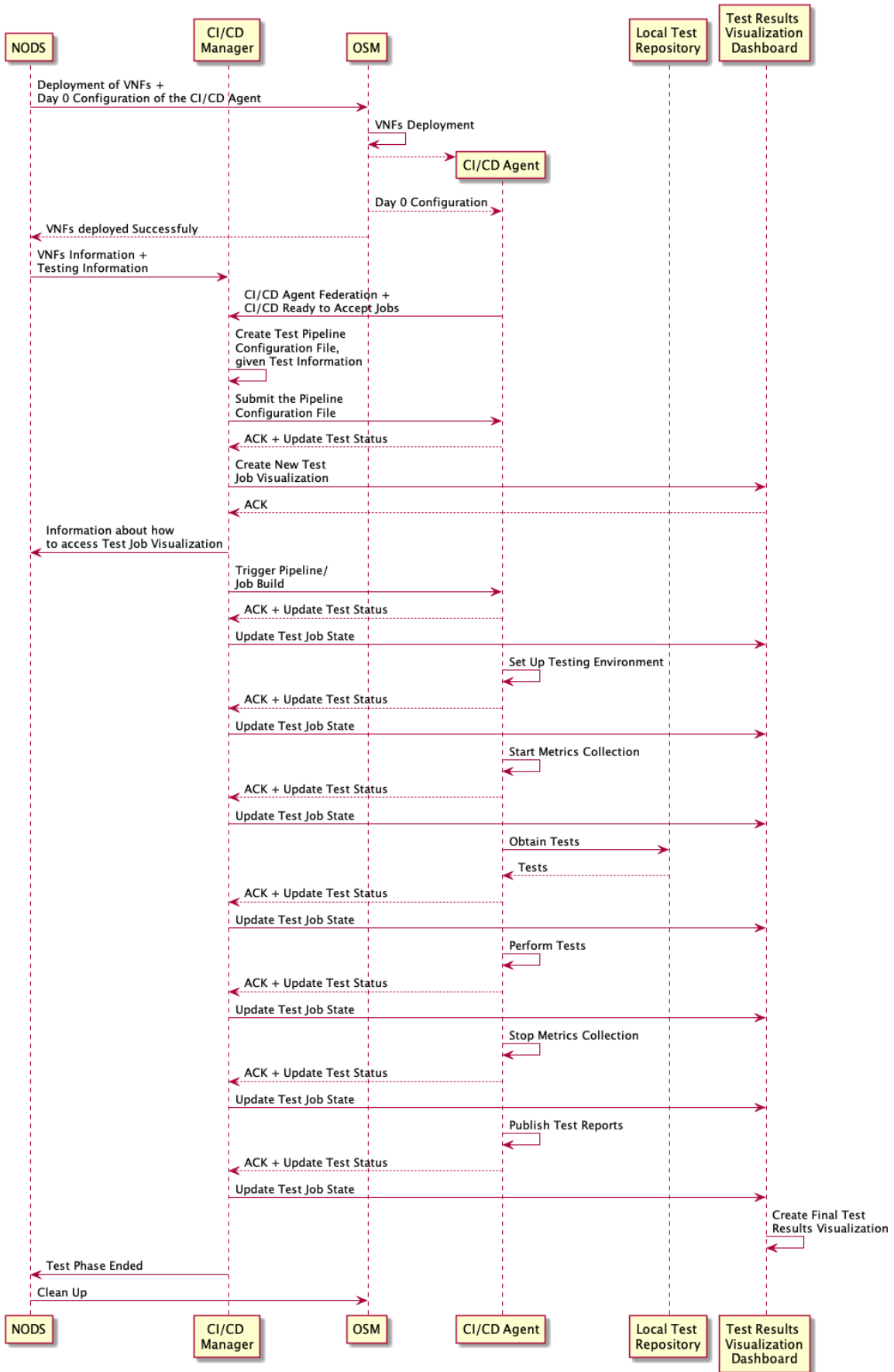


Figure 4.6: CI/CD Service's Interaction Diagram.



## 4.3 IMPLEMENTATION

Since the CI/CD Service is considerably complex, this section is divided into multiple subsections to present how each component was implemented. Besides this, the Testing Descriptors and the mechanisms for metrics collection are also described in this section.

### 4.3.1 CI/CD Manager

The CI/CD Manager is composed of the following components: (i) a REST API that makes available all the business logic of the CI/CD Manager, (ii) a PostgreSQL database which holds all the information relative to the validation processes that were submitted to the CI/CD Manager, and (iii) the Results Repository, which stores the outputs of each validation process.

Similar to the Descriptors Validator Service's API, this REST API was implemented using the FastAPI framework and is composed of four collections of endpoints. These collections of endpoints allow grouping all the endpoints provided by the API in a logical form. The endpoints were grouped in the following categories: (i) testbeds, which contains the methods relative to gathering of the testbeds information, (ii) agents, which groups the endpoints used for the communication between the CI/CD Manager and the CI/CD Agents, (iii) tests, which is composed of the endpoints used to submit new tests and new test status, for instance, and, finally, (iv) the gui collection, that groups all the endpoints that the Test Results Visualization Dashboard will consume. Fig 4.7 displays the Swagger<sup>3</sup> documentation of all the endpoints, enabling the reader to find out which endpoints are provided by the API.

As mentioned before, the CI/CD Manager is triggered by the NODS after deploying all the VNFs of a NetApp. To initiate a validation process, the NODS will submit a Testing Descriptor, already augmented by this component, to the CI/CD Manager, via the `/tests/new` endpoint. The Testing Descriptor is presented and described in Section 4.3.6.

After the submission of the Testing Descriptor, the CI/CD Manager validates the structure of this file using the Python module `cerberus`<sup>4</sup>. The CI/CD Manager is currently performing this validation because the integration between this entity and the NODS has not been achieved yet. Once these two entities are fully integrated, the structural validation of the descriptor will be performed by the NODS.

If the Testing Descriptor is valid, the process may continue, and several other validations are performed. For instance, it is necessary to validate if all tests indicated in the Testing Descriptor exist in the facilities where the VNFs were deployed and if the Testing Descriptor contains all the parameters needed to execute these tests. If everything is according to what is expected, the CI/CD Manager will create a new test instance and generate a communication token that will then be sent to the CI/CD Agent that will perform the tests. The CI/CD Agent was previously chosen according to the testbed where the NetApp was deployed and will use a communication token, sent by the CI/CD Manager, in all communications to prove its identity.

---

<sup>3</sup><https://swagger.io/>

<sup>4</sup><https://docs.python-cerberus.org/en/stable/>

# 5GASP CI\_CD\_Manager 0.0.1 OAS3

/openapi.json

REST API of the 5GASP CI\_CD\_Manager

|                 |                                                             |   |
|-----------------|-------------------------------------------------------------|---|
| <b>agents</b>   | Operations related with the CI/CD Agents.                   | ^ |
| POST            | /agents/new Register new CI/CD Agent                        | ∨ |
| GET             | /agents/all Get all CI/CD Agents                            | ∨ |
| <b>tests</b>    | Operations related with the tests performed on the NetApps. | ^ |
| GET             | /tests/all Get all tests                                    | ∨ |
| GET             | /tests/per-testbed Get testbed's tests                      | ∨ |
| GET             | /tests/test-status Get the status of test                   | ∨ |
| POST            | /tests/test-status Update the status of a test              | ∨ |
| POST            | /tests/new Create a new test                                | ∨ |
| POST            | /tests/publish-test-results Publish test results            | ∨ |
| <b>testbeds</b> | Operations related with the testbeds.                       | ^ |
| GET             | /testbeds/all Get all testbeds                              | ∨ |
| <b>gui</b>      | Endpoints provided to the GUI.                              | ^ |
| GET             | /gui/testing-process-status Get Testing Process Status      | ∨ |
| GET             | /gui/test-console-log Get testing process console log       | ∨ |
| GET             | /gui/test-base-information Get test base information        | ∨ |
| GET             | /gui/tests-performed Get the performed Robot Tests          | ∨ |
| GET             | /gui/test-output-file Get Test Output File                  | ∨ |

Figure 4.7: 5GASP CI/CD Manager API Endpoints.

After this, the Jenkins pipeline configuration is generated from the Testing Descriptor's information and submitted to the CI/CD Agent. Code Block 19 presents a portion of this configuration, relative to the execution of the tests, and its other stages are further addressed in Section 4.3.2. This pipeline configuration contains all the testing key variables and the information needed to access the testbed's LTR and is dynamically generated using a template stored in the CI/CD Manager.

```
pipeline {
  agent any
  stages {
    ...
    stage('Perform Tests') {
      environment {
        bandwidth_host1_ip = '10.0.13.60'
        bandwidth_host1_username = 'ubuntu'
        bandwidth_host1_password = 'password'
        bandwidth_host2_ip = '10.0.13.101'
        bandwidth_host2_username = 'ubuntu'
      }
    }
  }
}
```

```

bandwidth_host2_password = 'password'
bandwidth_threshold = '0.5'
bandwidth_comparator = 'more than'
transmission_speed_host1_ip = '10.0.13.60'
transmission_speed_host1_username = 'ubuntu'
transmission_speed_host1_password = 'password'
transmission_speed_host2_ip = '10.0.13.101'
transmission_speed_host2_username = 'ubuntu'
transmission_speed_host2_password = 'password'
transmission_speed_threshold = '100'
transmission_speed_comparator = 'less or equal than'
packet_loss_host1_ip = '10.0.13.60'
packet_loss_host1_username = 'ubuntu'
packet_loss_host1_password = 'password'
packet_loss_host2_ip = '10.0.13.101'
packet_loss_host2_username = 'ubuntu'
packet_loss_host2_password = 'password'
packet_loss_threshold = '20'
packet_loss_comparator = 'less or equal than'
comm_token = credentials('communication_token')
test_id = 10
}
steps {
  catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE'){
    sh 'python3 -m pip install robotframework==4.1.1 paramiko==2.7.2'
    sh 'python3 -m robot.run -d ~/test_results/"$JOB_NAME"/bandwidth
↔ ~/test_repository/"$JOB_NAME"/tests/bandwidth/testBandwidth.robot'
    sh 'python3 -m robot.run -d ~/test_results/"$JOB_NAME"/transmission_speed
↔ ~/test_repository/"$JOB_NAME"/tests/transmission_speed/testTransmissionSpeed.robot'
    sh 'python3 -m robot.run -d ~/test_results/"$JOB_NAME"/packet_loss
↔ ~/test_repository/"$JOB_NAME"/tests/packet_loss/testPacketLoss.robot'
  }
}
post {
  failure {
    sh 'curl --retry 5 --header "Content-Type: application/json" --request
↔ POST
    --data
↔ \{"communication_token":\{"$comm_token"\},"test_id":\{"$test_id"\},
    "success":false, "state": "PERFORMED_TESTS_ON_CI_CD_AGENT"}\'
    http://192.168.94.53:8000/tests/test-status'
  }
  success {
    sh 'curl --retry 5 --header "Content-Type: application/json" --request
↔ POST
    --data
↔ \{"communication_token":\{"$comm_token"\},"test_id":\{"$test_id"\},
    "success":true, "state": "PERFORMED_TESTS_ON_CI_CD_AGENT"}\'
    http://192.168.94.53:8000/tests/test-status'
  }
}

```

```

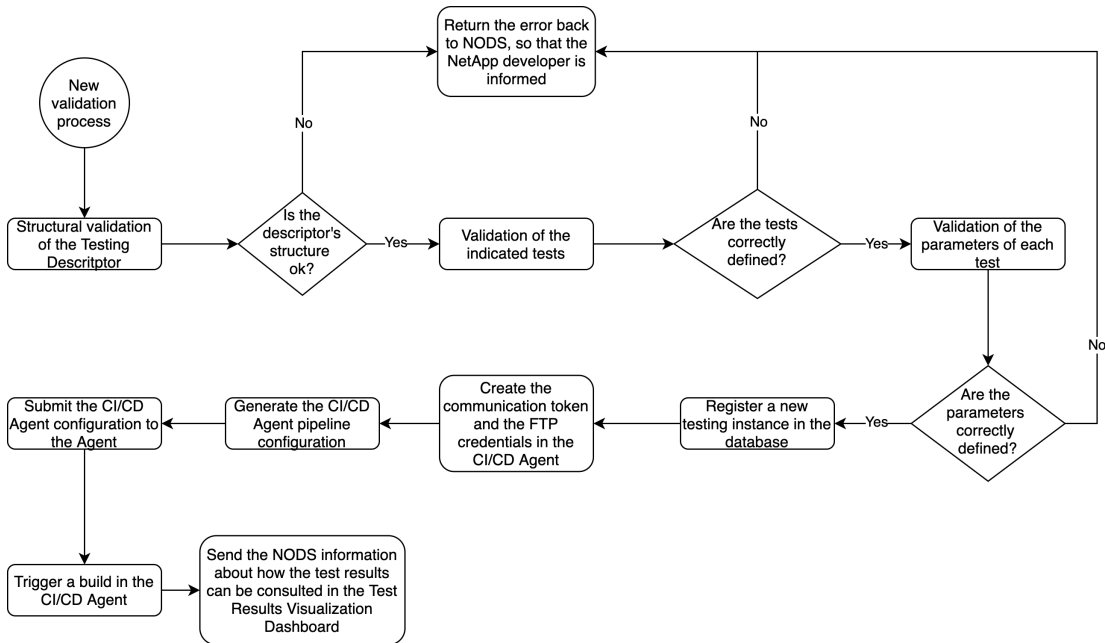
    }
  }
  ...
}
}

```

**Code Block 19:** Jenkins Pipeline Configuration.

After successfully submitting the pipeline’s configuration, the CI/CD Manager will trigger a Jenkins build via the recently created pipeline and inform the NODS that a validation process is starting. To do so, it sends the NODS information such as the unique test id, the CI/CD Agent chosen to perform the tests, and an automatically generated access token, which will allow the NetApp developer to access the Test Results Visualization Dashboard.

The described workflow can be observed in Fig. 4.8.



**Figure 4.8:** Workflow of a New Test Submission to the CI/CD Manager.

During this workflow, and while the testing and validation process is being executed, the CI/CD Manager makes available an endpoint to update the test status. This endpoint is available on `/tests/test-status`. The possible status of a validation process are presented in Table 4.1.

```

|-- obu-network-service-vobu-1
| |-- log.html
| |-- output.xml
| --- report.html
--- obu-network-service-vrsu-1
    |-- log.html
    |-- output.xml
    --- report.html

```

**Code Block 20:** Structure of the Results Repository.

| Test Status                                |
|--------------------------------------------|
| SUBMITTED_TO_CI_CD_MANAGER                 |
| PROVISIONED_CI_CD_AGENT                    |
| AUTHENTICATED_ON_CI_CD_AGENT               |
| CREATED_COMMUNICATION_TOKEN_ON_CI_CD_AGENT |
| CREATED_PIPELINE_SCRIPT                    |
| SUBMITTED_PIPELINE_SCRIPT                  |
| ENVIRONMENT_SETUP_CI_CD_AGENT              |
| STARTED_MONITORING                         |
| OBTAINED_TESTS_ON_CI_CD_AGENT              |
| PERFORMED_TESTS_ON_CI_CD_AGENT             |
| ENDED_MONITORING                           |
| PUBLISHED_TEST_RESULTS                     |
| CLEANED_TEST_ENVIRONMENT                   |
| TEST_ENDED                                 |

**Table 4.1:** Possible Test Status.

When the validation process ends, the CI/CD Agent will inform the CI/CD Manager of this event via the `/tests/publish-test-results` endpoint. The CI/CD Manager, based on the information submitted by the CI/CD Agent, will parse the Robot `output.xml` file to gather the results of each performed test and several other relevant information.

The `output.xml` file, already addressed before, is gathered from the Results Repository, provided by the CI/CD Manager itself. This repository stores all the testing results submitted by the CI/CD Agents and is implemented using a password-protected FTP server. The access credentials are made available to the CI/CD Agents via their Jenkins API, which allows the creation of secret credentials. The structure of the Results Repository is presented in Code Block 20.

The CI/CD Manager's REST API also provides the needed endpoints for a new CI/CD Agent to register himself. However, the process of registering a new CI/CD Agent is not

detailed in this section but instead in Section 4.3.2.

Finally, to fully understand the stored information about the testing and validation process, the database relational model is presented in Fig. 4.9.

The CI/CD Manager’s database is a PostgreSQL object-relational database and is accessed via the SQLAlchemy<sup>5</sup> Python module, which makes available several mechanisms to interact with the database and perform Create, Read, Update and Delete (CRUD) operations.

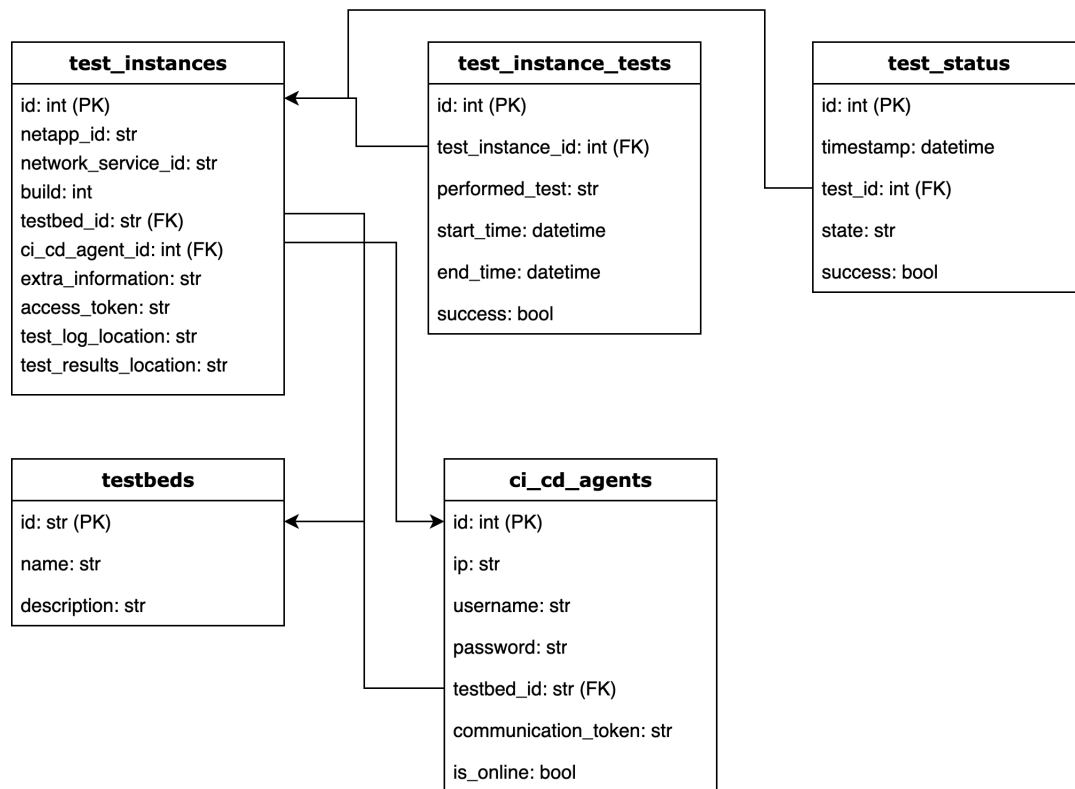


Figure 4.9: CI/CD Manager’s Database Schema.

### 4.3.2 CI/CD Agents

The CI/CD Agents are responsible for gathering the tests from the LTR, executing them, and sending their outputs and results to the CI/CD Manager. Considering implementation aspects, a CI/CD Agent is nothing more than a Jenkins Agent, making its REST API available to the CI/CD Manager.

To deploy a CI/CD Agent, there are two possibilities. The first one is to deploy a VM, manually install Jenkins and all the needed software, and then make a request to the CI/CD Manager to register this new Agent. The second option, and the most simple one, is to deploy a previously configured CI/CD Agent snapshot. This snapshot can easily be deployed in OpenStack, and it includes a *systemd* service that performs all the configurations needed. Once Jenkins is available, this service will change its password, install all the required software

<sup>5</sup><https://www.sqlalchemy.org/>

```

#cloud-config
password: password
chpasswd: {expire: False}
ssh_pwauth: True
write_files:
- content: |
    [JENKINS]
    DefaultUser = admin
    DefaultPassword = admin
    PasswordOutputFile = /var/lib/jenkins/jenkins_new_pw
    [CI_CD_MANAGER]
    Url = http://10.0.13.104:8000
    [LOG]
    Filepath = /var/lib/jenkins/logs/startup.log
path: /var/lib/jenkins/config.ini
permissions: '0644'

```

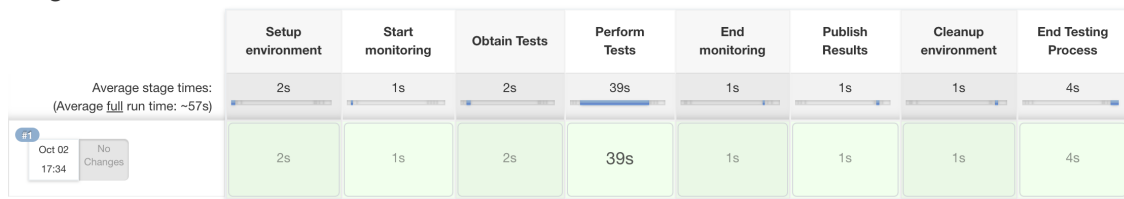
**Code Block 21:** CI/CD Agent’s Cloud-Init Configuration.

tools, and then register the Agent on the CI/CD Manager. This will all happen automatically once the snapshot is booted.

For the CI/CD Manager to register a new CI/CD Agent, the following information is needed: (i) the Agent’s IP, (ii) the Jenkins login credentials, and (iii) the testbed where the Agent is deployed. When deploying the CI/CD Agent snapshot in OpenStack, to provide this information, a cloud-init<sup>6</sup> file will have to be onboarded. The cloud-init will create a configuration file - *config.ini* - used as the source of the several configurations mentioned before. The needed cloud-init configuration can be observed in Code Block 21.

After the initial configuration of the CI/CD Agent, and it is registered in the CI/CD Manager, the Agent is ready to receive validation jobs. These jobs will be performed through a Jenkins Pipeline, which the CI/CD Manager will submit. The pipeline is composed of eight stages, displayed in Fig. 4.10.

### Stage View



**Figure 4.10:** CI/CD Agent’s Pipeline Stages.

In the first stage, the testing environment is configured. To do so, several directories that will store the tests and results are created. During the second stage, Telegraf will be executed, and the metrics collection process starts. Then, the tests are gathered from the LTR and stored locally. These tests will then be executed in the *Perform Tests* stage. After performing the tests, the CI/CD Agent will stop the metrics collection process, publish the tests’ results and outputs to the CI/CD Manager’s Results Repository, and clean up the

<sup>6</sup><https://cloudinit.readthedocs.io/en/latest/>

testing environment. The final stage is to inform the CI/CD Manager that the testing process has ended.

### 4.3.3 LTR

The LTRs are the entities that store the tests that will be performed on the NetApps. Each testbed is composed of an LTR, with the available tests on that testbed.

To implement the LTRs, password-protected FTP servers were used, from where the CI/CD Agents will gather the tests. These tests are stored in the root of an FTP server, and each test files are inside a directory named after the unique test id. When a CI/CD Agent wishes to get some tests, it will only have to execute a *curl* or a *wget* command to retrieve them.

Currently, given the lack of maturity of the 5GASP project, only four tests have been created: (i) `open_ports`, (ii) `bandwidth`, (iii) `packet_loss`, and (iv) `transmission_speeds`. Since these are predefined global tests, they can be distributed across all the testbeds. Code Block 22 presents the file structure of an LTR.

```
.
|-- open_ports
|   |-- OpenPorts.py
|   --- testOpenPorts.robot
|-- bandwidth
|   |-- Bandwidth.py
|   --- testBandwidth.robot
|-- packet_loss
|   |-- PacketLoss.py
|   --- testPacketLoss.robot
--- transmission_speed
    |-- TransmissionSpeed.py
    --- testTransmissionSpeed.robot
```

**Code Block 22:** File Structure of an LTR.



As mentioned before, these tests were developed using the Robot Framework. All are composed of two files: (i) a Robot file, which describes all the test cases, and (ii) a Library file, which contains the test implementation, imported in the Robot file. Code Block 23 presents the Robot file for the *packet\_loss* test, and Code Block 24 its Library file. Given these two Code Blocks, it is easy to understand that the Library file implements a function to obtain the percentage of lost packets. This value will then be consumed in the Robot file and compared to a defined threshold, thus, getting the final test result: success or failure.

```

*** Settings ***
Library          PacketLoss.py

*** Test Cases ***
Testing the packet loss percentage
    ${COMPARATOR}=    Run Keyword If    '${packet_loss_comparator}' == 'more than' Set
↪ Variable    >
    ...    ELSE IF    '${packet_loss_comparator}' == 'more or equal than' Set Variable    >=
    ...    ELSE IF    '${packet_loss_comparator}' == 'less than' Set Variable    <
    ...    ELSE IF    '${packet_loss_comparator}' == 'less or equal than' Set Variable    <=
    ...    ELSE        Fail    \nComparator has not been defined

    ${loss_percentage}=    Packet Loss

    IF    '${loss_percentage}' != '-1'
Should Be True    ${loss_percentage} ${COMPARATOR} ${packet_loss_threshold}
    ELSE
FAIL    \nImpossible to compute packet loss percentage
    END

```

**Code Block 23:** testPacketLoss.robot

```

import paramiko, re
import os

host1 = os.getenv('packet_loss_host1_ip')
username1 = os.getenv('packet_loss_host1_username')
password1 = os.getenv('packet_loss_host1_password')
host2 = os.getenv('packet_loss_host2_ip')

def packet_loss():
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    try:
        client.connect(hostname=host1, username=username1, password=password1)
    except:
        print("[!] Cannot connect to the SSH Server")
        exit()
    stdin, stdout, stderr = client.exec_command(f"ping -c 20 {host2}")
    pingResult = stdout.read().decode()
    regex = re.compile(r',( [0-9,.] +)% packet loss,')
    result = regex.search(pingResult)

```

```

return float(result.group(1)) if result else -1

if __name__ == '__main__':
    packet_loss()

```

#### Code Block 24: PacketLoss.py

In the described test, several parameters are needed. This is common among all the other tests of the 5GASP ecosystem. These parameters must be defined in the onboarded Testing Descriptor. This poses a problem since the developer has to know which parameters to specify in the Testing Descriptor. To solve this, the CI/CD Manager provides an endpoint that lists all the tests available in a testbed and the parameters that are required by these tests. The information provided by this endpoint can be seen in Code Block 25.

```

{
  "message": "",
  "success": true,
  "data": {
    "tests": {
      "bandwidth": {
        "id": "bandwidth",
        "name": "bandwidth",
        "description": "Tests the bandwidth between to VNFs. The results are in
↪ bits/sec",
        "ftp_base_location": "tests/bandwidth/",
        "test_filename": "testBandwidth.robot",
        "test_type": "Robot",
        "test_variables": [
          {
            "variable_name": "host1_ip",
            "description": "IP of Host/VNF 1",
            "mandatory": true,
            "possible_options": [],
            "augmented_by_nods": true,
            "type": "str"
          },
          {
            "variable_name": "host1_username",
            "description": "Login username for Host/VNF 1",
            "mandatory": true,
            "possible_options": [],
            "augmented_by_nods": false,
            "type": "str"
          },
          {
            "variable_name": "host1_password",
            "description": "Login password for Host/VNF 1",
            "mandatory": true,
            "possible_options": [],
            "augmented_by_nods": false,

```

```

        "type": "str"
    },
    {
        "variable_name": "host2_ip",
        "description": "IP of Host/VNF 2",
        "mandatory": true,
        "possible_options": [],
        "augmented_by_nods": true,
        "type": "str"
    },
    {
        "variable_name": "host2_username",
        "description": "Login username for Host/VNF 2",
        "mandatory": true,
        "possible_options": [],
        "augmented_by_nods": false,
        "type": "str"
    },
    {
        "variable_name": "host2_password",
        "description": "Login password for Host/VNF 2",
        "mandatory": true,
        "possible_options": [],
        "augmented_by_nods": false,
        "type": "str"
    },
    {
        "variable_name": "comparator",
        "description": "Comparator to be used while validating the
↔ bandwidth",
        "mandatory": true,
        "possible_options": [
            "more than",
            "more or equal than",
            "less than",
            "less or equal than"
        ],
        "augmented_by_nods": false,
        "type": "str"
    },
    {
        "variable_name": "threshold",
        "description": "Threshold in mbits per second",
        "mandatory": true,
        "possible_options": [],
        "augmented_by_nods": false,
        "type": "float"
    }
}
],
},

```

```

    ...
  },
  "errors": []
}

```

**Code Block 25:** Listing of the tests available in a testbed.

#### 4.3.4 Metrics Collection

The metrics collection process relies on three different components: (i) Telegraf, (ii) InfluxDB, and (iii) Chronograf. The last two components are part of the MR, while Telegraf is injected into the VNFs from which the metrics will be collected.

As mentioned before, a Telegraf configuration file is also injected into the VNFs. This file lists all the metrics that should be collected by Telegraf and contains information about how to access InfluxDB to store the collected metrics.

To collect the metrics, several Telegraf input plugins are used. These describe the metrics scrapping process so that Telegraf knows how to obtain these metrics. Code Block 26 lists the Input Plugins section of the configuration file injected into the VNFs.

```

#####
#                               INPUT PLUGINS                               #
#####

[[inputs.system]]

[[inputs.mem]]

[[inputs.diskio]]

[[inputs.processes]]

[[inputs.swap]]

[[inputs.net]]

[[inputs.cpu]]
  percpu = true
  totalcpu = true
  collect_cpu_time = false
  report_active = false

[[inputs.disk]]
  ignore_fs = ["tmpfs", "devtmpfs", "devfs", "iso9660", "overlay", "aufs", "squashfs"]

```

**Code Block 26:** Telegraf's Configuration File Input Plugins Section.

After injecting the Telegraf binary and its configuration file, the CI/CD Agents trigger Telegraf's execution, consequently starting the metrics collection process.

Then, after gathering and executing the intended tests on the NetApps, the CI/CD Agents stop Telegraf's execution, ending the metrics collection process.

Once the validation process has ended, the NetApp developers will get a URL to access Chronograf, which makes available a dashboard where the metrics can be consulted. Fig. 4.11 presents an example of this dashboard.

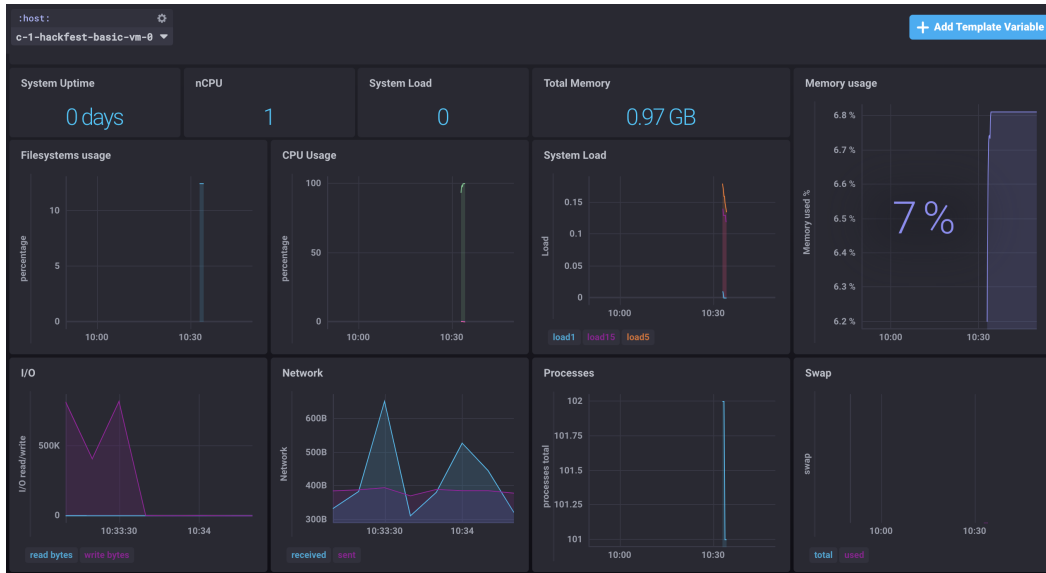


Figure 4.11: Chronograf's Dashboard Example.

### 4.3.5 TRVD

The TRVD allows the NetApp developers to monitor their NetApp's validation process. Whenever a new validation process is triggered in the CI/CD Manager, this entity will generate a random *access\_token* and send it back to the NODS so the developer can obtain this token.

Using the unique test id and the *access\_token*, the NetApp developers can access the TRVD and obtain the following information: (i) the test base information, such as the testbed where the NetApp is being tested and the time the test began and ended, (ii) information about the test stages, (iii) information about the performed tests and their outputs, and (iv) the metrics collected during the validation process. Figs. 4.12, 4.13, and 4.14 portrait all the information available via the TRVD.

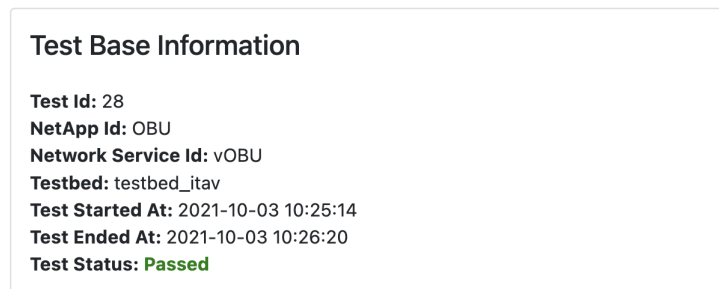


Figure 4.12: TRVD - Test Base Information.

| Testing Process Stages |                                                       |              |                 |
|------------------------|-------------------------------------------------------|--------------|-----------------|
| Timestamp              | Stage Name                                            | Stage Status | Observations    |
| 2021-10-03<br>10:25:14 | submitted_to_ci_cd_manager                            | Success      | No Observations |
| 2021-10-03<br>10:25:15 | provisioned_ci_cd_agent                               | Success      | No Observations |
| 2021-10-03<br>10:25:15 | authenticated_on_ci_cd_agent                          | Success      | No Observations |
| 2021-10-03<br>10:25:16 | created_communication_token_between_manager_and_agent | Success      | No Observations |
| 2021-10-03<br>10:25:19 | created_pipeline_script                               | Success      | No Observations |
| 2021-10-03<br>10:25:20 | submitted_pipeline_script                             | Success      | No Observations |
| 2021-10-03<br>10:25:31 | environment_setup_ci_cd_agent                         | Success      | No Observations |
| 2021-10-03<br>10:25:32 | started_monitoring                                    | Success      | No Observations |
| 2021-10-03<br>10:25:35 | obtained_tests_on_ci_cd_agent                         | Success      | No Observations |
| 2021-10-03<br>10:26:14 | performed_tests_on_ci_cd_agent                        | Success      | No Observations |
| 2021-10-03<br>10:26:15 | ended_monitoring                                      | Success      | No Observations |
| 2021-10-03<br>10:26:17 | published_test_results                                | Success      | No Observations |
| 2021-10-03<br>10:26:19 | cleaned_test_environment                              | Success      | No Observations |
| 2021-10-03<br>10:26:20 | test_ended                                            | Success      | No Observations |

Figure 4.13: TRVD - Test Stages.

| Tests Performed    |                        |                        |             |                                                      |                          |                             |
|--------------------|------------------------|------------------------|-------------|------------------------------------------------------|--------------------------|-----------------------------|
| Test Name          | Start                  | End                    | Test Status | Test Description                                     | Test Log                 | Test Report                 |
| bandwidth          | 2021-10-03<br>10:21:37 | 2021-10-03<br>10:21:43 | Passed      | Test the bandwidth between the OBU and vOBU          | <a href="#">Test Log</a> | <a href="#">Test Report</a> |
| transmission_speed | 2021-10-03<br>10:21:44 | 2021-10-03<br>10:21:49 | Passed      | Test the transmission speed between the OBU and vOBU | <a href="#">Test Log</a> | <a href="#">Test Report</a> |
| packet_loss        | 2021-10-03<br>10:21:50 | 2021-10-03<br>10:22:09 | Passed      | Test the packet loss between the OBU and vOBU        | <a href="#">Test Log</a> | <a href="#">Test Report</a> |
| open_ports         | 2021-10-03<br>10:22:12 | 2021-10-03<br>10:22:12 | Passed      | Test the open ports in the OBU VNF                   | <a href="#">Test Log</a> | <a href="#">Test Report</a> |

| Collected Metrics                                                                                           |                                                                 |
|-------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| To get the metrics collected during the testing and validation process, please open Chronograf's Dashboard. |                                                                 |
| <b>Chronograf</b>                                                                                           |                                                                 |
| URL                                                                                                         | <a href="http://10.0.12.266:8888/">http://10.0.12.266:8888/</a> |
| Username                                                                                                    | admin                                                           |
| Password                                                                                                    | admin                                                           |

Figure 4.14: TRVD - Tests Performed and Collected Metrics.

Regarding the implementation of the TRVD, very simple technologies were used: (i) HTML, (ii) Bootstrap, (iii) JS, (iv) jQuery, (v) AJAX, and (vi) CSS. This is sustained by the

simplicity of the intended web interface. Besides, since the TRVD will act as a monitoring web interface, developing a highly complicated and stylish interface was not ideal.

To obtain the information on the validation processes, the TRVD communicates with the CI/CD Manager via REST and renders the collected data through HTML, Bootstrap, and CSS. Although, if the client wishes to obtain the Robot Framework HTML log files, the TRVD will open a new browser tab with a direct request to the CI/CD Manager since this entity has an endpoint that collects these files and returns them, already rendered. This can be observed in Fig. 4.15.

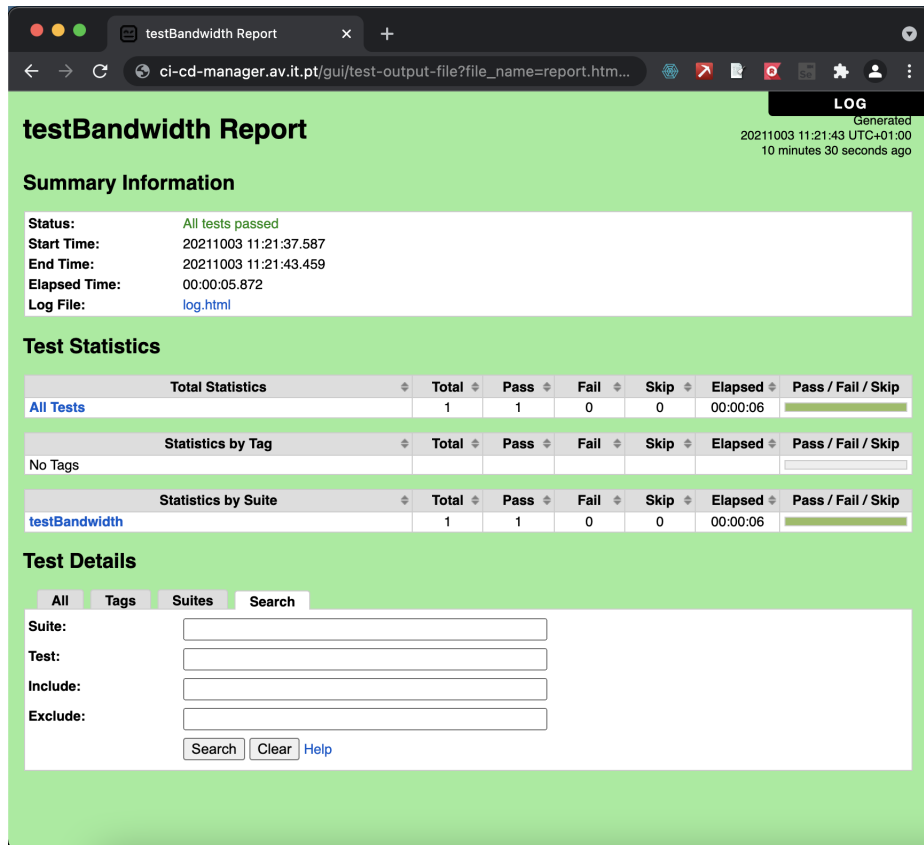


Figure 4.15: TRVD - Test Outputs and Results.

#### 4.3.6 Testing Descriptors

Although the creation of the Testing Descriptors was not initially defined in the action plan for this dissertation, due to developments in the 5GASP project, the design of these descriptors turned out to be a combined effort between *Instituto de Telecomunicações - Aveiro*, and the SME *OdinS*, thus, being presented in this document.

The Testing Descriptors, as mentioned before, will be onboarded by the NetApp developers to the 5GASP NODS, which will augment them and, after the deployment of all the NetApp's VNFs, will send them to the CI/CD Manager to start a new validation process.

A Testing Descriptor is divided into three different elements that represent the three phases of the testing process: (i) setup, (ii) execution, and (iii) validation. The setup phase provides

information for the NODS to deploy all the needed artifacts to perform the validation process and describes all the test cases that will be performed, also configuring all the required test parameters. Besides, in this section are also listed the VNFs on which metrics will be collected. The second phase groups all the tests cases into batches. These tests will be executed together. The last phase describes the gathering of information and the validation of the results of the tests.

When reading Section 4.3.2, it is possible to notice a mismatch between the current implementation of the validation pipeline and what was mentioned regarding the Test Descriptors. Since 5GASP is a highly dynamic project, the architectural and implementation aspects are constantly evolving. Thus, this mismatch.

Implementation-wise, the Testing Descriptors are YAML files, following a specific template provided to the NetApp developers. Besides this, the developers can also consult the test-specific information provided by the CI/CD Manager and displayed in Code Block 25.

Code Block 27 presents a Testing Descriptor, ready to be submitted to the CI/CD Manager, and is used to further describe these descriptors.

```
test_info:
  test_id: 1
  netapp_id: vOBU # From the NODS during the onboarding
  network_service_id: vOBU_ns # From the NODS during the onboarding
  testbed_id: gaia5G # From the NODS during the onboarding
  description: Infrastructure tests for vOBU NetApp
  startTime: startTime # From the NODS during the onboarding
  endTime: endTime # From the NODS during the onboarding
# Test definition in three phases
test_phases:
# Test setup: deploy VNFs if required and define test cases
setup:
  deployments:
# Prepare deployment of custom test VNFs
- deployment_id: 1
  name: vOBU_traffic_generator_vnf_deployment
  descriptor: vOBU_traffic_generator_vnfd
  id: none # From NODS once deployed
  parameters: # optional
    - key: ip_dest
      value: 1.1.1.1
- deployment_id: 2
  name: vOBU_service_consumer_vnf_deployment
  descriptor: vOBU_service_consumer_vnfd
  id: none # From NODS once deployed
  parameters: # optional
    - key: ip_source
      value: 2.2.2.2
- deployment_id: 3
  name: vOBU_vnf_testing
  id: none # From NODS once deployed
  descriptor: vOBU_testing_nsd
```



```

# Define the test cases
testcases:
  # Platform-specific tests
  - testcase_id: 1
    type: predefined
    scope: infrastructure
    name: infrastructure_kpi
    kpi: deployment_time
  # NetApp-specific tests
  - testcase_id: 2
    type: developer_defined
    scope: infrastructure
    name: bandwidth
    parameters:
      - key: host1_ip
        value: 10.0.0.1
      - key: host1_username
        value: root
      - key: host1_password
        value: password
      - key: host2_ip
        value: 10.0.0.2
      - key: host2_username
        value: root
      - key: host2_password
        value: password
      - key: desiredValue
        value: 100 mbps
      - key: comparator
        value: more than
  - testcase_id: 3
    type: developer_defined
    scope: operational
    name: packet_loss_ratio
    parameters:
      - key: host1_ip
        value: 10.0.0.1
      - key: host1_username
        value: root
      - key: host1_password
        value: password
      - key: desiredValue
        value: 1 %
      - key: comparator
        value: less than
  # Custom test VNFs
  - testcase_id: 5
    type: custom_vnfs
    scope: operational
    name: generator_consumer_vnfs

```

```

    vnfs_deploy_id: [1,2]
metrics_collection:
# List the VNFs on which metrics will be collected
- metrics_collection_id: 1
  ip: 10.0.0.1
  username: root
  password: password
- metrics_collection_id: 2
  ip: 10.0.0.2
  username: root
  password: password
# Execute the test cases per scope, define instances, start delay and output of results
execution:
- batch_id: 1
  scope: infrastructure
  executions:
    - execution_id: 1
      name: infrastructure_tests
      testcase_ids: [1]
      instances: 1
    - execution_id: 2
      name: bandwidth_test
      testcase_ids: [2]
      instances: 1
- batch_id: 2
  scope: operational
  executions:
    - execution_id: 1
      name: packet_loss_ratio_test
      testcase_ids: [3,4]
      instances: 1
    - execution_id: 2
      name: custom_vnfs
      testcase_ids: [5]
      instances: 1
# Validate outputs of the test cases
validation:
- validation_id: 1
  execution_id: 5
  file: "vnf_log.txt"

```

**Code Block 27:** Testing Descriptor.

From Code Block 27, it is possible to realize that several information will have to be filled in by the NODS, as mentioned before. The developer may opt to fill this information manually during the initial onboarding of the Test Descriptor to the NODS. However, it is this entity that will have the final word in supplying this information.

The NODS will have to supply information on the NetApp id, the Network Service id (when considering multiple-domain scenarios), the id of the Testbed where the VNFs were deployed, and several timestamps related to the deployment of the VNFs. Besides, it will also

provide the IPs for the VNFs that will be tested and several other VNF-specific information.

During the setup phase, it is possible to observe that several test-VNFs are listed. Although this was not yet implemented when writing this document.

These VNFs are of utmost importance in creating a real-world testing scenario. For instance, they will generate traffic to congest the network or generate network noise.

In Code Block 27 it is also possible to observe the definition of the test-cases and their parameters. This information will be parsed by the CI/CD Manager and then forwarded to the CI/CD Agents to perform the defined tests.

Lastly, the validation phase describes where the logs from a test-VNF should be stored. This section might also present alternative mechanisms to validate the testing results, although, for now, only the Robot Framework is being used to perform these validations.

#### 4.4 CHAPTER SUMMARY

This chapter presents the initial implementation approach of the CI/CD Service and some of its outputs. From what was presented, it is possible to conclude that this implementation already allows a straightforward validation of NetApps. However, there is still a lot of work to be developed to reach a CI/CD Service able to certificate NetApps according to the defined 5G standards.

The 5GASP's CI/CD Service is mainly composed of five components: (i) the CI/CD Manager, (ii) the CI/CD Agents, (iii) the LTRs, (iv) the TRVD, and (v) the MR . These components are briefly described in Section 4.2.1 and their implementation is addressed in Sections 4.3.1, 4.3.2, 4.3.3, 4.3.4, and 4.3.5. Section 4.2.1 also presents the motives for choosing Jenkins to implement the CI/CD Agents, while Section 4.2.5 explains why Robot Framework was the chosen TAF. Finally, Section 4.2.7 presents the reasons for choosing InfluxDB and Chronograf to implement the MR.

Besides the aforementioned components, the 5GASP NODS is also crucial in the validation process since it augments the Testing Descriptors, providing the needed information to perform the tests on the NetApps, and acts as a trigger to the testing and validation process. Section 4.2.2 describes how this entity interacts with the CI/CD Service, and Section 4.3.6 provides extra information on the augmentations performed by the NODS.

The metrics collection is also a fundamental process since it enables the NetApp developers to understand how their NetApp behaved, performance-wise, during the testing process. Sections 4.2.8 and 4.3.4 present relevant information on this process.

Lastly, Section 4.3.6 presents the work developed to achieve a Testing Descriptor structure that can accommodate all the aspects related to the validation process, defining its setup and execution. As previously stated, this work was developed in partnership with a Spanish SME - *OdinS*.



# Results and Evaluation

Having implemented the proposed solutions, it is now needed to validate and evaluate them. Thus, this chapter addresses the results obtained by each implemented Service and presents an evaluation based on these.

Section 5.1 addresses the Descriptors Validator Service. It starts by describing its usage scenarios and then addresses the methodology for testing and evaluating this tool. Then, it presents the results gathered using the described methodology.

On the other hand, Section 5.2 addresses the results and evaluation of the CI/CD Service, following the same structure as Section 5.1.

## 5.1 DESCRIPTORS VALIDATOR

### 5.1.1 Usage Scenarios

The Descriptors Validator Service provides syntactic, semantic, and reference validation of VNF and NS descriptors. Besides this, the Correction Suggestions module enables a simpler descriptor correction process for the developers.

This Service implements a REST API to provide all these features, thus enabling a straightforward integration of this tool with several other services, increasing the Descriptors Validator Service usage scenarios.

Currently, this Service has not been integrated with the 5GASP NODS since this entity already provides a mechanism to validate VNFDs and NSDs. Although, in the future, due to the added benefits of the Correction Suggestions module, this integration is highly probable. Besides the possibility of integrating this service with the NODS, several 5G research projects will likely be interested in using this tool to allow a better user experience when developing NetApps.

At the time of writing this document, the Descriptors Validator Service is available via (i) a REST API, (ii) a CLI, and (iii) a WebUI.

### 5.1.2 Testing Scenario

Although its implementation is completed, the Descriptors Validator Service has still not been used in production scenarios, and neither was integrated with Netapp's onboarding services. Given this, the validation of this tool was performed with a subset of VNFDs and NSDs gathered from the OSM Hackfests. These are community events, which aim to introduce OSM to NetApp developers, and where several VNFs and NSs are developed and used as proofs of concept.

To evaluate the Descriptors Validator Service, several VNFs and NSs with different Information Models were used. ETSI provides access to an FTP server that stores all the VNFs and NSs which have been presented during its Hackfests. Thus, this FTP server offers a vast repository of descriptors, which were used to validate the tool developed during this dissertation.

The aforementioned descriptors were all validated during the OSM Hackfests, being onboarded to OSM and their base VNF and NS packages deployed during these events.

A subset of descriptors, addressing the IMs from OSM 5 to OSM 9, was used to validate the Descriptors Validator Service. This subset was composed of:

- Six descriptors developed accordingly with the IM used in OSM 5;
- Twelve descriptors developed accordingly with the IM used in OSM 6;
- Four descriptors developed accordingly with the IM used in OSM 7;
- Six descriptors developed accordingly with the IM used in OSM 8;
- Eight descriptors developed accordingly with the IM used in OSM 9;

The initial subset was then augmented with invalid descriptors. These descriptors were created from the ones mentioned before, introducing syntactic, semantic, and reference errors. For each IM version, three invalid descriptors were added to the validation descriptors subset. The result was a validation subset of fifty-one descriptors - thirty-six valid descriptors and fifteen invalid ones - which was then shuffled.

To validate the Descriptors Validator Service, all the descriptors were validated using this tool's REST API. Then, several indicators were collected: (i) the classification provided by the Descriptors Validator Service, (ii) the time it took from submitting a descriptor to receive the validation information, and (iii) the validation time of a descriptor. The last indicator only contemplates the time necessary to perform the descriptor's syntactic, semantic, and reference validation after loading it to memory.

Given these indicators, the evaluation of the Descriptors Validator Service addresses the following aspects: (i) if the tool outputted the correct classification (valid or invalid), (ii) if all the errors in the invalid descriptors were listed by this service, (iii) the time needed to process the descriptor, validate it, and create correction suggestions, if the descriptor was invalid, and (iv) the time needed to validate the descriptor after it was loaded into memory.

Besides this, the Correction Suggestion module was also evaluated. To do so, a new descriptors validation subset was created. This subset was composed of all the invalid descriptors present in the previously described validation subset. Thus, this subset was composed of fifteen invalid descriptors.

The Correction Suggestion module was then evaluated on (i) the quality of the outputted suggestions and (ii) the time needed to achieve a correction suggestion.

The results of the validation of the Descriptors Validator Service and its Correction Suggestion module are addressed in Section 5.1.3.

### 5.1.3 Results

The previously presented validation scenario results are presented in the Appendix of this document, in Table 1. These address only the first descriptors subset and present the validation performance.

To allow a better experience while reading this document, not forcing the reader to immediately consult this document's Appendix, Table 5.1 presents some of the results regarding to the descriptors which followed the OSM 8 Information Models.

| IM    | Descriptor                                     | Descriptor Status | Obtained Classification | Errors Found                                                                                                                                                                                              |
|-------|------------------------------------------------|-------------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| osm_8 | invalid-reference_osm8_vnfd-vnf_mpls_vnfd.yaml | Invalid           | Invalid                 | Reference errors: The value 'vnf-cp1' on vnfd:vnfd-catalog/vnfd/vdu/interface/external-connection-point-ref doesn't match it's reference (vnfd:vnfd-catalog/vnfd/connection-point/name) value of vnf-cp0. |
| osm_8 | invalid-semantics_osm8_vnfd-endpoint_vnfd.yaml | Invalid           | Invalid                 | Semantic errors: The value '1 gigabyte' on vnfd:vnfd-catalog/vnfd/vdu/vm-flavor/memory-mb doesn't match it's datatype (uint64).                                                                           |
| osm_8 | invalid-syntax_osm8_nsd-hackfest_epa_nsd.yaml  | Invalid           | Invalid                 | Syntactic errors: Invalid Tag nsd:nsd-catalog/nsd/vlds.                                                                                                                                                   |
| osm_8 | valid_osm8_nsd-hackfest-basic-ns-sriov.yaml    | Valid             | Valid                   |                                                                                                                                                                                                           |
| osm_8 | valid_osm8_nsd-hackfest_epa_nsd.yaml           | Valid             | Valid                   |                                                                                                                                                                                                           |

**Table 5.1:** Subset of the Descriptors Validator Service Classification Results.

As it is possible to observe, the descriptors written for OSM 9, which addresses the SOL006 standard, are not listed in the Results Table. These descriptors were all classified as invalid, which raised severe concerns about the Information Models Parser module, which translates the IMs to a technology-agnostic data structure.

After performing several tests on this module, no error was found, since the produced data structure was according to what was expected. This was followed by a thorough examination of both the Information Models and the VNFDs and NSDs, which allowed to understand the reason why the Descriptors Validator Service was wrongly classifying the descriptors.

This examination concluded that the Descriptors Validator Service was wrongly classifying the descriptors, not because of a design or implementation error, but because, although presented in an OSM Hackfest, the descriptors did not fully follow the Information Models presented in OSM's documentation. Although, these descriptors could be onboarded to OSM and originate a valid VNF or NS. This stresses one of the ongoing problems of OSM, which is the inconsistency between OSM's implementation and its documentation.

During the development of the Descriptors Validator Service, several inconsistencies in OSM's documentation were found. These inconsistencies are widely addressed by OSM's community, and an enormous effort is being made to fix these problems. Currently, the OSM community uses a Slack Workspace<sup>1</sup> to get support from ETSI and other OSM users.

Regarding the results obtained with the first descriptors subset, all the descriptors were correctly classified, and the Descriptors Validator Service was able to find all the introduced errors.

Besides the correct or incorrect classification of the descriptors, the time needed to perform this classification is also an important factor in the adoption of the developed tool. Suppose the Descriptors Validator Service takes a long time to validate the descriptors. In that case, the NetApp developers will prefer to directly onboard their descriptors to OSM and receive the validation results performed by its validation service.

Table 2, in the Appendix of this document, presents the processing time of the Descriptors Validator Service for all the descriptors of the firstly described descriptors subset. This Table also shows the size of the descriptors, in bytes, and the time needed to perform only the validation phase after the descriptor is loaded to memory.

Once again, a reduced sample of the processing times is presented in Table 5.2.

---

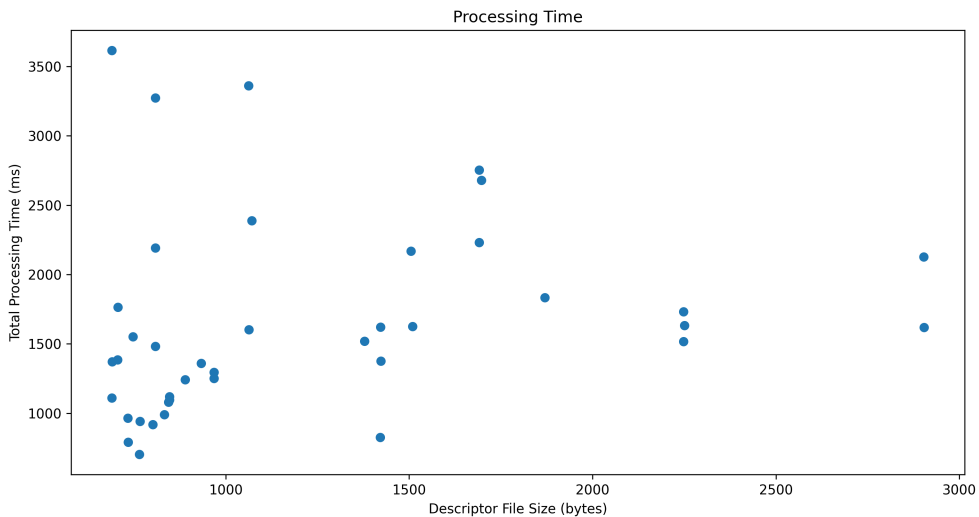
<sup>1</sup><https://opensourcemano.slack.com/>



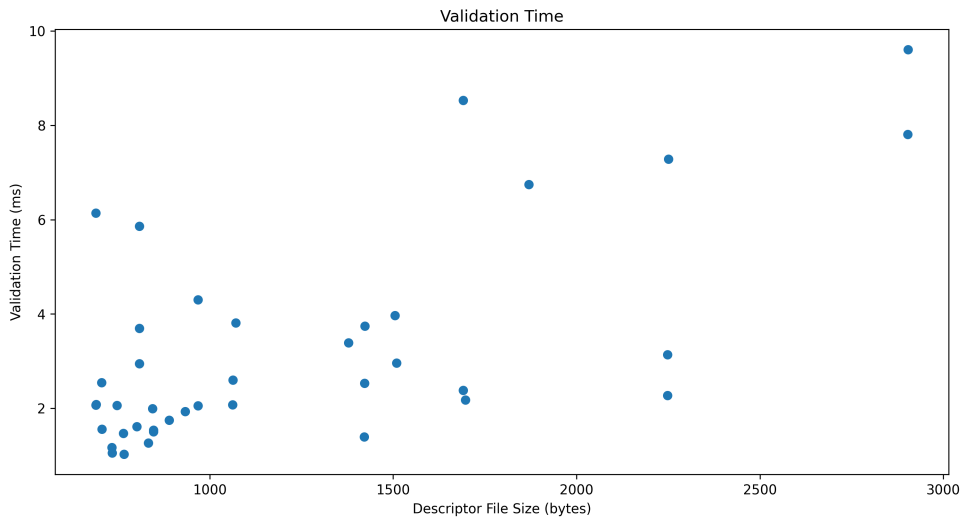
| IM    | Descriptor                                     | File Size (bytes) | Total Processing Time (ms) | Validation Time (ms) |
|-------|------------------------------------------------|-------------------|----------------------------|----------------------|
| osm_8 | invalid-reference_osm8_vnfd_vnf_mpls_vnfd.yaml | 808               | 3274.13                    | 3.69                 |
| osm_8 | invalid-semantics_osm8_vnfd_endpoint_vnfd.yaml | 1697              | 2678.98                    | 2.17                 |
| osm_8 | invalid-syntax_osm8_nsd_hackfest_epa_nsd.yaml  | 690               | 1370.3                     | 2.08                 |
| osm_8 | valid_osm8_nsd_hackfest-basic-ns-sriov.yaml    | 846               | 1119.52                    | 1.53                 |
| osm_8 | valid_osm8_nsd_hackfest_epa_nsd.yaml           | 689               | 1109.26                    | 2.06                 |

**Table 5.2:** Sample of the Descriptors Validator Service Processing Times.

This information was synthesized, and two scatter plots were created. The first scatter plot, presented in Fig. 5.1, relates the descriptors' file size, in bytes, with the overall processing time. The second, illustrated in Fig. 5.2, relates the size of the descriptors with the validation time after it is loaded to memory.



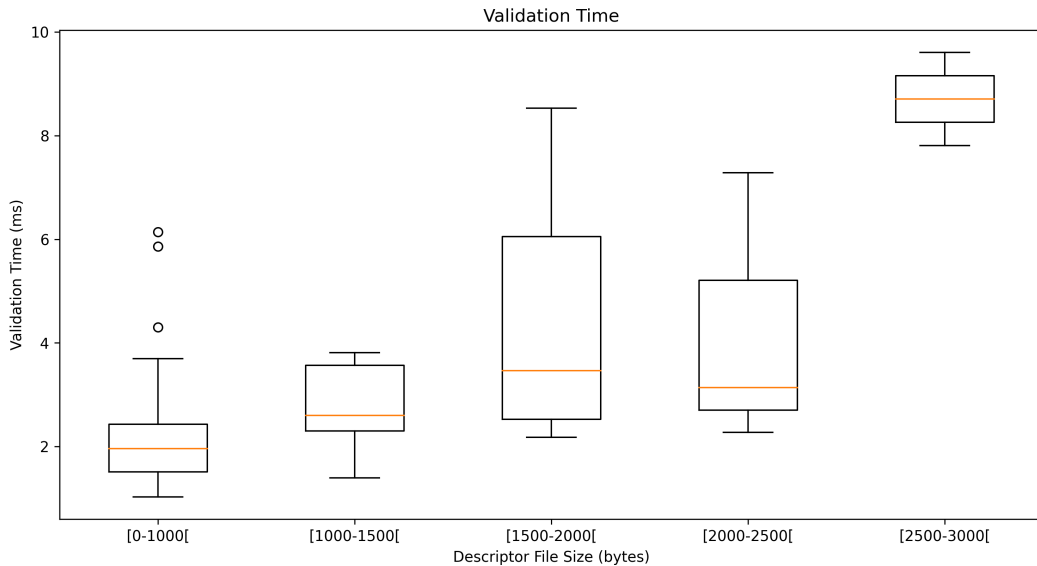
**Figure 5.1:** Relation Between the Descriptor's File Size and its Overall Processing Time - Scatter Plot.



**Figure 5.2:** Relation Between the Descriptor’s File Size and its Validation Time - Scatter Plot.

The overall processing time, presented in Fig. 5.1, contemplates the time needed to submit a descriptor to the Descriptors Validator Service, its loading to memory, the validation time, and the time required to generate correction suggestions, while the validation time scatter plot, depicted in Fig. 5.2, only addresses the validation time.

A box plot addressing the relationship between the descriptor’s size and the validation time is also presented in Fig. 5.3.



**Figure 5.3:** Relation Between the Descriptor’s File Size and its Validation Time - Box Plot.

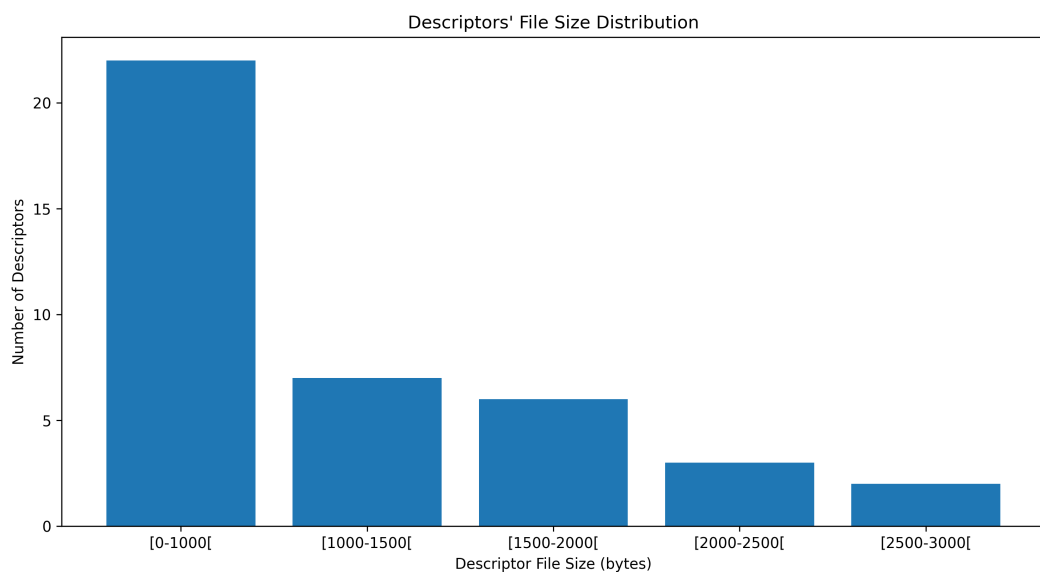
Examining this box plot, it is possible to gather the data presented in Table 5.3.

| <b>Descriptors File Size (bytes)</b> | <b>Median of the Validation Time (ms)</b> |
|--------------------------------------|-------------------------------------------|
| Less than 1000 bytes                 | 1.95                                      |
| Between a 1000 bytes and 1500 bytes  | 2.59                                      |
| Between a 1500 bytes and 2000 bytes  | 3.46                                      |
| Between a 2000 bytes and 2500 bytes  | 3.14                                      |
| Between a 2500 bytes and 3000 bytes  | 8.71                                      |

**Table 5.3:** Relation Between the Descriptor's File Size and its Median Validation Time.

Given this data, it is possible to conclude that the validation time increases with the file size of the descriptor. Although, it is possible to observe that the descriptors with a file size between 2000 and 2500 bytes had an inferior validation time than those with a file size between 1500 and 2000 bytes. This is due to the heterogeneous distribution of the descriptors' file sizes.

Fig. 5.4 presents the distribution of the subset descriptor's file size. From this figure, it is possible to conclude that about 50% of the descriptors have a file size of less than 1000 bytes. Besides, the subset only contains three descriptors with a file size between 2000 and 2500 bytes and two descriptors with a file size between 2500 and 3000 bytes.



**Figure 5.4:** Descriptor's File Size Distribution.

As mentioned before, a second descriptors subset was used to evaluate the Correction Suggestions module. This subset comprises twelve descriptors with an equal proportion of syntactical, semantic, and reference invalid descriptors. Originally, this subset was composed of fifteen invalid descriptors, although, as explained before, the descriptors targeting the OSM 9 Information Models were not addressed.

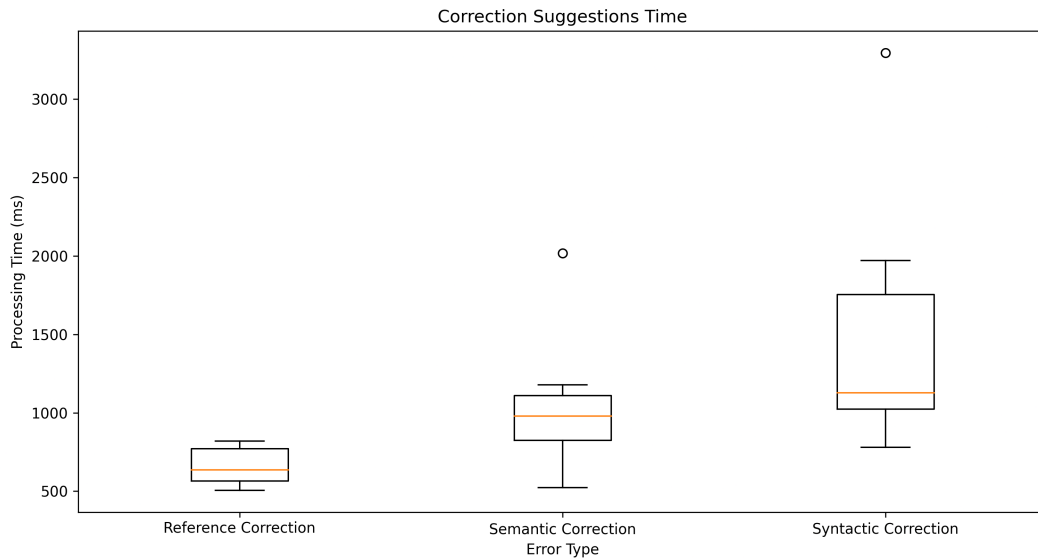
Table 3, in this document’s Appendix, lists the results obtained by the Correction Suggestions Module. Table 5.4, presents a sample of the results presented in Table 3 of the Appendix. This sample only addresses the invalid descriptors which followed OSM 8 Information Models.

| IM    | Descriptor                                     | Error Found                                                                                                                                                                             | Correction Suggestions                                                              |
|-------|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| osm_8 | invalid-syntax_osm8_nsd_hackfest_epa_nsd.yaml  | Invalid Tag nsd:nsd-catalog/nsd/vlds.                                                                                                                                                   | ['nsd:nsd-catalog/nsd/vld', 'nsd:nsd-catalog/nsd/vld/id', 'nsd:nsd-catalog/nsd/id'] |
| osm_8 | invalid-semantics_osm8_vnfd_endpoint_vnfd.yaml | The value '1 gigabyte' on vnfd:vnfd-catalog/vnfd/vdu/vm-flavor/memory-mb doesn't match it's datatype (uint64).                                                                          | [1024, 2048, 4096]                                                                  |
| osm_8 | invalid-reference_osm8_vnfd_vnf_mpls_vnfd.yaml | The value 'vnf-cp1' on vnfd:vnfd-catalog/vnfd/vdu/interface/external-connection-point-ref doesn't match it's reference (vnfd:vnfd-catalog/vnfd/connection-point/name) value of vnf-cp0. | ['vnf-cp0']                                                                         |

**Table 5.4:** Sample of the Correction Suggestions Module Results.

To validate if a correction suggestion is valid, thus correcting the discovered error, the descriptors were updated according to the suggestions given by the Correction Suggestions Module and then onboarded to the different versions of OSM via its CLI. If the onboarding of the descriptors to OSM is successful, it is possible to conclude that the correction suggestion is valid. This happened will all the given suggestions, which indicates an excellent performance of this module.

The time needed to achieve a correction suggestion was also addressed. Its analysis is presented in Fig. 5.5. Examining this box plot, it is possible to gather the data presented in Table 5.5. It is possible to conclude that generating correction suggestions for reference errors is less demanding than generating these suggestions for semantic and syntactic errors. The last two are the errors for which generating a correction suggestion is more challenging.



**Figure 5.5:** Relation Between the Descriptor’s Error Type and the Time Needed to Generate Correction Suggestions.

| Error Type      | Median of the Time Needed to Generate Correction Suggestions (ms) |
|-----------------|-------------------------------------------------------------------|
| Reference Error | 636.51                                                            |
| Semantic Error  | 978.94                                                            |
| Syntactic Error | 1127.41                                                           |

**Table 5.5:** Relation Between the Descriptor’s Error Type and its Median Correction Suggestions Generation Time.

The results presented in Fig. 5.5 and Table 5.5 address only descriptors with one error. Since the Correction Suggestions Module sequentially and individually generates suggestions for the errors, the time needed to generate correction suggestions for descriptors with several errors is proportional to the results presented.

Given the obtained results, it is possible to speculate that the Descriptors Validator Service can be widely adopted by NetApp developers, given that its classification performance is excellent and that it takes no more than five seconds to validate a descriptor with less than three errors, thus having a good user experience and not increasing the development time of

VNFs and NSDs.

## 5.2 5GASP CI/CD SERVICE

### 5.2.1 Usage Scenarios

The 5GASP CI/CD Service was built aiming for maximum modularity, enabling its straightforward integration, not only with the 5GASP ecosystem but also with other NetApp-related projects. Although, the work developed during this dissertation targeted the integration of this Service with the 5GASP ecosystem.

Thus, the primary usage scenario of this Service is the test and validation of NetApps within the 5GASP ecosystem. To enable this, the CI/CD Service will have to be integrated with the 5GASP NODS, which has not been possible until the date of writing this document, given the lack of maturity of the 5GASP project.

When integrated with the NODS, the CI/CD Service will be triggered by this entity after all the VNFs of a NetApp are deployed. To trigger the CI/CD Service, as mentioned before, the NODS will have to submit a Testing Descriptor to the CI/CD Manager. This descriptor is an augmented version of the descriptor onboarded by the NetApp developer, as detailed in Section 4.2.2 .

The CI/CD Service will then return to the NODS information regarding how the NetApp developer can follow the validation process, which is enabled by the TRVD's web interface.

Then, the validation stage begins, and the CI/CD Agents responsible for performing the tests, which will gather these tests from the LTRs, perform them, and sent their results and outputs back to the CI/CD Manager, allowing the developers to get feedback on the validation stage.

### 5.2.2 Testing Scenario

Like the Descriptors Validator Service, although the CI/CD Service's implementation has good maturity, it has not been in production since it would require its integration to the 5GASP NODS. Thus, it wasn't possible to test this Service in a real-world scenario.

Although, several tests were performed. To evaluate the CI/CD Service, two NSs have been used. These Network Services were composed of two VNFs and obtained from OSM Hackfests, targetting OSM's Release 9. Since the created validation tests only target infrastructure and security aspects, evaluating the CI/CD Service with more NSs and VNFs would not generate more value. Thus, the decision of testing the Service using just two NSs.

Three different test descriptors were developed to evaluate the CI/CD Service. These descriptors mimic the ones that will be submitted to the CI/CD Manager by the NODS and represent three different scenarios: (i) a successful validation process, (ii) a validation process where some of the tests performed failed, and a poorly defined Testing Descriptor which won't trigger a testing process on the CI/CD Agents, so, no tests will be executed. The two correctly defined Testing Descriptors will target all the tests already available in the 5GASP ecosystem but will define different parameters for the testing phase.

With these descriptors, it will be possible to understand if the CI/CD Service's behavior is the expected one. To do so, the results of the validation phase will have to be validated to verify if the feedback received by the NetApp developers is the desired one.

Besides this, the validation process execution time will also be targeted during the CI/CD Service evaluation. The validation phase should take as little time as possible so that the developers can have feedback on their NetApps as briefly as possible. Obviously, the execution time of the validation phase is not independent of the individual test execution times, which is why these values will also be collected and analyzed.

Lastly, it must be verified if the correct metrics are being collected. To evaluate the quality of these metrics, the Metrics Repository Dashboard will have to be explored to verify if it presents all the desired metrics and if they were collected correctly.

### 5.2.3 Results

The evaluation process of the CI/CD Service started with the VNF and NS preparation. The VNFs and NSs were augmented with cloud-inits, which is an essential step to allow the installation of the software involved in the validation process. This simulates the process that will occur in the NODS once a NetApp is onboarded, which is described in Section 4.2.2 .

Then, the VNFs and NSs were onboarded to an OSM Release 9 instance and deployed. Their IPs were collected so that they could be defined in the Testing Descriptors. Once again, this process mimics the processing of the Testing Descriptors in the NODS.

After this, the Testing Descriptors were conceived. Testing Descriptor A is a correctly defined Testing Descriptor and addresses all the tests loaded in the premises LTR. This test defines easily achieved KPIs, such as (i) a minimum bandwidth of 0.5 Mbps, (ii) a maximum packet loss of 20%, and (iii) a maximum latency of 200 ms. Regarding the *open\_ports* test, this descriptor correctly describes the ports that should be open. Testing Descriptor B was also correctly defined, although it was created to fail one of the performed test cases. To achieve this, the parameters of the *open\_ports* test were defined in order for this test to fail. The last developed Test Descriptor - Testing Descriptor C - addressed tests that didn't exist in the LTR on premises. Due to this situation, the validation phase shouldn't start, and the NetApp developer should be informed that the chosen tests are not available.

The test cases defined in the Testing Descriptor A are presented in Code Block 28.

```
testcases:
  - testcase_id: 1
    type: predefined
    scope: infrastructure
    name: bandwidth
    description: Test the bandwidth between the OBU and vOBU
    parameters:
      - key: host1_ip
        value: 10.0.13.60
      - key: host1_username
        value: ubuntu
      - key: host1_password
```

```

    value: password
- key: host2_ip
  value: 10.0.13.101
- key: host2_username
  value: ubuntu
- key: host2_password
  value: password
- key: threshold
  value: 0.5
- key: comparator
  value: more than

- testcase_id: 2
type: predefined
scope: infrastructure
name: transmission_speed
description: Test the transmission speed between the OBU and vOBU
parameters:
  - key: host1_ip
    value: 10.0.13.60
  - key: host1_username
    value: ubuntu
  - key: host1_password
    value: password
  - key: host2_ip
    value: 10.0.13.101
  - key: host2_username
    value: ubuntu
  - key: host2_password
    value: password
  - key: threshold
    value: 200
  - key: comparator
    value: less or equal than

- testcase_id: 3
type: predefined
scope: operational
name: packet_loss
description: Test the packet loss between the OBU and vOBU
parameters:
  - key: host1_ip
    value: 10.0.13.60
  - key: host1_username
    value: ubuntu
  - key: host1_password
    value: password
  - key: host2_ip
    value: 10.0.13.101
  - key: host2_username

```



```

    value: ubuntu
  - key: host2_password
    value: password
  - key: threshold
    value: 20
  - key: comparator
    value: less or equal than

- testcase_id: 4
type: predefined
scope: security
name: open_ports
description: Test the open ports in the OBU VNF
parameters:
  - key: host
    value: 10.0.13.60
  - key: expected_open_ports
    value: 22|tcp

```

**Code Block 28:** Testing Descriptor A Test Cases.

The testing of the CI/CD Service started with the submission of Testing Descriptor C, which, as expected, didn't trigger a validation pipeline. Besides this, the CI/CD Manager returned the message presented in Code Block 29.

```

{
  "message": "Error on validating the tests to perform",
  "success": false,
  "data": [],
  "errors": [
    "The script_injection test doesn't exist in the selected testbed."
  ]
}

```

**Code Block 29:** Output of the Submission of a Testing Descriptor Describing a Non-Existent Test.

Then, Testing Descriptor A was submitted to the CI/CD Manager, which responded with a success message containing the unique test identifier and the *access\_token* needed to follow the validation process via the TRVD.

As expected, all the performed tests were successful, and it was possible to obtain this information via the TRVD. Fig. 5.6 displays the web interface of the TRVD after the validation process was concluded, and Fig. 5.7 shows the *report.html* file for the open ports test.

Finally, Testing Descriptor B was submitted to the CI/CD Manager, triggering a validation process. As expected, the validation process was unsuccessful because the *open\_ports* test had failed since the targetted VNF had open ports that weren't defined in the descriptor. Fig. 5.8 shows the web interface of the TRVD, and Fig. 5.9 displays the *open\_ports* test *report.html* file, created by the Robot Framework. Via the TRVD, a NetApp developer can

also obtain the console log of the validation process. Fig. 5.10 displays the errors relative to the open ports tests, available in the validation process console log.

| Test Name          | Start               | End                 | Test Status | Test Description                                     | Test Log                 | Test Report                 |
|--------------------|---------------------|---------------------|-------------|------------------------------------------------------|--------------------------|-----------------------------|
| bandwidth          | 2021-10-03 10:21:37 | 2021-10-03 10:21:43 | Passed      | Test the bandwidth between the OBU and vOBU          | <a href="#">Test Log</a> | <a href="#">Test Report</a> |
| transmission_speed | 2021-10-03 10:21:44 | 2021-10-03 10:21:49 | Passed      | Test the transmission speed between the OBU and vOBU | <a href="#">Test Log</a> | <a href="#">Test Report</a> |
| packet_loss        | 2021-10-03 10:21:50 | 2021-10-03 10:22:09 | Passed      | Test the packet loss between the OBU and vOBU        | <a href="#">Test Log</a> | <a href="#">Test Report</a> |
| open_ports         | 2021-10-03 10:22:12 | 2021-10-03 10:22:12 | Passed      | Test the open ports in the OBU VNF                   | <a href="#">Test Log</a> | <a href="#">Test Report</a> |

Figure 5.6: TRVD’s Web Interface Portraiting a Successful Validation Process.

REPORT  
Generated  
20211003 11:22:12 UTC+01:00  
13 minutes 21 seconds ago

## testOpenPorts Log

### Test Statistics

| Total Statistics | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip |
|------------------|-------|------|------|------|----------|--------------------|
| All Tests        | 1     | 1    | 0    | 0    | 00:00:00 | ██████████         |

| Statistics by Tag | Total | Pass | Fail | Skip | Elapsed | Pass / Fail / Skip |
|-------------------|-------|------|------|------|---------|--------------------|
| No Tags           |       |      |      |      |         | ██████████         |

| Statistics by Suite           | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip |
|-------------------------------|-------|------|------|------|----------|--------------------|
| <a href="#">testOpenPorts</a> | 1     | 1    | 0    | 0    | 00:00:00 | ██████████         |

### Test Execution Log

**SUITE** testOpenPorts

**Full Name:** testOpenPorts

**Source:** [/var/lib/jenkins/test\\_repository/OBU-vOBU-26/tests/open\\_ports/testOpenPorts.robot](#)

**Start / End / Elapsed:** 20211003 11:22:12.735 / 20211003 11:22:12.866 / 00:00:00.131

**Status:** 1 test total, 1 passed, 0 failed, 0 skipped

00:00:00.131

---

**TEST** Testing the open ports

00:00:00.087

Figure 5.7: *report.html* Robot File for the Successful Execution of the Open Ports Test.

| Test Name          | Start               | End                 | Test Status | Test Description                                     | Test Log                 | Test Report                 |
|--------------------|---------------------|---------------------|-------------|------------------------------------------------------|--------------------------|-----------------------------|
| bandwidth          | 2021-10-03 10:32:32 | 2021-10-03 10:32:38 | Passed      | Test the bandwidth between the OBU and vOBU          | <a href="#">Test Log</a> | <a href="#">Test Report</a> |
| transmission_speed | 2021-10-03 10:32:39 | 2021-10-03 10:32:43 | Passed      | Test the transmission speed between the OBU and vOBU | <a href="#">Test Log</a> | <a href="#">Test Report</a> |
| packet_loss        | 2021-10-03 10:32:45 | 2021-10-03 10:33:04 | Passed      | Test the packet loss between the OBU and vOBU        | <a href="#">Test Log</a> | <a href="#">Test Report</a> |
| open_ports         | 2021-10-03 10:33:07 | 2021-10-03 10:33:07 | Failed      | Test the open ports in the OBU VNF                   | <a href="#">Test Log</a> | <a href="#">Test Report</a> |

Figure 5.8: TRVD’s Web Interface Portraiting an Unsuccessful Validation Process.

# testOpenPorts Log

**REPORT**  
Generated  
20211003 11:33:07 UTC+01:00  
4 minutes 35 seconds ago

## Test Statistics

| Total Statistics    | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip                                                     |
|---------------------|-------|------|------|------|----------|------------------------------------------------------------------------|
| All Tests           | 1     | 0    | 1    | 0    | 00:00:00 | <div style="width: 100%; height: 10px; background-color: #f00;"></div> |
| Statistics by Tag   | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip                                                     |
| No Tags             |       |      |      |      |          | <div style="width: 100%; height: 10px; background-color: #ccc;"></div> |
| Statistics by Suite | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip                                                     |
| testOpenPorts       | 1     | 0    | 1    | 0    | 00:00:00 | <div style="width: 100%; height: 10px; background-color: #f00;"></div> |

## Test Execution Log

**SUITE** testOpenPorts 00:00:00.128

**Full Name:** testOpenPorts  
**Source:** /var/lib/jenkins/test\_repository/OBU-vOBU-27/tests/open\_ports/testOpenPorts.robot  
**Start / End / Elapsed:** 20211003 11:33:07.434 / 20211003 11:33:07.562 / 00:00:00.128  
**Status:** 1 test total, 0 passed, 1 failed, 0 skipped

---

**TEST** Testing the open ports 00:00:00.081

**Full Name:** testOpenPorts.Testing the open ports  
**Start / End / Elapsed:** 20211003 11:33:07.480 / 20211003 11:33:07.561 / 00:00:00.081  
**Status:** FAIL  
**Message:**  
 The open ports are not the ones expected

**KEYWORD** \$(open\_ports\_status) = OpenPorts.Open Ports 00:00:00.075

**IF** '\${open\_ports\_status}' == '0' 00:00:00.000

**ELSE IF** '\${open\_ports\_status}' == '1' 00:00:00.000

**ELSE IF** '\${open\_ports\_status}' == '2' 00:00:00.000

**ELSE IF** '\${open\_ports\_status}' == '3' 00:00:00.002

**Start / End / Elapsed:** 20211003 11:33:07.558 / 20211003 11:33:07.560 / 00:00:00.002

**KEYWORD** BuiltIn.Fail The open ports are not the ones expected 00:00:00.001

**Documentation:** Fails the test with the given message and optionally alters its tags.  
**Start / End / Elapsed:** 20211003 11:33:07.558 / 20211003 11:33:07.559 / 00:00:00.001  
 11:33:07.559 FAIL The open ports are not the ones expected

**ELSE** 00:00:00.000

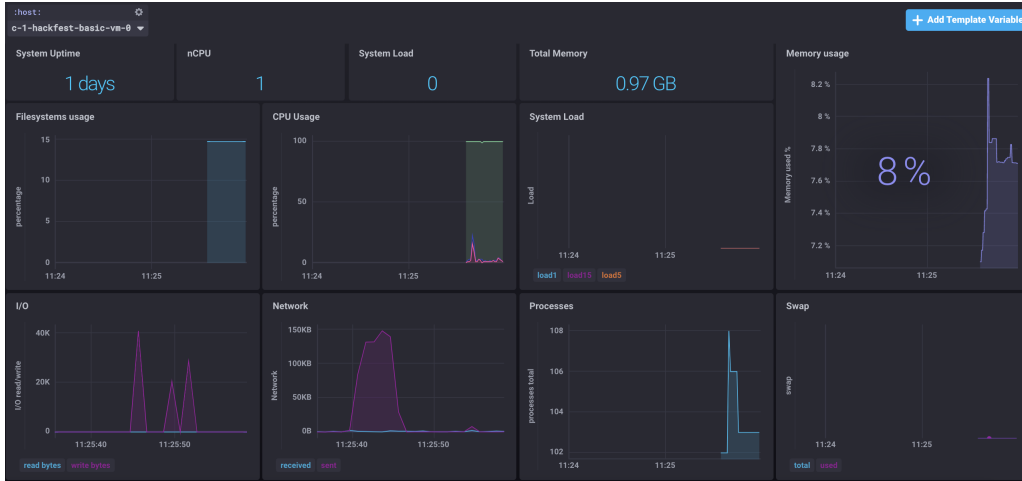
Figure 5.9: report.html Robot File for the Unsuccessful Execution of the Open Ports Test.

```
[Pipeline] sh
+ python3 -m robot.run -d /var/lib/jenkins/test_results/OBU-vOBU-18/open_ports /var/lib/
=====
testOpenPorts
=====
Testing the open ports | FAIL |
The open ports are not the ones expected
-----
testOpenPorts | FAIL |
1 test, 0 passed, 1 failed
-----
Output: /var/lib/jenkins/test_results/OBU-vOBU-18/open_ports/output.xml
Log: /var/lib/jenkins/test_results/OBU-vOBU-18/open_ports/log.html
Report: /var/lib/jenkins/test_results/OBU-vOBU-18/open_ports/report.html
```

Figure 5.10: Portion of the Console Log of an Unsuccessful Validation Process Execution.

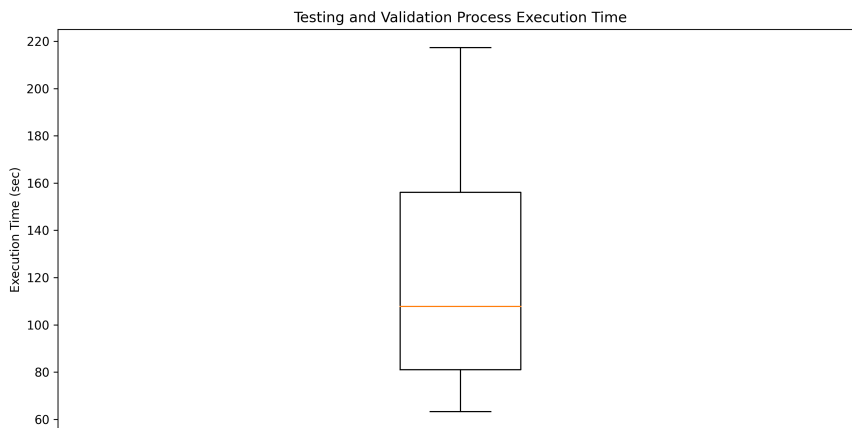
During the validation processes originated by the Testing Descriptors A and B, several metrics were collected. These metrics were collected and stored in the Metrics Repository. Fig. 5.11 depicts the information available in its dashboard.

From this figure, it is possible to conclude that all the defined metrics were collected correctly, being continuously collected from the VNFs, during the validation process.



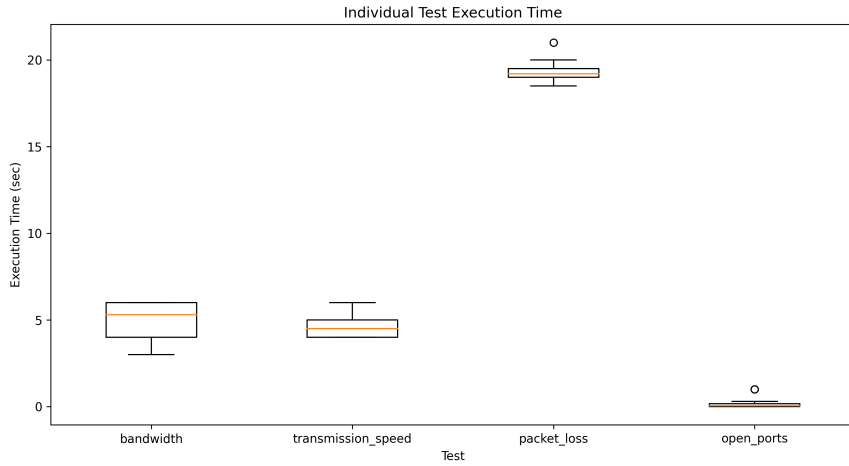
**Figure 5.11:** Metrics Repository Dashboard.

The last aspect that is still missing to be evaluated is the execution times. To get the overall validation process execution time and the execution time of each individual test, twenty validation processes were triggered using the Testing Descriptor A. Then, the results were gathered and analyzed. Fig. 5.12 presents a box plot portraying the overall validation process execution time. The median overall execution time is 107.78 seconds, which is an excellent result since it indicates that the validation process takes less than two minutes.



**Figure 5.12:** Overall Validation Process Execution Time.

The overall execution time is dependent on the individual test execution times, which are presented in Fig. 5.13.



**Figure 5.13:** Execution Time for Each Individual Test.

Table 5.6 summarizes the median of the execution times.

| Test               | Median of the Test Execution Times (sec) |
|--------------------|------------------------------------------|
| bandwidth          | 5.311                                    |
| transmission_speed | 4.509                                    |
| packet_loss        | 19.201                                   |
| open_ports         | 0.055                                    |

**Table 5.6:** Median of Each Test Execution Times.

Given the results presented in Table 5.6, it is possible to conclude that the test that has a longer execution time is the *packet\_loss* test. This is easily understandable since, to ensure a correct calculus of the percentage of lost packets, several batches of packets must be exchanged between VNFs. Overall, the median execution time of all the tests combined is 29.076 seconds.

In the future, with the development of new tests, the overall validation time might be increased, trading lower execution times with higher test coverage.

Given all the correct outputs and results, it is possible to conclude that the CI/CD Service, without a doubt, generates added value, being capable of validating a NetApp's VNFs, assuring a better development lifecycle, without compromising the development's lifecycle duration.

### 5.3 CHAPTER SUMMARY

This chapter started by addressing the results and evaluation of the Descriptors Validator Service. In Section 5.1, the main usage scenarios of this tool were addressed, and Section 5.2 described the methodology for testing and evaluating this Service.

The defined methodology defined two individual evaluations: (i) the evaluation of the validation of the descriptors and (ii) the evaluation of the correction suggestions for the invalid descriptors.

To evaluate the classification performed by the Descriptors Validator Service, a mixed subset of VNF and NS descriptors was used. This subset was composed of fifty-one descriptors - thirty-six valid descriptors and fifteen invalid ones - and allowed to validate if the classification of the descriptors - valid or invalid - was the one expected. During this evaluation, several performance metrics were gathered. From these metrics, it was possible to conclude that even for descriptors with a large size, the overall processing time was inferior to nine seconds, while descriptors with small sizes were validated in less than four seconds.

The second phase of the Descriptors Validator Service's evaluation addressed the Correction Suggestions module. Here, a different subset of descriptors was used. This subset comprised fifteen invalid descriptors with an equal proportion of syntactical, semantic, and reference errors. After obtaining the Correction Suggestions module's results, the descriptors were updated accordingly and onboarded to OSM to validate if the correction suggestions were correct. All the corrected descriptors were onboarded correctly, demonstrating the quality of the correction suggestions outputted by the Correction Suggestions module.

The time needed to achieve the correction suggestions was also analyzed and presented in Table 5.5.

After presenting the results and evaluation of the Descriptors Validator Service, this chapter presented the evaluation and results obtained by the CI/CD Service. It started by addressing that this Service had not been deployed in a production environment since it was still not integrated with the 5GASP NODS and then presented a testing scenario to evaluate it. This testing scenario mimicked several actions that the NODS would perform, which enabled the creation of a more real-world-like scenario.

To evaluate the CI/CD Service, two different Network Services were deployed via OSM Release 9. To perform the validation of these two NSs, three testing descriptors were created. Two of these descriptors triggered a validation process, while the remaining one was used to evaluate the structural validation process of the Testing Descriptors.

From the validation processes that were triggered, it was possible to confirm that the LTR tests were correctly being gathered and performed on the NetApps and that the outputs and results of the tests were being obtained and made available to the NetApp developers. Besides, the metrics collection process was also successful.

The overall validation process execution time was also analyzed, and it was possible to conclude that the median execution time was 107.78 seconds.

Overall, both the results gathered from the Descriptors Validator Service and the CI/CD Service demonstrated the good performance of these tools, which enables their usage outside the initially defined usage scenarios.

# Conclusions

## 6.1 CONCLUSIONS

With the vast migration to a NFV environment, several concerns arose. Network operators have to assure the reliability and correct behavior of Virtualized Network Functions to make sure these are compliant with the defined SLAs.

Although, in the past, the validation of VNFs was performed manually, the current paradigm strives for automation mechanisms to perform these operations. Projects like 5GTANGO, 5G-EVE, 5GinFIRE, and 5GASP, for instance, are proof that an enormous effort is being made to achieve complete automation in the process of validating NetApps.

The work presented in this document is aligned with the 5GASP's goals, and targets two main goals: (i) the creation of VNF and NS Package Validator and (ii) the development of a fully automated NetApp validation pipeline.

Regarding the creation of a VNF and NS Package Validator, only the descriptors validation was addressed. A Descriptors Validation Service was designed and implemented. This Service aimed for modularity and decoupling, which enables it to be easily extended and used in different scenarios. It targeted different Information Models for different MANO frameworks, although the implementation presented in this document only addressed the OSM's YANG IMs.

Despite not being deployed in a production scenario, the Descriptors Validation Service was validated using several OSM Hackfests descriptors. The results obtained support that most NetApp developers may use this tool since the descriptors' outputted classification was always the one expected. Besides, the low classification and correction suggestions generation times point out that the NetApp development process wouldn't be extended with the usage of this tool.

Using this tool, it was also possible to conclude that the OSM IMs are not entirely aligned with the implementation of this framework, which heavily increases the learning curve of new NetApp developers. Although, ETSI is trying to fix these issues, aiming to provide better support for the developers, thus increasing OSM's adoption rates.

Regarding the developed CI/CD Service, it is one of the most critical goals of the 5GASP project. This service aims to provide a fully automated NetApp validation pipeline, and it is expected to address different MANO frameworks. At the time of writing this document, only OSM is being addressed, due to the lack of maturity of the 5GASP project. However, future plans cover including ONAP in the MANO frameworks supported.

The CI/CD Service's results and evaluation indicate that this tool has enormous potential. Its modularity and technology independence are some of the key factors for its success.

When analyzing the CI/CD Service's results, it was possible to conclude that, although its implementation is not completed, it could perform several tests and present their outputs to the NetApp developers, via a completely automated process. The low validation times also support that this service could be deployed in several NetApp-validation-oriented research projects.

In conclusion, the great results gathered from both developed tools heavily support the prospect of their wide adoption in NetApp validation scenarios. Besides, the work produced in this dissertation presents a starting point for advancing the state-of-the-art in the automated validation of 5G NetApps.

## 6.2 FUTURE WORK

Since both the Descriptors Validator Service and the CI/CD Service's development are at an early stage, there are several requirements yet not achieved.

Regarding the Descriptors Validator Service, it is still missing to create a TOSCA IM Wrapper. This would allow the validation of descriptors targeting not only OSM but also ONAP, for instance.

Besides, the Descriptors Validator Service is just a component of a VNF and NS Package Validator, which also aims to validate VNF Juju Charms and perform several security checks. For instance, the possibility of injecting scripts in VNFs, which may affect the underlying infrastructure where they will be deployed. When the VNF and NS Package Validator is entirely developed, in the context of 5GASP, it will have to be integrated with the NODS.

The same happens to the CI/CD Service.

To enable the communication between the NODS and the CI/CD Service, TMF standards, like TMF653 - Service Test Management - could be addressed. The standardization of the communication interfaces of the CI/CD Service is the one the most crucial next steps regarding the development of this Service.

Besides, more predefined-tests must be developed in order to amplify the test coverage of the CI/CD Service. These tests will have to address, not only infrastructure aspects, but also security aspects, since security tests are heavily mentioned in 5GASP's proposal. Support for developer-defined tests will also have to be provided since these will allow tailored testing of a NetApp's behavior.

Another important addition will be the creation of mechanisms to interact with the external VNFs involved in the validation phase. As mentioned before, these will be responsible for creating different testing scenarios emulating real-world situations, for instance, a testing



scenario where the network is heavily congested. Although the 5GASP NODS will manage the deployment of these VNFs, the CI/CD Service will still have to communicate with them to trigger the testing scenarios aforementioned.

Lastly, there are still missing key features in the metrics collection system. For instance, the federation of this service and the implementation of access restrictions to its web interface.



# References

- [1] R. Mijumbi, J. Serrat, J.-I. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, 2016. DOI: 10.1109/MCOM.2016.7378433.
- [2] D. Cotroneo, L. De Simone, A. Iannillo, A. Lanzaro, R. Natella, J. Fan, and W. Ping, "Network Function Virtualization: Challenges and Directions for Reliability Assurance," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, 2014, pp. 37–42. DOI: 10.1109/ISSREW.2014.48.
- [3] M. Chiosi and et all, "White paper: Network Functions Virtualisation - An Introduction, Benefits, Enablers, Challenges & Call for Action," Tech. Rep. 1, Oct. 2012.
- [4] ETSI ISG NFV, "ETSI GS NFV 002 V1.1.1: Network Function Virtualisation (NFV): Architectural Framework," Tech. Rep. 1, Oct. 2013.
- [5] NFV White Paper, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1," Oct. 2012.
- [6] *5GASP*, Accessed: 07-08-2021. [Online]. Available: <https://5g-ppp.eu/5gasp/>.
- [7] C. Tipantuña and P. Yanchapaxi, "Network functions virtualization: An overview and open-source projects," in *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, 2017, pp. 1–6. DOI: 10.1109/ETCM.2017.8247541.
- [8] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016. DOI: 10.1109/COMST.2015.2477041.
- [9] S. Chayapathi Rajendra and. Farrukh Hassan and P. Shah, *Network Functions Virtualizations (NFV) with a touch of SDN*. Pearson Education, Inc, 2017, ISBN: 0134463056.
- [10] *Network Functions Virtualisation (NFV)*, Accessed: 20-07-2021. [Online]. Available: <https://www.etsi.org/technologies/nfv>.
- [11] ETSI ISG NFV, "ETSI GS NFV 003 V1.5.1: Network Function Virtualisation (NFV): Terminology for main concepts in NFV," Tech. Rep. 1, Jan. 2020.
- [12] S. Zhang, "An Overview of Network Slicing for 5G," *IEEE Wireless Communications*, vol. 26, no. 3, pp. 111–117, 2019. DOI: 10.1109/MWC.2019.1800234.
- [13] M. Abu-Lebdeh, D. Naboulsi, R. Glitho, C. Wette, and Tchouati, "NFV Orchestrator Placement for Geo-Distributed Systems," Nov. 2017. DOI: 10.1109/NCA.2017.8171391.
- [14] ETSI ISG NFV, "ETSI GS NFV-MAN 001 - V1.1.1 - Network Functions Virtualisation (NFV); Management and Orchestration," Tech. Rep. 1, Dec. 2014.
- [15] C. Parada, J. Bonnet, E. Fotopoulou, A. Zafeiropoulos, E. Kapassa, M. Touloupou, D. Kyriazis, R. Vilalta, R. Muñoz, R. Casellas, R. Martínez, and G. Xilouris, "5GTANGO: A Beyond-Mano Service Platform," in *2018 European Conference on Networks and Communications (EuCNC)*, 2018, pp. 26–30. DOI: 10.1109/EuCNC.2018.8443232.
- [16] *ONAP - About*, Accessed: 21-07-2021. [Online]. Available: <https://www.onap.org/about>.

- [17] *ONAP - Platform Architecture*, Accessed: 21-07-2021. [Online]. Available: <https://www.onap.org/architecture>.
- [18] *SONATA - NFV Platform*, Accessed: 22-07-2021. [Online]. Available: <https://www.sonata-nfv.eu/>.
- [19] T. Soenen, S. Van Rossem, W. Tavernier, F. Vicens, D. Valocchi, P. Trakadas, P. Karkazis, G. Xilouris, P. Eardley, S. Kolometsos, M.-A. Kourtis, D. Guija, S. Siddiqui, P. Hasselmeyer, J. Bonnet, and D. Lopez, "Insights from SONATA: Implementing and integrating a microservice-based NFV service platform with a DevOps methodology," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–6. DOI: 10.1109/NOMS.2018.8406139.
- [20] S. Dräxler, H. Karl, M. Peuster, H. R. Kouchaksaraei, M. Bredel, J. Lessmann, T. Soenen, W. Tavernier, S. Mendel-Brin, and G. Xilouris, "SONATA: Service programming and orchestration for virtualized software networks," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2017, pp. 973–978. DOI: 10.1109/ICCW.2017.7962785.
- [21] *Open Source Mano*, Accessed: 22-07-2021. [Online]. Available: <https://osm.etsi.org/>.
- [22] OSM - Technical Steering Committee, "OSM Release NINE - Release Notes," Tech. Rep. 1, Dec. 2020.
- [23] L. Mamushiane, A. A. Lysko, T. Mukute, J. Mwangama, and Z. D. Toit, "Overview of 9 Open-Source Resource Orchestrating ETSI MANO Compliant Implementations: A Brief Survey," in *2019 IEEE 2nd Wireless Africa Conference (WAC)*, 2019, pp. 1–7. DOI: 10.1109/AFRICA.2019.8843421.
- [24] M.-I. Csoma, B. Koné, R. Botez, I.-A. Ivanciu, A. Kora, and V. Dobrota, "Management and Orchestration for Network Function Virtualization: An Open Source MANO Approach," in *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, 2020, pp. 1–6. DOI: 10.1109/RoEduNet51892.2020.9324847.
- [25] ETSI, "OSM Release ONE - Release Notes," Tech. Rep. 1, Oct. 2016.
- [26] *cloud-init 21.2 Documentation*, Accessed: 23-07-2021. [Online]. Available: <https://cloudinit.readthedocs.io/en/21.2/>.
- [27] —, "OSM Scope, Functionality, Operation and Integration Guidelines," Tech. Rep. 1, Dec. 2019.
- [28] —, "OSM Information Model - Release 2," Tech. Rep. 1, Jul. 2017.
- [29] Y. Girma Mamuye, F. Yousaf, V. Sciancalepore, and X. Costa-Pérez, *Benchmarking Open-Source NFV MANO Systems: OSM and ONAP*, Mar. 2020.
- [30] T. Panagiotis, K. Panagiotis, L. Helen, T. Zahariadis, F. Vicens, A. Zurita, P. Alemany, T. Soenen, C. Parada, J. Bonnet, E. Fotopoulou, A. Zafeiropoulos, E. Kapassa, M. Touloupou, and D. Kyriazis, "Comparison of Management and Orchestration Solutions for the 5G Era," *Journal of Sensor and Actuator Networks*, vol. 9, p. 4, Jan. 2020. DOI: 10.3390/jsan9010004.
- [31] ETSI, "OSM Release 5 - Technical Overview," Tech. Rep. 1, Jan. 20179.
- [32] *OSM Release 6*, Accessed: 02-08-2021. [Online]. Available: [https://osm.etsi.org/wikipub/index.php/OSM\\_Release\\_SIX](https://osm.etsi.org/wikipub/index.php/OSM_Release_SIX).
- [33] ETSI ISG NFV, "OSM Release SEVEN - Release Notes," Tech. Rep. 1, Jun. 2021.
- [34] —, "OSM Release EIGHT - Release Notes," Tech. Rep. 1, Jul. 2020.
- [35] —, "OSM Release NINE - Release Notes," Tech. Rep. 1, Dec. 2020.
- [36] *JuJu Charms*, Accessed: 02-08-2021. [Online]. Available: <https://jaas.ai/docs/what-is-juju>.
- [37] M. Gharbaoui, B. Martini, and P. Castoldi, "Programmable and Automated Deployment of Tenant-Managed SDN Network Slices," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–6. DOI: 10.1109/NOMS47738.2020.9110302.
- [38] *Creating your VNF Charm*, Accessed: 02-08-2021. [Online]. Available: [https://osm.etsi.org/wikipub/index.php/Creating\\_your\\_VNF\\_Charm](https://osm.etsi.org/wikipub/index.php/Creating_your_VNF_Charm).

- [39] M. Gokarna and R. Singh, “DevOps: A Historical Review and Future Works,” in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, 2021, pp. 366–371. DOI: 10.1109/ICCCIS51004.2021.9397235.
- [40] S. Zhong, C. Liping, and C. Tian-en, “Agile planning and development methods,” in *2011 3rd International Conference on Computer Research and Development*, vol. 1, 2011, pp. 488–491. DOI: 10.1109/ICCRD.2011.5764064.
- [41] M. Virmani, “Understanding DevOps and bridging the gap from continuous integration to continuous delivery,” in *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, 2015, pp. 78–82. DOI: 10.1109/INTECH.2015.7173368.
- [42] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, “What is DevOps? A Systematic Mapping Study on Definitions and Practices,” in *Proceedings of the Scientific Workshop Proceedings of XP2016*, ser. XP ’16 Workshops, Edinburgh, Scotland, UK: Association for Computing Machinery, 2016, ISBN: 9781450341349. DOI: 10.1145/2962695.2962707. [Online]. Available: <https://doi.org/10.1145/2962695.2962707>.
- [43] S. W. Hussaini, “Strengthening harmonization of Development (Dev) and Operations (Ops) silos in IT environment through systems approach,” in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014, pp. 178–183. DOI: 10.1109/ITSC.2014.6957687.
- [44] *DevOps Days Ghent*, Accessed: 02-08-2021. [Online]. Available: <https://legacy.devopsdays.org/events/2009-ghent/>.
- [45] P. Agrawal and N. Rawat, “Devops, A New Approach To Cloud Development & Testing,” in *2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, vol. 1, 2019, pp. 1–4. DOI: 10.1109/ICICT46931.2019.8977662.
- [46] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices,” *IEEE Access*, vol. 5, pp. 3909–3943, 2017. DOI: 10.1109/ACCESS.2017.2685629.
- [47] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Conallen, and K. Houston, *Object-Oriented Analysis and Design with Applications, Third Edition*, Third. Addison-Wesley Professional, 2007, ISBN: 9780201895513.
- [48] P. Duvall, S. M. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*, ser. Addison-Wesley Signature Series. Upper Saddle River, NJ: Addison-Wesley, 2007, ISBN: 978-0-321-33638-5. [Online]. Available: <http://my.safaribooksonline.com/9780321336385>.
- [49] *SDN/NFV DevOps: Release Automation for Network Operators*, Accessed: 03-08-2021. [Online]. Available: <https://devops.com/sdn-nfv-devops-release-automation-for-network-operators/>.
- [50] *Jenkins*, Accessed: 08-08-2021. [Online]. Available: <https://jenkins.io/>.
- [51] *Circle CI*, Accessed: 08-08-2021. [Online]. Available: <https://circleci.com/>.
- [52] *TeamCity*, Accessed: 08-08-2021. [Online]. Available: <https://www.jetbrains.com/teamcity/>.
- [53] *Gitlab CI*, Accessed: 08-08-2021. [Online]. Available: <https://docs.gitlab.com/ee/ci/>.
- [54] *Travis CI*, Accessed: 08-08-2021. [Online]. Available: <https://travis-ci.com/>.
- [55] *Drone CI*, Accessed: 08-08-2021. [Online]. Available: <https://drone.io/>.
- [56] *Bamboo*, Accessed: 08-08-2021. [Online]. Available: <https://www.atlassian.com/software/bamboo>.
- [57] *Katalon Studio*, Accessed: 23-08-2021. [Online]. Available: <http://www.cnn.com/2017/04/18/us/75th-anniversary-doolittle-raid>.
- [58] *Robot Framework*, Accessed: 23-08-2021. [Online]. Available: <https://robotframework.org/>.
- [59] *Selenium*, Accessed: 23-08-2021. [Online]. Available: <https://www.selenium.dev/about/>.
- [60] *Selenium*, Accessed: 23-08-2021. [Online]. Available: <https://www.selenium.dev/>.

- [61] *Eggplant*, Accessed: 23-08-2021. [Online]. Available: <https://www.eggplantsoftware.com/test-automation>.
- [62] *Citrus*, Accessed: 23-08-2021. [Online]. Available: <https://citrusframework.org/>.
- [63] *TICK Stack*, Accessed: 24-08-2021. [Online]. Available: <https://www.influxdata.com/time-series-platform/>.
- [64] *Prometheus*, Accessed: 24-08-2021. [Online]. Available: <https://prometheus.io/>.
- [65] *Grafana*, Accessed: 24-08-2021. [Online]. Available: <https://grafana.com/>.
- [66] *ELK Stack*, Accessed: 24-08-2021. [Online]. Available: <https://www.elastic.co/what-is/elk-stack>.
- [67] *5GTANGO*, Accessed: 02-08-2021. [Online]. Available: <https://www.5gtango.eu/>.
- [68] *5G EVE*, Accessed: 02-08-2021. [Online]. Available: <https://www.5g-eve.eu/>.
- [69] *5GinFIRE*, Accessed: 02-08-2021. [Online]. Available: <https://5ginfire.eu/>.
- [70] *5G-VINNI*, Accessed: 07-08-2021. [Online]. Available: <https://5g-ppp.eu/5g-vinni/>.
- [71] C. Tranoris, *Openslice: An opensource OSS for Delivering Network Slice as a Service*, 2021. arXiv: 2102.03290 [cs.NI].
- [72] *Openslice*, Accessed: 07-08-2021. [Online]. Available: <https://openslice.readthedocs.io/en/stable/>.
- [73] P. Twamley, M. Müller, P.-B. Bök, G. K. Xilouris, C. Sakkas, M. A. Kourtis, M. Peuster, S. Schneider, P. Stavrianos, and D. Kyriazis, “5GTANGO: An Approach for Testing NFV Deployments,” in *2018 European Conference on Networks and Communications (EuCNC)*, 2018, pp. 1–218. DOI: 10.1109/EuCNC.2018.8442844.
- [74] 5GTANGO, “D4.2 Final release of the service validation SDK toolset,” Tech. Rep. 1, Jun. 2019.
- [75] W. Nakimuli, G. Landi, R. Perez, M. Pergolesi, M. Molla, C. Ntogkas, G. Garcia-Aviles, J. Garcia-Reinoso, M. Femminella, P. Serrano, F. Lombardo, J. Rodriguez, G. Reali, and S. Salsano, “Automatic deployment, execution and analysis of 5G experiments using the 5G EVE platform,” in *2020 IEEE 3rd 5G World Forum (5GWF)*, 2020, pp. 372–377. DOI: 10.1109/5GWF49715.2020.9221060.
- [76] 5G EVE, “D5.8 - Testing and validation methodologies final report with the final version of testing and validation suite,” Tech. Rep. 1, May 2021.
- [77] —, “D5.9 - Testing, Validation, and Performance Diagnosis Methodologies User Manual,” Tech. Rep. 1, May 2021.
- [78] *C\_5.1\_SmartCity\_CNIT Test Case Blueprint*, Accessed: 06-08-2021. [Online]. Available: [https://github.com/5GEVE/blueprint-yaml/blob/master/UC%5C\\_5.1%5C\\_SmartCity%5C\\_CNIT/tcb%5C\\_cnit%5C\\_smart%5C\\_city%5C\\_2.yaml](https://github.com/5GEVE/blueprint-yaml/blob/master/UC%5C_5.1%5C_SmartCity%5C_CNIT/tcb%5C_cnit%5C_smart%5C_city%5C_2.yaml).
- [79] *UC\_2.2\_SmartTourism\_SEGITTUR Experiment Blueprint*, Accessed: 06-08-2021. [Online]. Available: [https://github.com/5GEVE/blueprint-yaml/blob/master/UC%5C\\_2.2%5C\\_SmartTourism%5C\\_SEGITTUR/segittur%5C\\_camerachange%5C\\_exp%5C\\_v10.yml](https://github.com/5GEVE/blueprint-yaml/blob/master/UC%5C_2.2%5C_SmartTourism%5C_SEGITTUR/segittur%5C_camerachange%5C_exp%5C_v10.yml).
- [80] *5GinFIRE*, Accessed: 07-08-2021. [Online]. Available: <https://cordis.europa.eu/project/id/732497>.
- [81] A. P. Silva, C. Tranoris, S. Denazis, S. Sargento, J. Pereira, M. Luís, R. Moreira, F. Silva, I. Vidal, B. Nogales, R. Nejabati, and D. Simeonidou, “5GinFIRE: An end-to-end open5G vertical network function ecosystem,” *Ad Hoc Networks*, vol. 93, 2019, ISSN: 1570-8705. DOI: <https://doi.org/10.1016/j.adhoc.2019.101895>.
- [82] 5GinFIRE, “D3.2 - 5G Experimentation portal, tools and middleware,” Tech. Rep. 1, Jul. 2018.
- [83] *Open API Manifesto*, Accessed: 08-08-2021. [Online]. Available: <https://www.tmforum.org/open-apis/open-api-manifesto/>.
- [84] TM Forum, “Service Catalog Management - API REST Specification,” Tech. Rep. 1, Oct. 2020.

- [85] *Pyang - README*, Accessed: 21-08-2021. [Online]. Available: <https://github.com/mbj4668/pyang/blob/master/README.md>.
- [86] *Web Framework Benchmarks*, Accessed: 29-08-2021. [Online]. Available: <https://www.techempower.com/benchmarks/#section=data-r20&hw=ph&test=query&a=2&f=zhb2tb-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-ziimf3-zik0zj-zik0zj-zik0zj-cn3>.
- [87] 5GASP Consortium, “5GASP Project Proposal,” Tech. Rep., Jun. 2020.
- [88] 5GASP, “D2.1 - Architecture, Model Entities Specification and Design,” Tech. Rep. 1, Jul. 2021.
- [89] —, “D5.1 - Initial Report on Test-Plan Creation and Testing Methodologies,” Tech. Rep. 1, Oct. 2021.





# Appendix

## DESCRIPTORS VALIDATOR SERVICE CLASSIFICATION RESULTS

| IM    | Descriptor                                            | Descriptor Status | Obtained Classification | Errors Found                                                                                                                                                                                             |
|-------|-------------------------------------------------------|-------------------|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| osm_5 | invalid-reference_osm5_vnfd-frvnf_vnfd.yaml           | Invalid           | Invalid                 | Reference errors: The value 'vnf-mgt' on vnfd:vnfd-catalog/vnfd/mgmt-interface/cp doesn't match one of its reference (vnfd:vnfd-catalog/vnfd/connection-point/name) values of 'vnf-mgmt2', 'vnf-public'. |
| osm_5 | invalid-semantics_osm5_nsd-ubuntu-cloudinit_nsd.yaml  | Invalid           | Invalid                 | Semantic errors: The value 'ELANS' on nsd:nsd-catalog/nsd/vld/type doesn't match one of its allowed values (['ELAN', 'ELINE', 'ETREE']).                                                                 |
| osm_5 | invalid-syntax_osm5_vnfd-vnf_nsh_vnfd.yaml            | Invalid           | Invalid                 | Syntactic errors: Invalid Tag vnfd:vnfd-catalog/vnfd/connection-point/types.                                                                                                                             |
| osm_5 | valid_osm5_nsd-ubuntu-cloudinit_nsd.yaml              | Valid             | Valid                   |                                                                                                                                                                                                          |
| osm_5 | valid_osm5_nsd-vyos_hnf_nsd.yaml                      | Valid             | Valid                   |                                                                                                                                                                                                          |
| osm_5 | valid_osm5_nsd-webserver_vimmetric_autoscale_nsd.yaml | Valid             | Valid                   |                                                                                                                                                                                                          |

|       |                                                           |         |         |                                                                                                                                                                                                         |
|-------|-----------------------------------------------------------|---------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| osm_5 | valid_osm5_vnfd-cirros_vnfd.yaml                          | Valid   | Valid   |                                                                                                                                                                                                         |
| osm_5 | valid_osm5_vnfd-frrvnf_vnfd.yaml                          | Valid   | Valid   |                                                                                                                                                                                                         |
| osm_5 | valid_osm5_vnfd-vnf_nsh_vnfd.yaml                         | Valid   | Valid   |                                                                                                                                                                                                         |
| osm_6 | invalid-reference_osm6_vnfd-hackfest_proxycharm_vnfd.yaml | Invalid | Invalid | Reference errors: The value 'vnf-mgmts' on vnf:vnfd-catalog/vnfd/mgmt-interface/cp doesn't match one of it's reference (vnfd:vnfd-catalog/vnfd/connection-point/name) values of 'vnf-data', 'vnf-mgmt'. |
| osm_6 | invalid-semantics_osm6_vnfd-clean_vnfd.yaml               | Invalid | Invalid | Semantic errors: The value 'first' on vnf:vnfd-catalog/vnfd/vdu/count doesn't match it's datatype (uint64).                                                                                             |
| osm_6 | invalid-syntax_osm6_nsd-cirros_vdu_alarm_nsd.yaml         | Invalid | Invalid | Syntactic errors: Invalid Tag nsd:nsd-catalog/nsd/constituent-vnfs.                                                                                                                                     |
| osm_6 | valid_osm6_nsd-cirros_vdu_alarm_nsd.yaml                  | Valid   | Valid   |                                                                                                                                                                                                         |
| osm_6 | valid_osm6_nsd-hackfest-basic-ns-sriov.yaml               | Valid   | Valid   |                                                                                                                                                                                                         |
| osm_6 | valid_osm6_nsd-hackfest_epa_nsd.yaml                      | Valid   | Valid   |                                                                                                                                                                                                         |
| osm_6 | valid_osm6_nsd-hackfest_simplecharm_nsd.yaml              | Valid   | Valid   |                                                                                                                                                                                                         |
| osm_6 | valid_osm6_nsd-native_nsd.yaml                            | Valid   | Valid   |                                                                                                                                                                                                         |
| osm_6 | valid_osm6_nsd-ubuntuvnfvnfmetric_autoscale_nsd.yaml      | Valid   | Valid   |                                                                                                                                                                                                         |
| osm_6 | valid_osm6_vnfd-clean_vnfd.yaml                           | Valid   | Valid   |                                                                                                                                                                                                         |

|       |                                                                                    |         |         |                                                                                                                                                                                                                 |
|-------|------------------------------------------------------------------------------------|---------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| osm_6 | valid_osm6_vnfd-<br>endpoint_vnfd.yaml                                             | Valid   | Valid   |                                                                                                                                                                                                                 |
| osm_6 | valid_osm6_vnfd-<br>hackfest-<br>basic_vnfd.yaml                                   | Valid   | Valid   |                                                                                                                                                                                                                 |
| osm_6 | valid_osm6_vnfd-<br>hackfest_proxycharm_<br>vnfd.yaml                              | Valid   | Valid   |                                                                                                                                                                                                                 |
| osm_6 | valid_osm6_vnfd-<br>hackfest_simplecharm_<br>vnfd.yaml                             | Valid   | Valid   |                                                                                                                                                                                                                 |
| osm_6 | valid_osm6_vnfd-<br>vnf_mpls_vnfd.yaml                                             | Valid   | Valid   |                                                                                                                                                                                                                 |
| osm_7 | invalid-<br>reference_osm7_vnfd-<br>hackfest_simple_<br>na-<br>tivecharm_vnfd.yaml | Invalid | Invalid | Reference errors: The value 'vnf-mgmt-network' on vnfd:vnfd-catalog/vnfd/mgmt-interface/cp doesn't match one of it's reference (vnfd:vnfd-catalog/vnfd/connection-point/name) values of 'vnf-data', 'vnf-mgmt'. |
| osm_7 | invalid-<br>semantics_osm7_vnfd-<br>hackfest_simple_<br>na-<br>tivecharm_vnfd.yaml | Invalid | Invalid | Semantic errors: The value 'EXTERNAL' on vnfd:vnfd-catalog/vnfd/vdu/interface/type doesn't match one of it's allowed values (['INTERNAL', 'EXTERNAL']).                                                         |
| osm_7 | invalid-<br>syntax_osm7_nsd-<br>hackfest_simple_k8s_<br>nsd.yaml                   | Invalid | Invalid | Syntactic errors: Invalid Tag nsd:nsd-catalog/nsd/shortname.                                                                                                                                                    |
| osm_7 | valid_osm7_nsd-<br>hackfest_simple_k8s_<br>nsd.yaml                                | Valid   | Valid   |                                                                                                                                                                                                                 |
| osm_7 | valid_osm7_nsd-<br>hackfest_simple_<br>proxycharm_nsd.yaml                         | Valid   | Valid   |                                                                                                                                                                                                                 |

|       |                                                        |         |         |                                                                                                                                                                                                          |
|-------|--------------------------------------------------------|---------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| osm_7 | valid_osm7_vnfd-hackfest_simple_na-tivecharm_vnfd.yaml | Valid   | Valid   |                                                                                                                                                                                                          |
| osm_7 | valid_osm7_vnfd-hackfest_simple_proxycharm_vnfd.yaml   | Valid   | Valid   |                                                                                                                                                                                                          |
| osm_8 | invalid-reference_osm8_vnfd_vnf_mpls_vnfd.yaml         | Invalid | Invalid | Reference errors: The value 'vnf-cp1' on vnf:vnfd-catalog/vnfd/vdu/interface/external-connection-point-ref doesn't match it's reference (vnfd:vnfd-catalog/vnfd/connection-point/name) value of vnf-cp0. |
| osm_8 | invalid-semantics_osm8_vnfd-endpoint_vnfd.yaml         | Invalid | Invalid | Semantic errors: The value '1 gigabyte' on vnf:vnfd-catalog/vnfd/vdu/vm-flavor/memory-mb doesn't match it's datatype (uint64).                                                                           |
| osm_8 | invalid-syntax_osm8_nsd-hackfest_epa_nsd.yaml          | Invalid | Invalid | Syntactic errors: Invalid Tag nsd:nsd-catalog/nsd/vlds.                                                                                                                                                  |
| osm_8 | valid_osm8_nsd-hackfest-basic-ns-sriov.yaml            | Valid   | Valid   |                                                                                                                                                                                                          |
| osm_8 | valid_osm8_nsd-hackfest_epa_nsd.yaml                   | Valid   | Valid   |                                                                                                                                                                                                          |
| osm_8 | valid_osm8_nsd-native_nsd.yaml                         | Valid   | Valid   |                                                                                                                                                                                                          |
| osm_8 | valid_osm8_vnfd-endpoint_vnfd.yaml                     | Valid   | Valid   |                                                                                                                                                                                                          |
| osm_8 | valid_osm8_vnfd-hackfest_proxycharm_vnfd.yaml          | Valid   | Valid   |                                                                                                                                                                                                          |
| osm_8 | valid_osm8_vnfd_vnf_mpls_vnfd.yaml                     | Valid   | Valid   |                                                                                                                                                                                                          |

**Table 1:** Descriptors Validator Service Classification Results.

DESCRIPTORS VALIDATOR SERVICE PROCESSING TIMES

| IM    | Descriptor                                                | File Size (bytes) | Total Processing Time (ms) | Validation Time (ms) |
|-------|-----------------------------------------------------------|-------------------|----------------------------|----------------------|
| osm_5 | invalid-reference_osm5_vnfd_frrvnf_vnfd.yaml              | 2904              | 1618.54                    | 9.60                 |
| osm_5 | invalid-semantics_osm5_nsd_ubuntu-cloudinit_nsd.yaml      | 766               | 941.92                     | 1.02                 |
| osm_5 | invalid-syntax_osm5_vnfd_vnf_nsh_vnfd.yaml                | 706               | 1764.9                     | 1.55                 |
| osm_5 | valid_osm5_nsd_ubuntu-cloudinit_nsd.yaml                  | 747               | 1552.69                    | 2.05                 |
| osm_5 | valid_osm5_nsd_vyos_hnf_nsd.yaml                          | 832               | 989.88                     | 1.26                 |
| osm_5 | valid_osm5_nsd_webserver_vimmetric_autoscale_nsd.yaml     | 1423              | 1377.06                    | 3.74                 |
| osm_5 | valid_osm5_vnfd_cirros_vnfd.yaml                          | 1378              | 1518.19                    | 3.38                 |
| osm_5 | valid_osm5_vnfd_frrvnf_vnfd.yaml                          | 2903              | 2126.21                    | 7.80                 |
| osm_5 | valid_osm5_vnfd_vnf_nsh_vnfd.yaml                         | 705               | 1385.82                    | 2.54                 |
| osm_6 | invalid-reference_osm6_vnfd_hackfest_proxycharm_vnfd.yaml | 2251              | 1633.55                    | 7.28                 |
| osm_6 | invalid-semantics_osm6_vnfd_clean_vnfd.yaml               | 1509              | 1626.04                    | 2.95                 |
| osm_6 | invalid-syntax_osm6_nsd_cirros_vdu_alarm_nsd.yaml         | 1422              | 1620.74                    | 2.52                 |
| osm_6 | valid_osm6_nsd_cirros_vdu_alarm_nsd.yaml                  | 1421              | 825.78                     | 1.39                 |
| osm_6 | valid_osm6_nsd_hackfest-basic-nssriov.yaml                | 846               | 1096.61                    | 1.50                 |
| osm_6 | valid_osm6_nsd_hackfest_epa_nsd.yaml                      | 689               | 3616.0                     | 6.14                 |

|       |                                                                           |      |         |      |
|-------|---------------------------------------------------------------------------|------|---------|------|
| osm_6 | valid_osm6_nsd_hackfest_<br>simplecharm_nsd.yaml                          | 764  | 703.32  | 1.46 |
| osm_6 | valid_osm6_nsd_native_nsd.yaml                                            | 968  | 1294.64 | 2.05 |
| osm_6 | valid_osm6_nsd_ubuntuvm_<br>vnfmetric_autoscale_nsd.yaml                  | 801  | 918.2   | 1.61 |
| osm_6 | valid_osm6_vnfd_clean_vnfd.yaml                                           | 1505 | 2168.22 | 3.96 |
| osm_6 | valid_osm6_vnfd_endpoint_<br>vnfd.yaml                                    | 1691 | 2753.74 | 8.53 |
| osm_6 | valid_osm6_vnfd_hackfest-<br>basic_vnfd.yaml                              | 889  | 1241.96 | 1.74 |
| osm_6 | valid_osm6_vnfd_hackfest_<br>proxycharm_vnfd.yaml                         | 2248 | 1516.59 | 3.13 |
| osm_6 | valid_osm6_vnfd_hackfest_<br>simplecharm_vnfd.yaml                        | 1870 | 1833.86 | 6.74 |
| osm_6 | valid_osm6_vnfd_vnf_mpls_<br>vnfd.yaml                                    | 808  | 1482.24 | 5.86 |
| osm_7 | invalid-<br>reference_osm7_vnfd_hackfest_<br>simple_nativecharm_vnfd.yaml | 1071 | 2387.59 | 3.81 |
| osm_7 | invalid-<br>semantics_osm7_vnfd_hackfest_<br>simple_nativecharm_vnfd.yaml | 1062 | 3361.75 | 2.07 |
| osm_7 | invalid-<br>syntax_osm7_nsd_hackfest_<br>simple_k8s_nsd.yaml              | 733  | 964.28  | 1.16 |
| osm_7 | valid_osm7_nsd_hackfest_<br>simple_k8s_nsd.yaml                           | 734  | 791.39  | 1.05 |
| osm_7 | valid_osm7_nsd_hackfest_<br>simple_proxycharm_nsd.yaml                    | 844  | 1079.46 | 1.98 |
| osm_7 | valid_osm7_vnfd_hackfest_<br>simple_nativecharm_vnfd.yaml                 | 1063 | 1603.09 | 2.59 |
| osm_7 | valid_osm7_vnfd_hackfest_<br>simple_proxycharm_vnfd.yaml                  | 933  | 1360.59 | 1.92 |
| osm_8 | invalid-<br>reference_osm8_vnfd_vnf_mpls_<br>vnfd.yaml                    | 808  | 3274.13 | 3.69 |
| osm_8 | invalid-<br>semantics_osm8_vnfd_endpoint_<br>vnfd.yaml                    | 1697 | 2678.98 | 2.17 |

|       |                                                       |      |         |      |
|-------|-------------------------------------------------------|------|---------|------|
| osm_8 | invalid-<br>syntax_osm8_nsd_hackfest_epa_<br>nsd.yaml | 690  | 1370.3  | 2.08 |
| osm_8 | valid_osm8_nsd_hackfest-basic-ns-<br>sriov.yaml       | 846  | 1119.52 | 1.53 |
| osm_8 | valid_osm8_nsd_hackfest_epa_<br>nsd.yaml              | 689  | 1109.26 | 2.06 |
| osm_8 | valid_osm8_nsd_native_nsd.yaml                        | 968  | 1252.26 | 4.30 |
| osm_8 | valid_osm8_vnfd_endpoint_<br>vnfd.yaml                | 1691 | 2230.24 | 2.38 |
| osm_8 | valid_osm8_vnfd_hackfest_<br>proxycharm_vnfd.yaml     | 2248 | 1731.76 | 2.27 |
| osm_8 | valid_osm8_vnfd_vnf_mpls_<br>vnfd.yaml                | 808  | 2191.89 | 2.94 |

**Table 2:** Descriptors Validator Service Processing Times.

#### CORRECTION SUGGESTIONS MODULE RESULTS

| <b>IM</b> | <b>Descriptor</b>                                         | <b>Error Found</b>                                         | <b>Correction Suggestions</b>                                                                                                                                                       |
|-----------|-----------------------------------------------------------|------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| osm_8     | invalid-<br>syntax_osm8_nsd_hackfest_<br>epa_nsd.yaml     | Invalid Tag nsd:nsd-<br>catalog/nsd/vlds.                  | ['nsd:nsd-<br>catalog/nsd/vld',<br>'nsd:nsd-<br>catalog/nsd/vld/id',<br>'nsd:nsd-<br>catalog/nsd/id']                                                                               |
| osm_6     | invalid-<br>syntax_osm6_nsd_cirros_<br>vdu_alarm_nsd.yaml | Invalid Tag nsd:nsd-<br>catalog/nsd/constituent-<br>vnfds. | ['nsd:nsd-<br>catalog/nsd/constituent-<br>vnfd',<br>'nsd:nsd-<br>catalog/nsd/constituent-<br>vnfd/vnfd-id-ref',<br>'nsd:nsd-<br>catalog/nsd/constituent-<br>vnfd/start-by-default'] |

|       |                                                      |                                                                                                                          |                                                                                                                                             |
|-------|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| osm_7 | invalid-syntax_osm7_nsd_hackfest_simple_k8s_nsd.yaml | Invalid Tag nsd:nsd-catalog/nsd/shortname.                                                                               | ['nsd:nsd-catalog/nsd/short-name', 'nsd:nsd-catalog/nsd/vld/short-name', 'nsd:nsd-catalog/nsd/name']                                        |
| osm_5 | invalid-syntax_osm5_vnfd_vnf_nsh_vnfd.yaml           | Invalid Tag vnfd:vnfd-catalog/vnfd/connection-point/types.                                                               | ['vnfd:vnfd-catalog/vnfd/connection-point/type', 'vnfd:vnfd-catalog/vnfd/connection-point', 'vnfd:vnfd-catalog/vnfd/connection-point/name'] |
| osm_6 | invalid-semantics_osm6_vnfd_clean_vnfd.yaml          | The value 'first' on vnfd:vnfd-catalog/vnfd/vdu/count doesn't match it's datatype (uint64).                              | [1, 2, 4]                                                                                                                                   |
| osm_8 | invalid-semantics_osm8_vnfd_endpoint_vnfd.yaml       | The value '1 gigabyte' on vnfd:vnfd-catalog/vnfd/vdu/vm-flavor/memory-mb doesn't match it's datatype (uint64).           | [1024, 2048, 4096]                                                                                                                          |
| osm_5 | invalid-semantics_osm5_nsd_ubuntu-cloudinit_nsd.yaml | The value 'ELANS' on nsd:nsd-catalog/nsd/vld/type doesn't match one of it's allowed values (['ELAN', 'ELINE', 'ETREE']). | ['ELAN', 'ELINE']                                                                                                                           |



|       |                                                                                    |                                                                                                                                                                                                                              |                                |
|-------|------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|
| osm_7 | invalid-<br>semantics_osm7_vnfd_<br>hack-<br>fest_simple_nativecharm_<br>vnfd.yaml | The value 'EXTERNAL'<br>on vnfd:vnfd-catalog/<br>vnfd/vdu/interface/type<br>doesn't match one of<br>it's allowed values<br>(['INTERNAL',<br>'EXTERNAL']).                                                                    | ['EXTERNAL',<br>'INTERNAL']    |
| osm_5 | invalid-<br>reference_osm5_vnfd_frrvnf_<br>vnfd.yaml                               | The value 'vnf-mgt' on<br>vnfd:vnfd-<br>catalog/vnfd/mgmt-<br>interface/cp doesn't<br>match one of it's<br>reference (vnfd:vnfd-<br>catalog/vnfd/connection-<br>point/name) values of<br>'vnf-mgmt2',<br>'vnf-public'.       | ['vnf-mgmt2',<br>'vnf-public'] |
| osm_7 | invalid-<br>reference_osm7_vnfd_<br>hack-<br>fest_simple_nativecharm_<br>vnfd.yaml | The value<br>'vnf-mgmt-network' on<br>vnfd:vnfd-<br>catalog/vnfd/mgmt-<br>interface/cp doesn't<br>match one of it's<br>reference (vnfd:vnfd-<br>catalog/vnfd/connection-<br>point/name) values. of<br>'vnf-data', 'vnf-mgmt' | ['vnf-data', 'vnf-mgmt']       |
| osm_8 | invalid-<br>reference_osm8_vnfd_vnf_<br>mpls_vnfd.yaml                             | The value 'vnf-cp1' on<br>vnfd:vnfd-<br>catalog/vnfd/vdu/<br>interface/external-<br>connection-point-ref<br>doesn't match it's<br>reference (vnfd:vnfd-<br>catalog/vnfd/connection-<br>point/name) value of<br>vnf-cp0.      | ['vnf-cp0']                    |

|       |                                                                        |                                                                                                                                                                                                                    |                          |
|-------|------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| osm_6 | invalid-<br>reference_osm6_vnfd_<br>hack-<br>fest_proxycharm_vnfd.yaml | The value 'vnf-mgmts'<br>on vnfd:vnfd-<br>catalog/vnfd/mgmt-<br>interface/cp doesn't<br>match one of it's<br>reference (vnfd:vnfd-<br>catalog/vnfd/connection-<br>point/name) values of<br>'vnf-data', 'vnf-mgmt'. | ['vnf-data', 'vnf-mgmt'] |
|-------|------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|

**Table 3:** Correction Suggestions Module Results.