H2020 5GASP Project

Grant No. 101016448

D5.1 Initial Report on Test Plan Creation and Testing Methodologies

Abstract

The development of an automated CI/CD Service in 5GASP is one of the main goals of this project. This service will help SMEs and Telecommunication Operators to significantly speed up their NetApp deployment via continuous gathering and reporting of testing outputs from the involved 5GASP testbeds and the NetApps' own software implementation when they run within the 5GASP ecosystem. This document details the motivation behind the 5GASP's CI/CD Service, the goals it aims to achieve, and presents its initial architecture and components. Furthermore, the document also addresses the tests the CI/CD Service will execute and how the NetApp developers can use it to certify their NetApps.

Document properties

Document number	D5.1
Document title	Initial Report on Test Plan Creation and Testing Methodologies
Document responsible	Rafael Direito (ITAv)
Document editor	Rafael Direito (ITAv), Diogo Gomes (ITAv)
Editorial team	Rafael Direito (ITAv), Diogo Gomes (ITAv), Monika Leung (EANTC)
Target dissemination level	PU
Status of the document	Submission version
Version	1

Document history

Revision	Date	Issued by	Description
0.1	19/09/2021	ITAv	Initial draft
0.2	27/09/2021	EANTC, UNIVBRIS	Internal review
1	29/09/2021	ITAv	Submission version

List of Authors

Company	Contributors	Contribution
ITAv	Rafael Direito Diogo Gomes	Abstract, Objectives and structure of the document, CI/CD Service architecture and components, Service internal communications, and tests overview
VMware	Vesselin Arnaudov	Internal documentation review
UNIVBRIS	Syed Saqlain Ali	Document review
EANTC	Ben Shaw Monika Leung	Approach and methodology, CI/CD Service's ole in the 5GASP Project, Conclusions
UoP	Christos Tranoris Kostis Trantzas	Service interactions/communication, Communication/interactions with other services
OdinS	Jorge Gallego-Madrid Antonio Skarmeta	Description of CI/CD interactions, test descriptor overview and test execution overview
Lamda Networks	Leonidas Lymberopoulos	Section 2.2 - CI/CD Goals and section 4.1 Tests Overview.
ININ	Luka Korsic	Review of Section 4.2 – Testing Descriptors Overview
ORO	Andreea Bonea Marius Iordache	Section 2.1 - Motivation Section 2.3 - Related work/Literature Analysis

Disclaimer

This document has been produced in the context of the 5GASP Project. The research leading to these results has received funding from the European Community's H2020 Programme under grant agreement number 101016448.

Contents

List of Figures

List of Tables

List of Acronyms

3GPP	3rd Generation Partnership Project
5G	5th Generation
5G-PPP	5G Infrastructure Public Private Partnership
AI	Artificial intelligence
API	Application Programming Interface
CD	Continuous Deployment
CI	Continuous Integration
CLI	Command-line Interface
CNF	Cloud-Native Network Function
ETSI	European Telecommunications Standards Institute
EU	European Union
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	HyperText Markup Language
IETF	Internet Engineering Task Force
KPI	Key Performance Indicator
LTR	Local Test Repository
MANO	Management and Orchestration
ML	Machine Learning

NEST	Network Slice Template
NFV	Network Function Virtualization
NFVO	Network Function Virtualization Orchestrator
NODS	NetApp Onboarding and Deployment Services
NS	Network Service
NSD	Network Service Descriptor
NSaaS	Network Slice as a Service
ONAP	Open Networking Automation Platform
OSM	Open Source MANO
PNF	Physical Network Function
RAN	Radio Access Network
REST	Representational State Transfer
SDK	Software Development Kit
SME	Small and Medium-sized Enterprise
SUT	System Under Test
TEE	Test Execute Engine
TMF	Tele Management Forum
UI	User Interface
URL	Uniform Resource Locator
VC	Venture Capitalist
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	Virtual Network Function Descriptor
vOBU	Virtual On-Board Unit
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

Definitions

This document contains specific terms to identify elements and functions that are considered to be mandatory, strongly recommended or optional. These terms have been adopted for similar use to that in Internet Engineering Task Force (IETF) RFC2119 and have the following definitions:

- **MUST** This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
- **MUST NOT** This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
- **SHOULD** This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- **SHOULD NOT** This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- **MAY** This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides).

1 Introduction

1.1 Objectives of this document

The primary goal of this document is to describe all the efforts made to pave the way for achieving a Continuous Integration (CI)/ Continuous Deployment (CD) scenario in 5GASP. Thus, it is an outcome of Task 5.1.

It focuses on the creation of test plans and the associated methodologies that will ultimately compose the testing and validation process. The CI/CD Service's architecture and some aspects of its implementation are presented as well.

The test and validation process starts with the onboarding of a triplet to 5GASP Portal, which bundles the (i) NetApp Artefacts, (ii) the Network Slice Template (NEST) information, and (iii) the Testing Information. The Testing Information is composed of (i) a Tele Management Forum (TMF) ServiceTestSpecification [1] package and (ii) a Testing Descriptor. These are the starting point of the validation process, modeling it and providing the information needed to execute the Platform-specific tests and NetApp-specific tests. Therefore, there is a subsection dedicated to the Testing Descriptors, which also includes information about the TMF ServiceTestSpecification package.

5GASP's CI/CD Service will allow the NetApp developers to validate their NetApps using two types of tests: Platform-specific tests, which will already be onboarded in the CI/CD Service, and NetApp-specific tests, which the developers can submit alongside the Testing Descriptor. The latter are tailor-made tests which validate the functional behavior of a NetApp. Given that the CI/CD Service is not mature enough at the time of the submission of this document, only the Platform-specific tests will be addressed.

The CI/CD Service's main components are: (i) the CI/CD Manager, (ii) the CI/CD Agents, (iii) the Local Test Repositories, and (iv) the Test Visualization Dashboard. Each component and the defined architecture for the CI/CD Service will be presented with a complete description.

Finally, this document also targets the methodology used for executing the tests on the NetApps, and how the outputs of the validation process will be gathered and presented to the developers.

1.2. Approach and methodology

As part of the WP5, this deliverable contains the initial efforts of achieving a 5GASP CI/CD Service that is designed to test and validate the NetApps in the 5GASP test facility sites. The 5GASP NetApp Onboarding and Deployment Services (NODS), which is defined in WP3, allows the NetApps' developers to select one apposite 5GASP test facility site or multiple sites to onboard and deploy their NetApp. The combination of the selected facility and the NetApp form the System Under Test (SUT) in the 5GASP CI/CD Service.

To follow this methodology, the NetApps shall follow the 5GASP architecture requirements defined in WP2, the infrastructure integration requirements defined in WP3, and the design and development requirements defined in WP4. The 5GASP CI/CD Service provides two types of tests, (i)Platform-specific tests and (ii) NetApp-specific tests. The Platform-specific tests which focus on the compliance verification of the standards in D2.1 shall be under the control of 5GASP CI/CD Service; the NetApp-specific tests are for the NetApp dedicated verification (e.g. higher performance and security, private controlling messages, etc). Both types of tests will be triggered by 5GASP NODS through the Testing Descriptor provided by the NetApps' developers. Finally, the NetApps will be certificated after the successful validation of both tests.

The certificated NetApp by the 5GASP Service, the so-called 5GASP NetApp, will be published to the 5GASP Store, which is the portal of the NetApp Marketplace. The details of the 5GASP Store and the certification process will be defined in WP6.

This deliverable provides an initial methodology as an input for T5.2 that the testing tools should follow, and T5.3 that the NetApp's testers should follow.

1.3. Document structure

This document is composed of 6 sections, as follows:

Section 1 presents the main goals, structure and the underlying approach and methodologies for the 5GASP CI/CD Service. Section 2 introduces the motivation behind implementing a CI/CD Service in this project, presenting some related work and literature analysis, which summarizes the state of the art on the implementation of CI/CD tools in Network Function Virtualization (NFV) scenarios.

Section 3 discloses the CI/CD Service's architecture and components, focusing on the communication interfaces used in the CI/CD Service. Section 4 focuses on the tests that will validate the NetApps and presents an overview of the Testing Descriptors and test execution.

Section 5 presents the conclusion, which summarizes the content introduced in this deliverable. Finally, the last section lists the references used in this document.

2 CI/CD Service Overview

2.1 Motivation

The 5GASP proposed platform and DevOps CI/CD process is fully automated and aims to reduce the service deployment at a fraction of the 90 minutes target time proposed by the 5G Infrastructure Public Private Partnership (5G-PPP) Infrastructure target. The service creation time of a NetApp deployed in a targeted facility should only take minutes, which is a very short time, achieved using automation processes and containers. In this process, the continuous integration, delivery, and deployment time is reduced by using improved automated testing and early integration of changes in a shared codebase to reduce conflicts and faster deployment.

Velocity should be doubled by proper synchronization and merging of new code with existing releases. Both continuous integration and continuous delivery split the work in small increments, which should go through automated tests and quality checks before getting prepared for a production release. The focus is on making the process reliable and iterative. 5GASP components that we will constantly optimize are distributed both in the 5GASP experimentation platform (e.g. Service Orchestrator, CI/CD services) as well as in the services at the 5GASP experimentation infrastructure (Network Function Virtualization Orchestrators (NFVOs), 5G-System Virtual Network Functions (VNFs), Radio Access Network (RAN) Physical Network Function (PNFs)/VNFs, Transport network, etc.

To this end, the most consuming process, which is the Virtual Machine (VM) Images/Containers onboarding time, must be addressed. Other challenges may include the automation of the overall process, the detection of faults and errors in future builds and the reduction of build and test time. Increased attention is also focused on scalability and security aspects.

One of the main challenges in the testing process is the creation of testing environments that replicate the production ones. All 5GSASP NetApps will be deployed and tested on 5GASP multi-domain infrastructure, which contains a diverse set of deployed multi-vendor services on testbeds (different Management and Orchestration (MANO), such as Open Source MANO (OSM), Open Networking Automation Platform (ONAP), and different 5th Generation (5G) solutions). The above aspects ensure that 5GASP NetApps will be interoperable across vendors and deployment models and that the applications are automatically and continuously deployed to production or customer environments.

We review the two main stages of the NetApp's onboarding process which refer to the pre-installation and the integration in the facility. For the first stage, the software must be compiled in the facility infrastructure, the licensing needs to be managed and the automated validation needs to be performed systematically. This integration allows the NetApp to interact with the components of the testbed such as the orchestrator, the VNFs, as well as 5G network access, transport, or core sides. If these stages are optimized and automated, the releases could have shorter and more frequent release cycles, as well as increased reliability.

This would help with obtaining the feedback from customers faster to improve the product quality.

To accomplish these objectives, the 5G architectures will develop several functionalities and services, supported by a novel design and implementation. The NetApps will be deployed through the centralized 5GASP NODS which will be useful in the process of triggering and monitoring the experimentation lifecycle. The NODS will also support the community of developers by exposing standardized APIs for programmatic access. The CI/CD pipeline will coordinate the execution of tests by achieving proper integration with different orchestrators. CI/CD tools and services will be deployed and extended to support the VNF/CNF paradigm, by providing staging environments for pre-testing and validation. Automated mechanisms will also be implemented to update the validated NetApps to the open public NetApps marketplace.

2.2 Goals

The overall goal for 5GASP's CI/CD Service is to help Small and Medium-sized Enterprises (SMEs) to significantly speed up their NetApp deployment via continuous gathering and reporting of testing outputs from the involved 5GASP testbeds as well as the NetApps' own software implementation when they run within the 5GASP ecosystem. SMEs pursue to quickly validate their innovation concepts before they can undertake go/no go decisions as well as for deciding to apply for Venture Capitalist (VC) backing or for demonstrating results from their concepts to potential clients. Therefore, a stable and responsive CI/CD Service is an important goal that shall be pursued by the 5GASP consortium in order to provide faster testing times for SMEs. This would improve the development of their NetApp by quickly providing output data from the testing sessions that can be used for analysis and bug fixing.

Given that NetApps shall be tested in an interconnected ecosystem of testbeds, another important goal that should be met by 5GASP's CI/CD Service is that it should (i) detect interconnection and/or network performance problems between the relevant testbeds and (ii) the NetApp developers are automatically informed about such problems. By having this goal met, the CI/CD Services will help SMEs in saving time since this feature of the CI/CD Service can prevent developers from investing own time to debug their own implementation. Specifically, the developer shall be able to promptly be aware if the problem experienced in the communication between software components of the NetApp is due to an 5GASP interconnection testbed issue or a bug in the higher-level functions within their software implementation. Overall, this automated notification feature of the CI/CD Service would all be very convenient and shall aid NetApp developers to quickly locate the root cause of connectivity failures between components instead of waiting for the response of e-mail requests to 5GASP's support team.

Overall, the 5GASP CI/CD Service is an important element within the lifecycle of NetApps in the 5GASP ecosystem. With 5GASP's CI/CD Service, NetApp developers have the advantage to leverage state of the art resources in a cost-effective way and to use the testing data to iteratively refine their implementation. By being able to obtain resources from the real 5G ecosystem of 5GASP, SMEs shall be aided towards reaching their business goals and towards

succeeding in the market via capitalizing their innovations. Last but not least, SMEs might advertise during their pitches to VCs the fact that testing results are not obtained by a lab environment with limited resources but from a large-scale and state of the art and well-recognized European Union (EU) interconnected 5G testbed; this shall provide to SMEs greater credibility and support for realizing their business goals within the competitive 5G market.

2.3 Related work / Literature Analysis

Continuous Integration

CI/CD is a pipeline process that automates the steps involved between compiling a source code and having it deployed in production environment [1]. In terms of integrating CI/CD and fully cloud-native design, there are several challenges that need to be considered such as: lack of standards, technological maturity, and human adaptation resistance [3].

Several verticals would benefit from automated deployment with examples ranging from public safety, media and smart city use cases. In terms of architecture, it is advisable not to use a monolith application but rather an architecture of dynamically assembled and reusable microservices, organized into separate source code repositories as described in [3]. In this paper, the authors have utilized Jenkins, Helm, and Docker, and developed a Command-line Interface (CLI) application in Golang that helps shipping changes to Kubernetes and processing the results. The proposed CLI is responsible for orchestrating deployments and communicating with Kubernetes application API, verifying the existence and creating the namespace, performing additional validations, and ensuring that only images tested in the staging environment can be deployed to production. The aim is to deploy a new version of a microservice to a subset of users and evaluate its correctness and performance before making it available to all users.

Different tools can be employed to achieve an iterative and automated CI/CD. There are multiple CI tools available in the market like Jenkins, Bamboo, Gitlab, TeamCity, Subversion, etc. In [3] a comparison between Jenkins and Gitlab is performed. The review underlines that Jenkins demands more initial configuration time, i.e. installation of git, docker, aws-cli etc., while with Gitlab only a YAML Ain't Markup Language (YAML) file hast to be written. Furthermore, Gitlab gives provision to install runners on Kubernetes and Windows machines, but also on EC2. In terms of duration, Jenkins took 9 seconds for the overall deployment while Gitlab took around 400 seconds with the shared runner and around 11 seconds in case of the specific runner which is almost the same as Jenkins. The comparison highlights that as the number of plugins grows, Gitlab is easier to manage as every detail is in the YAML file and every change in the pipeline is part of CI.

No.	Feature	Jenkins	Gitlab	Drone.IO
1	Performance Monitoring	No	Yes	Yes
2	Server Monitoring	No	Yes	Yes
3	Plugin Support	Yes	No	Yes
4	Pipeline Security	Yes	Yes	Yes
5	JUnit Report Management	Yes	No	No
6	Schedule Pipeline Trigger	Yes	Yes	Yes

Table 1 - CI/CD Tools Comparison

However, in [4] the popularity of Jenkins is noted especially in comparison with Bamboo and TeamCity. The work also describes the integration stages in Jenkins: set version, installation in a specific Jenkins workspace, a zipping stage, and the upload of the zipped file in the JFrog artifactory, and finally calling the CD process. The authors note that after the automation in CI/CD the time for the Ansible deployment of single applications has reduced considerably, which results in improving the maintenance of applications significantly.

An alternative to Jenkins would be Drone [6][7], which comes with built-in Gitops functionality, GitHub and BitBucket integration. Moreover, it has fewer containerized and updated plugins, contains auto-scalers, and is less heavy than Jenkins. Drone is considered as a good alternative to Jenkins, since it is Docker native, which means that it requires less maintenance and script writing and benefits from up-to-date plugins. However, Jenkins is open-source and free, while Drone is open source, but also has enterprise versions.

Continuous Integration related 5GPPP Projects

SONATA [8] was developed based on CI/CD methodologies focusing on improving software quality while decreasing the development time and the gap between development and operation [7]. SONATA focused on a Software Development Kit (SDK), providing tools to package, test, validate, and onboard VNFs. Other focuses were the creation of a validation and verification platform and the development of a service platform where VNFs and Network Services (NSs) are instantiated. SONATA provided a customizable MANO framework, Network Slicing (NS) support and compliance with European Telecommunications Standards Institute's (ETSI) NFV reference architecture.

5G-VINNI [10] accomplished good results regarding NS by developing an internal Network Slice as a Service (NsaaS) mechanisms, advancing the state of the art regarding Key Performance Indicators (KPIs) Validation.

Further, 5G-TANGO [11] aimed at reducing the time-to-market of Network Services, bringing improved customization of network verticals, reducing the developers' effort and accelerating the DevOps model adoption for the validation of Network Services [11]. The offered validation and verification platform exposed an API to interact with the Gateway service, improved the test configuration scheduling, and provided a Test Descriptors repository. Moreover, the packages are validated and the tests are deployed using a testing

engine with all the testing outputs being stored in the test results repository [9]. Another significant outcome of the 5GTANGO project is the development of an SDK that allows the generation and validation of descriptors, emulation of the services and components, as well as benchmark tests and performance analysis of the services [10].

Another important project is the 5G European Validation Platform for Extensive Trials (5G-EVE) [14]. It has the scope of implementing and testing 5G interconnected infrastructures that form a unique 5G end to end facility, which will be made available for other future projects. Its main components are the Portal, the Interworking Layer and the Site-Specific Layer. 5G-EVE focuses on metrics from vertical services and from the underlying infrastructure. It also focuses on continuous testing, integration, and development by making available an Experiment Execution Manager to trigger the execution and validation of test cases through the Results Analysis and Validation component. Also, by using the Performance Diagnosis module, a KPI analysis and anomaly detection can be performed.

Automated validation is also in the scope of the 5GinFIRE [15] Horizon 2020 project which created and operated an open and extensible 5G NFV distributed platform where 5G components and architectures could be deployed and tested before production [14]. The Continuous Integration Pipeline was a contribution that enabled some automation using Jenkins. This Pipeline has several stages for fetching the VNF package, extracting the Virtual Network Function Descriptor (VNFD), performing syntactic, semantic, and reference testing, sending the results to the 5GinFIRE Portal, and cleaning the workspace.

5GASP aims to validate both VNFs and the underlying infrastructure on which they are deployed. To achieve this, 5GASP has the main goal in the creation of an automatic validation pipeline that validates the VNFs against NetApp-specific tests and collects metrics on their performance [15]. 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.

2.4 CI/CD service's role in the 5GASP Project

The CI/CD Service in 5GASP aims to reduce the time needed for service creation and its cost, by providing an open, lightweight, automated, and visualized certification platform. 5GASP designed the CI/CD Service as an open platform to reuse and elevate the achievements from other sources (e.g. open-source projects, other Horizon 2020 projects, and NetApp developers). The lightweight platform is used to locate the Test Execution Engine (TEE) near the NetApps for testing and monitoring. The automated platform has the capability to remotely test the NetApps at any time and the visualized platform enables quick identification and correction of NetApp imperfections.

- 5GASP CI/CD Service as an open platform:
    - Open source server (e.g. Jenkins) for continuous processes like development, integration, and delivery
    - Open-source framework (e.g. Robot Framework) for NetApp-specific tests
    - Open APIs following the industry standards (e.g. TMF, ETSI, 3rd Generation Partnership Project (3GPP)) that will be defined in D5.2 for 3rd party services

- Usage of software tools that will be defined in D5.3 for the OpenStack infrastructure
- Usage of open-source tools that will be defined in D5.3 for optimizable features

- 5GASP CI/CD Service as a lightweight platform:
  - VNF/Cloud-Native Network Function (CNF) based components for fast installation and termination
  - Modular components with a lot of plugins for extended features
  - Distributed components for scalability
  - Layered Descriptor for both NetApp (e.g. Network Service Descriptor (NSD) and VNFD) and Test Execution Engine (Testing Descriptor for Platform-specific and NetApp-specific tests)

- 5GASP CI/CD Service as an automated platform:
  - Automation framework (e.g. Robot Framework) for automation tools without manual testing
  - Automation tools that will be defined in D5.3 for agile testing
  - Artificial Intelligence (AI) / Machine Learning (ML) based tools that will be defined in D5.3 for intelligent testing
  - Remote-control server for offshore testing

- 5GASP CI/CD Service as a visualization platform:
  - Web portal for the processes
  - Human-readable descriptors
  - E-mail notification for defect management
  - Slack channel for test progress
  - CLI/Graphical User Interface (GUI) of the tools and servers
  - Graph panel of the test results
  - Test report as PDF/Microsoft Word
  - Digital certificate of the 5GASP NetApp

3 CI/CD Service Architecture and Components

3.1 Architecture

Given several specifications of each testbed across the 5GASP ecosystem, the CI/CD Service was conceived to operate in a distributed paradigm. The main components are the CI/CD Manager, CI/CD Agents, Local Test Repositories, and the Test Visualization Dashboard.

The central CI/CD Manager orchestrates and coordinates all the CI/CD Agents that are responsible for validating the NetApps. The sequence begins by onboarding a NetApp to 5GASP's Portal, which causes the pre-flight tests to take place. These tests validate only the structure of the onboarded descriptors, although not being a direct component of the CI/CD Service. If the pre-flight tests pass, the NetApps can be deployed on the available testbeds. Regarding the deployment of the NetApps, a multi-domain scenario must be considered, which heavily influences the design and architecture of the CI/CD Service.

After the submission of the NetApps, the NODS will trigger Day 0 configurations on a CI/CD Agent deployed in the Network Slice provided by the testbed where the testing process will take place. This means that the CI/CD Agents are deployed on the same level as the NetApps, which simplifies the testing phase since there is no need to have a central entity that is capable of dealing with different testbed specifications and their underlying infrastructure. The NODS is also responsible to trigger the CI/CD Service after the NetApps are fully deployed.

Besides the global CI/CD Manager and the local CI/CD Agents, each testbed has a Local Test Repository (LTR), that stores Platform-specific tests available for that specific testbed. The LTR is implemented using a File Transfer Protocol (FTP) server, that will make the test available for the CI/CD Agents so that they can gather and perform them on the NetApps. The tests are implemented using the Robot Framework [17] and validate the behavior of the NetApps and their impact on the underlying infrastructure.

The last component of the CI/CD Service is a Visualization Dashboard that enables the NetApp developers to get the outcome and output of the testing and validation phase. After all the tests are performed, the CI/CD Manager will store the results in an FTP server. The results are stored as Extensible Markup Language (XML) and HyperText Markup Language (HTML) files since the Robot Framework produces interactive outputs in the specified file formats. The Visualization Dashboard will then render these files and the developers can check the outcome of the testing phase. It will also provide a Uniform Resource Locator (URL) to access OSM's Grafana where the developers can observe the metrics that are collected during the validation process.

D2.1 already presented all the communication interfaces that are used in 5GASP. The CI/CD Service mainly uses the interfaces E2 - Interface for CI/CD communication, E5 - Interface for facility and testing services management, and E6 - Interface for facility interaction with the CI/CD Service. These communication interfaces are further described in Section 3.3. Besides,

Section 3.2 also provides additional information on the components of the CI/CD Service and their role.

The previously described architecture of the CI/CD Service can be observed in Figure 1, while Table 2 presents a description of each step shown in this diagram.

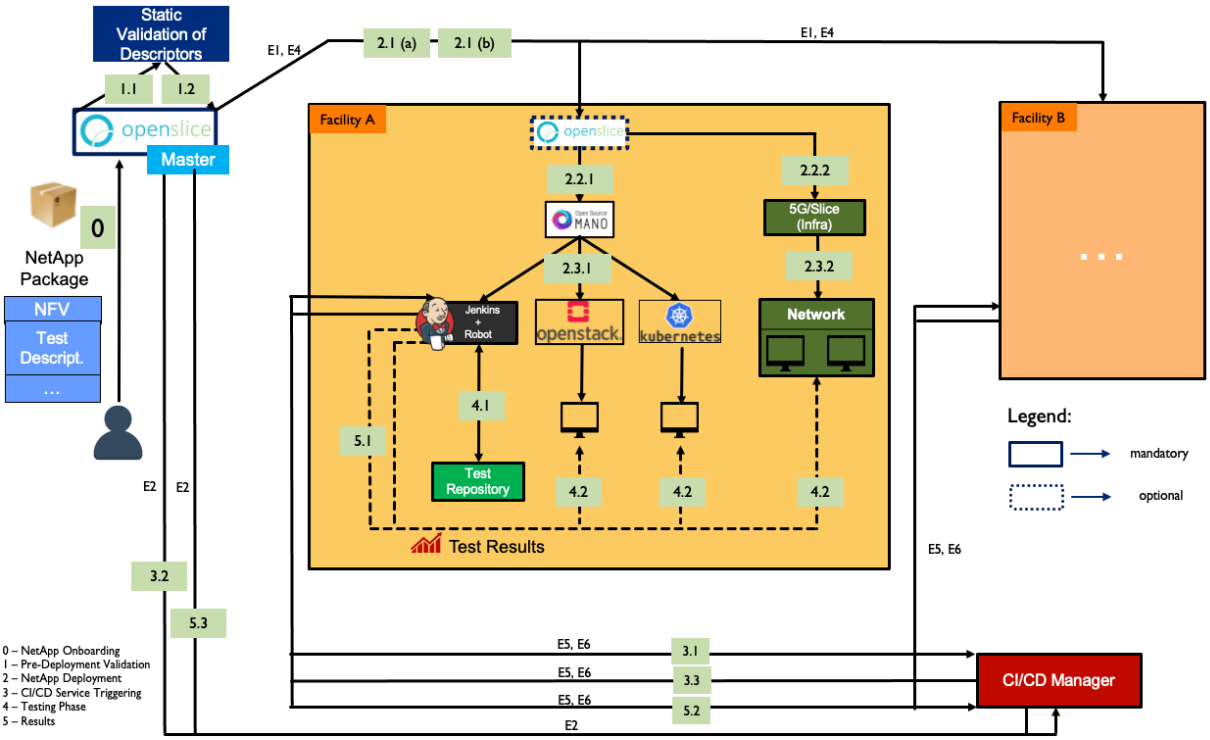


Figure 1 - CI/CD Service Architecture

Stage	Description
0	NetApp and test information submission to the NODS
1.1	Static validation of the VNF descriptors
1.2	Results of the descriptors validation. If all descriptors are valid, we can move to step 2.1
2.1 (a)	The NODS will directly order OSM to orchestrate and deploy the VNFs.
2.1 (b)	The NODS will order the facility Node to orchestrate and deploy the VNFs. This federation scenario will only be addressed once stage 2.1(a) is fully implemented. In this scenario, the facility Openslice node will be responsible for orchestrating the deployment of the VNFs.
2.2.1	Select the correct NSD/VNFD/NEST to be used for the deployment and send the NSD/VNFD/NEST to OSM
2.2.2	Deployment of the VNFs directly in a 5G infrastructure (MEC, RAN, etc.)
2.3.1 + 2.3.2	Deployment of the VNF on the chosen facility and Day 0 configuration of the CI/CD Agent
3.1	The CI/CD Agent will inform the CI/CD that it is online and ready to accept jobs

3.2	The NODS will trigger the CI/CD Service. This can only be done after step 3.1
3.3	The CI/CD Manager will send tasks to the CI/CD Agent
4.1	The CI/CD Agent will obtain Platform-specific tests (Robot Framework)
4.2	The CI/CD Agent will test the VNFs according to the tests that it got from the test repository and the dynamic tests that the NetApp developer uploaded to the portal
5.1 + 5.2 +5.3	The developer gets the results from the testing phase

Table 2 - CI/CD Service's Stages and Interactions

3.2 Components and Roles

The most crucial component of the CI/CD Service is the CI/CD Manager. This entity is accessible via a Representational State Transfer (REST) interface and has the following roles: (i) registering of the CI/CD Agents, (ii) providing information about the tests available in each testbed, (iii) creating the pipeline configuration files that will be submitted to the CI/CD Agents, which will guide the testing and validating phase, and (iv) continuously updating the status of a test and validation process.

Figure 2 shows the documentation of the base functions of the CI/CD Manager's API.

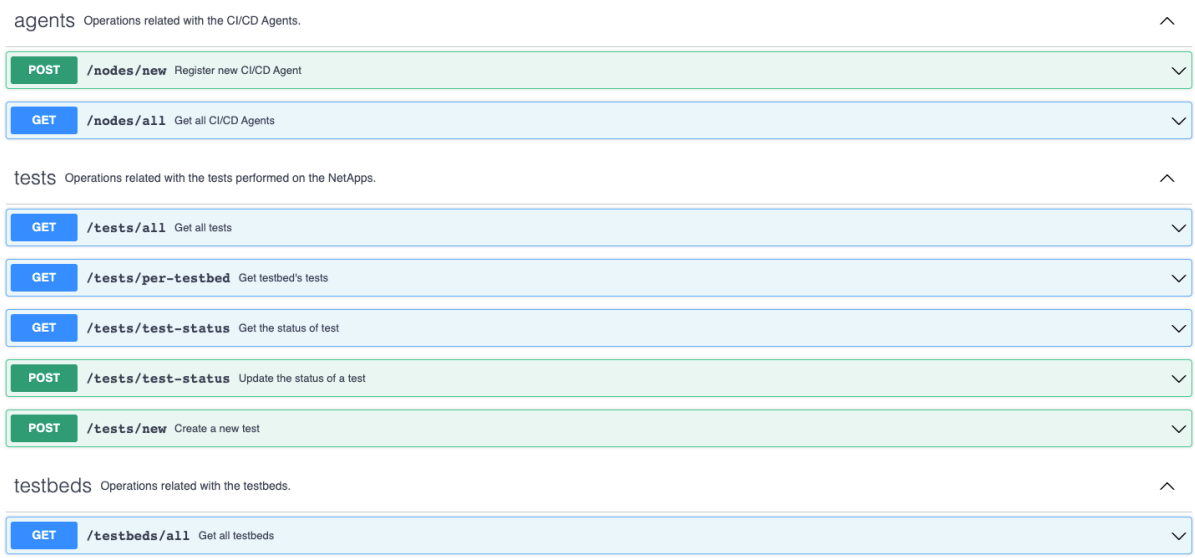


Figure 2 - CI/CD Manager's API Swagger Documentation

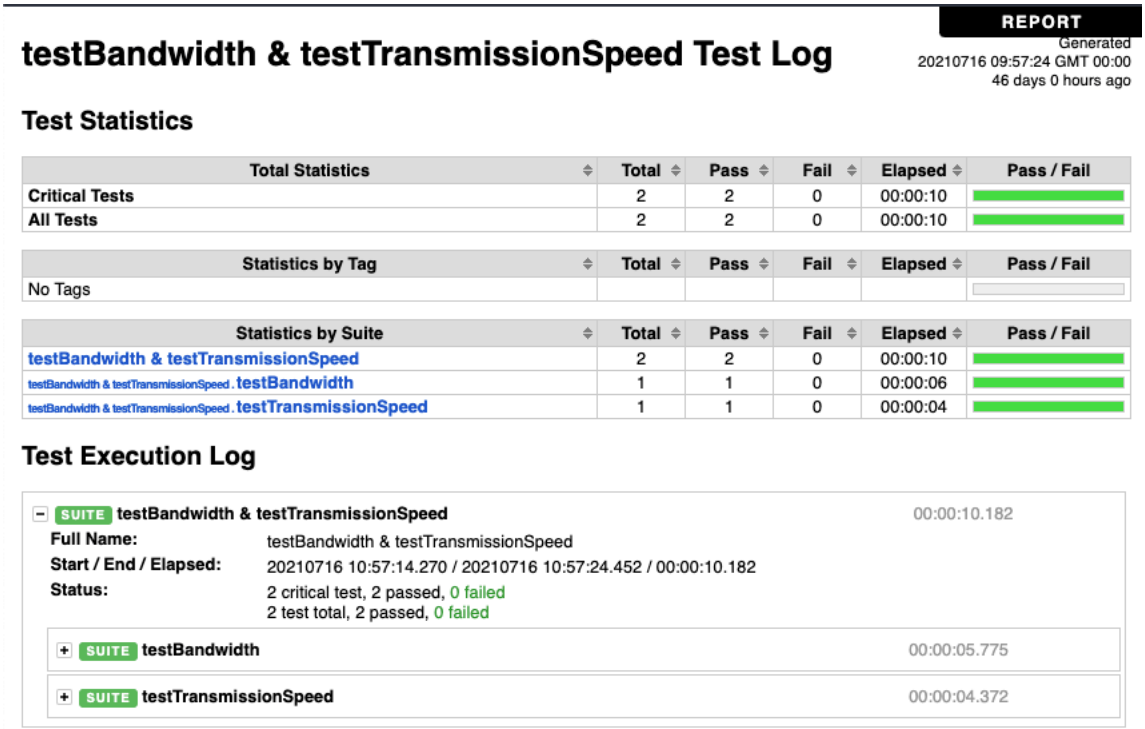
The CI/CD Manager receives the Testing Descriptors from the 5GASP NODS and based on these, it creates a Jenkins pipeline configuration file. Then, it will submit the file to the CI/CD Agent that will perform the validation process. The configuration file contains information about (i) how to set up the testing environment, (ii) how to obtain the tests from the LTR, (iii)

how to perform these tests, (iv) how to update the status of a testing process continuously, and (v) how to submit the test results to the Visualization Dashboard.

Another significant component of the CI/CD Service are the CI/CD Agents. Currently, Jenkins is used to provide the CI/CD Agents, although any other CI/CD tool can be used since the CI/CD Service is highly modular and decoupled. These Agents are deployed in the network slices provided by each testbed and heavily interact with the CI/CD Manager during the validation process. Once a CI/CD Agent is ready to receive work, it will register itself on the CI/CD Manager, submitting information such as its IP address and its login credentials. After this, the CI/CD Agent will be waiting for the CI/CD Manager to create test jobs. A test job is composed of the following steps: (i) setting up the testing environment, (ii) gathering the test from the LTR, (iii) performing the tests, (iv) gathering the testing process output, and (v) submitting the output to the Visualization Dashboard.

The Local Testing Repository is an entity which is present in every testbed. It is implemented using an FTP server that stores all the tests that can be performed within the facility. These tests are created using the Robot Framework and will be performed by the CI/CD Agents using Python. The CI/CD Manager has information about the tests stored in each LTR and provides this information to the NetApp developers, so they can choose the testbed where they want to deploy and test their NetApp.

The last component of the CI/CD Service is the Results Visualization Dashboard, composed of a WebUI and an FTP server. Once the testing phase is completed, the CI/CD Agents will submit the results to this FTP server. Robot Framework presents the results in two different ways: (i) an XML file and (ii) an HTML file that provides the developers a GUI to visualize the testing results. Figure 3 shows a rendered HTML results file from the Robot Framework.



testBandwidth & testTransmissionSpeed Test Log

REPORT
Generated
20210716 09:57:24 GMT 00:00
46 days 0 hours ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	2	2	0	00:00:10	
All Tests	2	2	0	00:00:10	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
testBandwidth & testTransmissionSpeed	2	2	0	00:00:10	
testBandwidth & testTransmissionSpeed . testBandwidth	1	1	0	00:00:06	
testBandwidth & testTransmissionSpeed . testTransmissionSpeed	1	1	0	00:00:04	

Test Execution Log

SUITE testBandwidth & testTransmissionSpeed 00:00:10.182
Full Name: testBandwidth & testTransmissionSpeed
Start / End / Elapsed: 20210716 10:57:14.270 / 20210716 10:57:24.452 / 00:00:10.182
Status: 2 critical test, 2 passed, 0 failed
2 test total, 2 passed, 0 failed

SUITE testBandwidth 00:00:05.775

SUITE testTransmissionSpeed 00:00:04.372

Figure 3 - Robot Framework HTML Output File

The WebUI of the Visualization Dashboard will use Robot's HTML files and render them. Besides, it also provides an URL to access OSM's Grafana. This way, the developers can observe the metrics that were collected during the validation phase. The Visualization Dashboard can be observed in Figure 4.

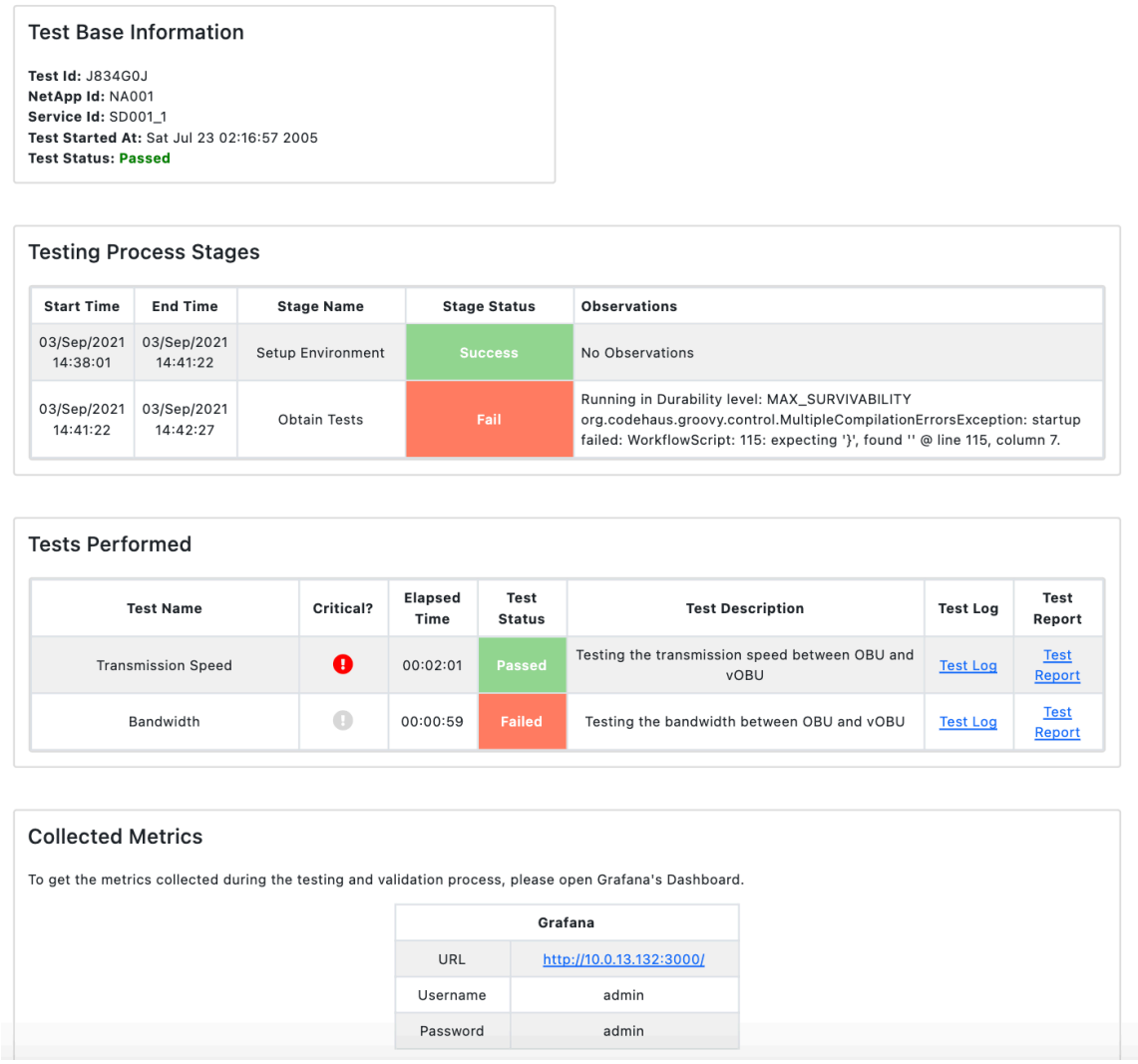


Figure 4 - CI/CD Service - Visualization Dashboard

Figure 6 presents the interaction diagram for the CI/CD Service. By viewing this figure, it is possible to understand the base operation of this service.

3.3 Service interactions/communication

This section focuses on the interactions and communication with the CI/CD Service, both internal and external. It addresses the communication interfaces that were described in D2.1 and can be observed in Figure 5.

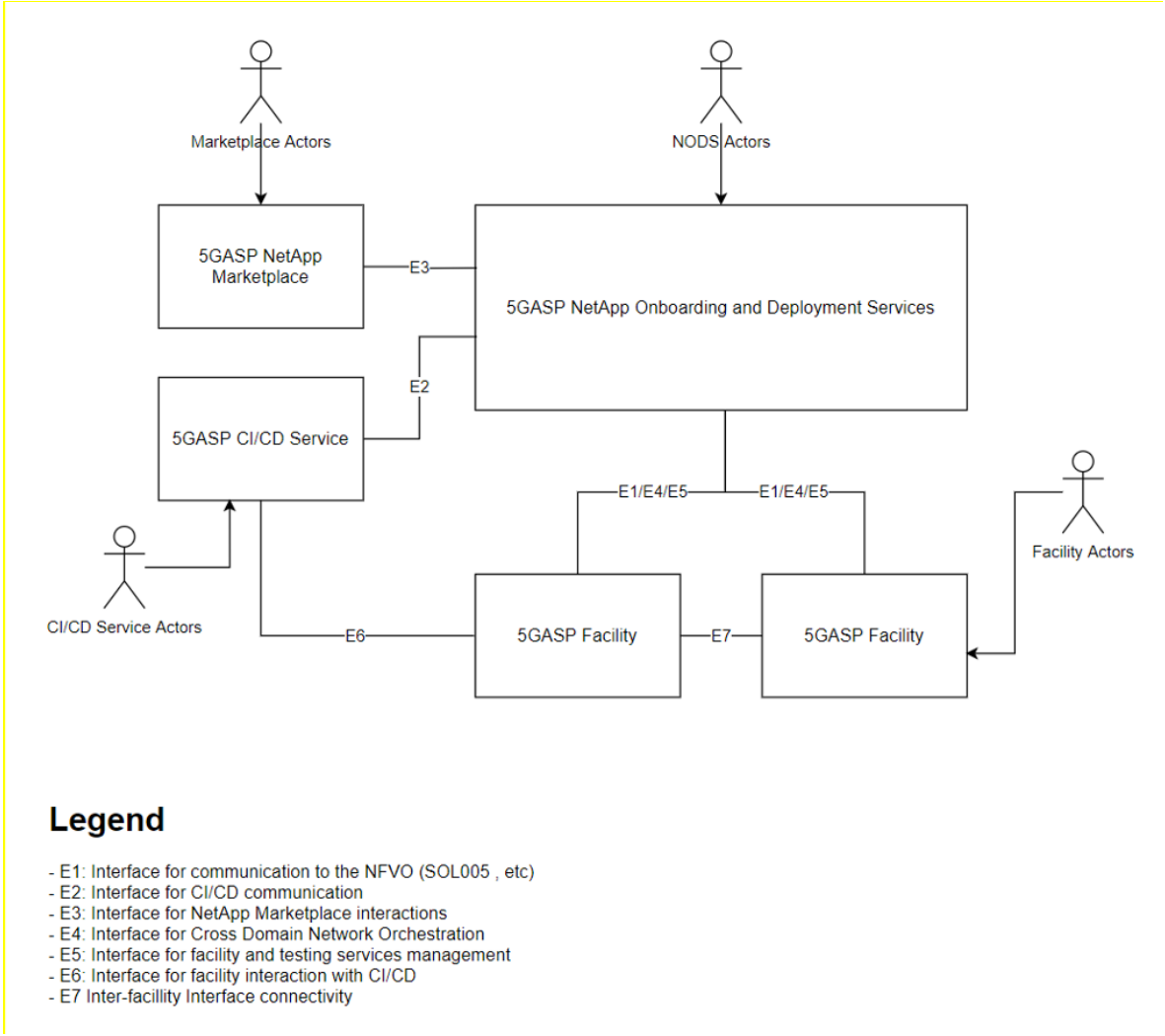


Figure 5 - 5GASP's Communication Interfaces

Following, the aspects of how the CI/CD pipeline is perceived and executed internally from the 5GASP CI/CD Service are presented in Section 3.3.1, along with the corresponding external interfaces that enable the CI/CD Service to communicate with its adjacent components (see section 3.3.2) to complete the automation circle. A more comprehensive view of the aforementioned processes is provided to the reader of this document through interaction diagrams for each case in the next subsections.

3.3.1 Service internal communication/interactions

5GASP's service internal communication will mainly occur via the interface E6. After 5GASP NODS triggers a validation process on the CI/CD Manager, via interface E2, this entity will be responsible for all testing processes and the communication with the other CI/CD Service's components.

The CI/CD Manager starts by creating a Jenkins pipeline configuration file, based on the Testing Descriptor onboarded on NODS, and submits it to the CI/CD Agent responsible for testing the NetApp, via the communication interface E6. Before this, the CI/CD Agent had already registered itself on the CI/CD Manager.

In this next step, the CI/CD Agent will have to communicate with the LTR present on the testbed to gather all the test files needed to start the testing process. The LTR is password-protected, so the CI/CD Manager will have to send the Agent the required credentials. This step occurs when a new test job is assigned to an Agent.

While validating the NetApp, the CI/CD Agent will use interface E6 to constantly update the test status to the CI/CD Manager, which provides an endpoint to do so.

After the testing process is finalized, the CI/CD Agent will submit the testing outputs to the Visualization Dashboard File Server, which will allow the NetApp developers to consult these files and receive feedback on the validation process.

Lastly, the CI/CD Manager will inform the NODS that the testing and validation process has ended, and it will submit information on how the developers will be able to check its outputs.

Figure 6 shows an interaction diagram with all the internal communication of the CI/CD Service, via interface E6, and the interactions between the NODS and the CI/CD Manager, via interface E2.

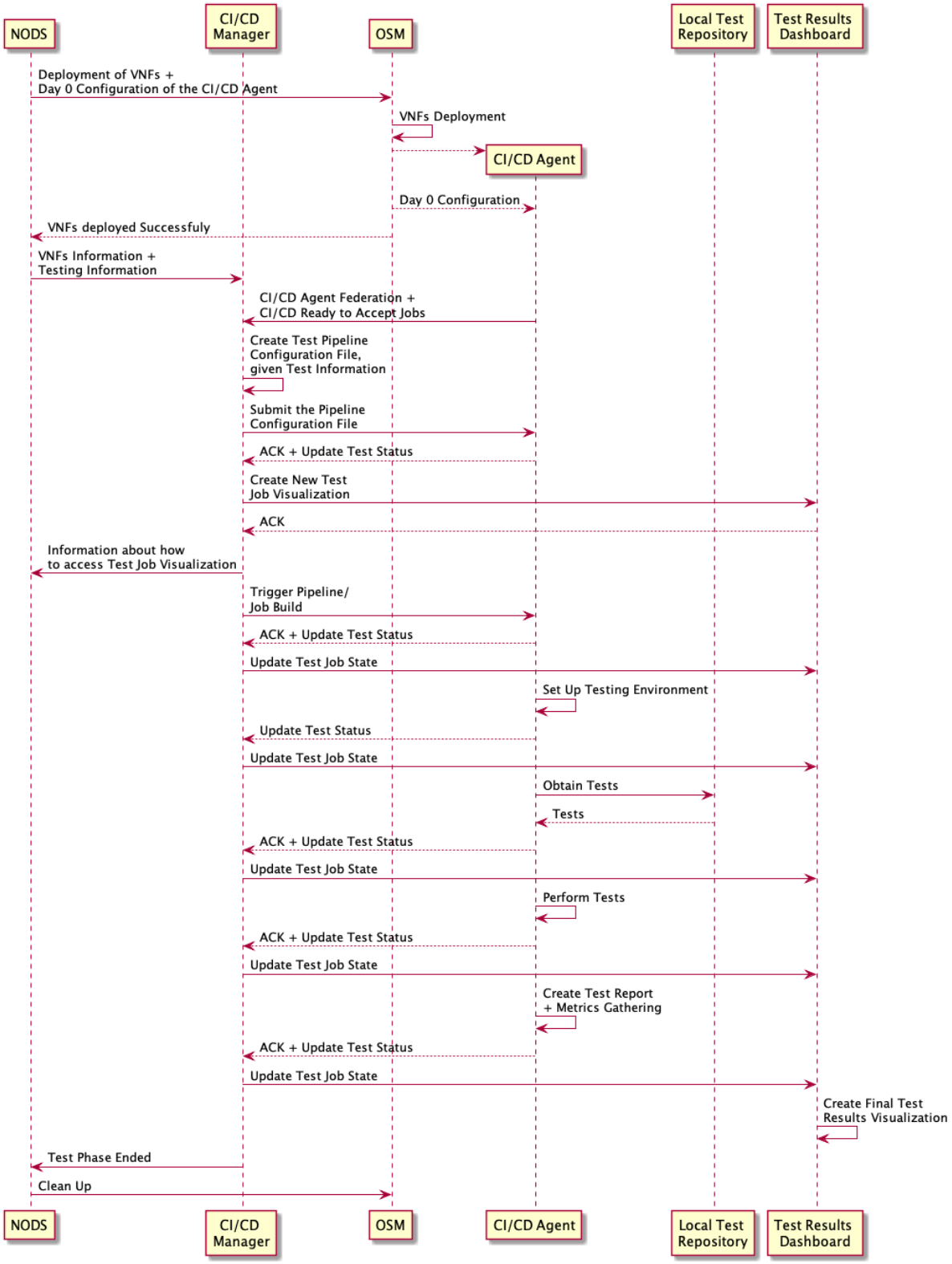


Figure 6 - CI/CD Service's Interaction Diagram

3.3.2 Communication/interactions with other services

As the previous section outlines the CI/CD pipeline from the scope of internal interactions and communication between its comprised services, this section focuses on the interaction of the CI/CD Manager with its external adjacent components through well-defined interfaces, as depicted in Figure 7.

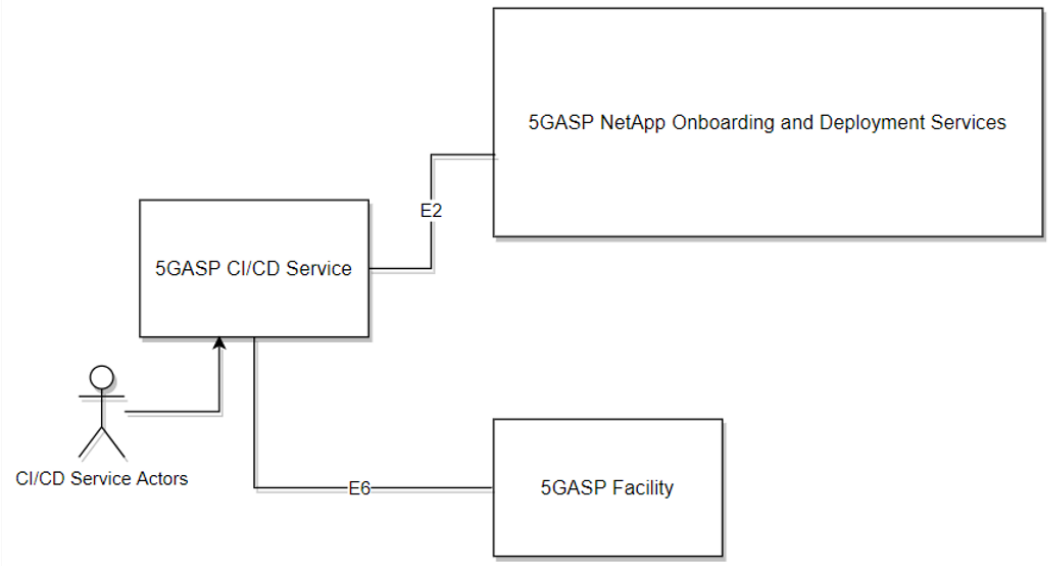


Figure 7 - CI/CD Manager Adjacent Components and Respective Interfaces

The CI/CD Manager establishes the interconnection with 5GASP NODS through interface E2. This interface is charged with the following tasks:

- Provision of VNFs and testing information to CI/CD Manager, after hosting slice and NetApp are deployed
- Exposure of test job visualization information to NODS, thus rendering it aware of the state of the CI/CD pipeline
- Notification to NODS about the completion of the test phase

The above procedure is represented more comprehensively through an interaction diagram in Figure 8.

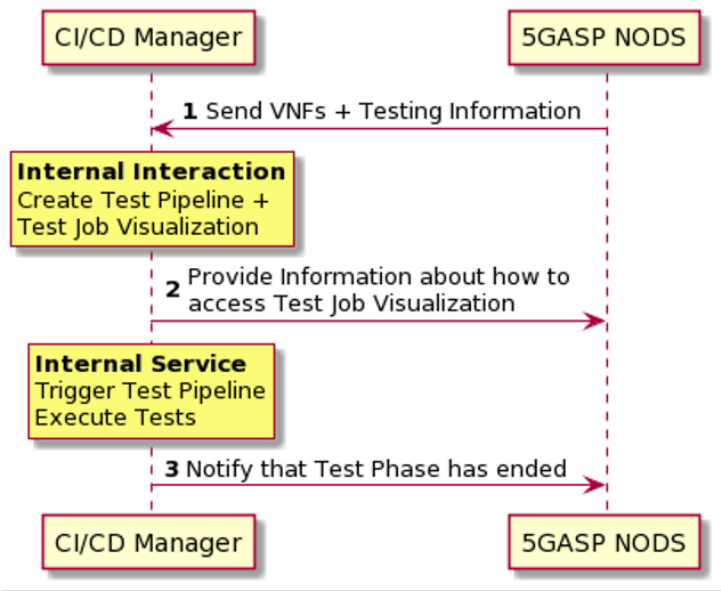


Figure 8 - E2 Interface Utilization

Specifically, after NODS's internal orchestrator completes the deployment process of the hosting network slice and the NetApp, the flow is transferred to CI/CD Service through interface E2. Consequently, NODS employs the CI/CD Service through a REST interface (already introduced in Section 3.2) forwarding the pipeline configuration files to guide the testing phase. At this stage, the provided test configurations, apart from the Test Descriptors initially selected by the developer, are augmented with details of the deployed NetApp, e.g. IP addresses.

Following the internal CI/CD procedure of creating the test pipeline and visualization, interface E2 is employed once again to expose this information to NODS. This enables a constant feedback chain to the developer about the on-going testing phase. A notable mention here is that this process coincides with the actual triggering of the test pipeline.

Eventually, as the testing phase is completed and final results are submitted to the Visualization Dashboard, the flow is transferred back to NODS, notifying the developer about these results and further enabling the completion of the automation circle.

4 Test execution

4.1 Tests overview

As discussed in Section 2.2, that one of the goals of the CI/CD Service is to be able to detect testbed-specific problems. The latter can be connectivity or network performance issues within the testbed targeted by the NetApp under testing by 5GASP's CI/CD Service. Furthermore, the NetApp developer must be aware if the problem experienced in the communication between software components of the NetApp is due to an 5GASP interconnection testbed issue or a bug in the higher-level functions within the NetApp's software implementation. At Last, each NetApp has typically its own specific tests that must pass before the NetApp can be moved from the staging to the production environment. Therefore, two main types of tests are involved in the CI/CD Service of 5GASP: (i) Platform-specific tests and (ii) NetApp-specific tests.

Platform-specific tests might access the connectivity and the performance of the communication between VNFs or CNFs of a NetApp. An example test to assess the bandwidth between the OBU and vOBU VNFs is provided in Section 4.2.

NetApp-specific tests shall be uploaded to the test repository by the developers of each NetApp. These tests intend to examine the behavior and/or performance of specific functionalities of the NetApp as well as to evaluate NetApp-specific KPIs. The aim of this testing process caters to the developer's assurance that their NetApp can be transferred to the production environment in the case that tests are successful. In contrast, if the tests fail, the developers must revise their implementation and re-run the tests.

To illustrate a paradigm of NetApp-specific tests, we shall use as an example of PrivacyAnalyzer NetApp. The goal of this NetApp is to analyze privacy issues within messages stemming from UEs (e.g. devices carried by First Responders in the PPDR use case). To this end, the developer of the PrivacyAnalyzer NetApp requires the following example tests to be able to be undertaken by the 5GASP CI/CD Service: (a) tests that measure the loss percentage of messages between the sender (e.g. a UE) and the receiver (i.e. Lamda Network's Qiqbus streaming platform that analyzes all incoming messages for privacy issues), (b) tests that measure the maximum volume of messages supported by the slice assigned to PrivacyAnalyzer (e.g. to indicate that the offered slice supports for example 100K messages/min), (c) min/max/average message processing time (in ms) within PrivacyAnalyzer, and (d) accuracy of the privacy analysis itself, expressed with the precision and recall metrics as described in the PrivacyAnalyzer section in D2.1. From the above, data for the tests {a, b, c} can be obtained from the testbed and thePrivacy Analyzer's monitoring container, whilst for test {d} we have discussed that synthetic data sets shall be stored and used for the testing process. Given that PrivacyAnalyzer's synthetic data is around 1 GB in size, the 5GASP CI/CD Service shall provide the necessary storage for hosting NetApp-specific synthetic data.

In conclusion, testing in 5GASP encompasses Platform-specific and NetApp-specific tests and possibly relevant datasets for testing a NetApp. Such information shall be described in the

Test Descriptors in Section 4.2 so that it is leveraged in the test execution discussed in Section 4.3.

4.2 Testing Descriptors overview

Due to the lack of existing standards related to the definition of Testing Descriptors to specify the test procedures, there is the need to validate NetApps that requires a design of our own model. We have based our design on ideas from other projects that also worked on testing methodologies, such as 5G-EVE [14] and 5GTANGO [11]. The designed and developed test descriptor is in its inception stage, as we are in the first year of the project. In its current state, it is a proof of concept that will be evaluated during the next months. Besides, the feedback received from the different partners of the project, once they begin to use it, will help to improve it through several iterations. These enhancements will be aligned with 5GASP objectives and focused on simplifying the design and development of the descriptor.

The designed test descriptor will be used in conjunction with TMF's ServiceTestSpecification and ServiceTest models [1].These models provide a standardized mechanism to place a test with all the necessary parameters to check the quality, performance, or reliability of a service. The onboarding and initial processing details of the ServiceTestSpecification is out of the scope of WP5, as it belongs to WP3, and it is detailed in D3.1. In summary, the Test Descriptor is uploaded as an attachment of a TMF ServiceTestSpecification, which describes the main aspects of the testing process, such as information about the NetApp under study, the parameters to be configured, and the measures to be taken. Then, 5GASP NODS will process the specification and it will generate a TMF ServiceTest that will be forwarded to the CI/CD Manager. The NODS will be agnostic from the implementation details of the 5GASP Test Descriptor.

The WP5 scope in this whole procedure is highlighted in Figure 9. The CI/CD Manager will receive the TMF ServiceTest. It will contain the information about the invocation of the test on a NetApp and the deployment data already filled by 5GASP NODS including all the information that has been gathered after the instantiation of the VNFs, NSs and the 5GASP Test Descriptor. The CI/CD Manager will have to implement the TMF model to process the ServiceTest and translate the necessary information to complete the 5GASP Test Descriptor, which can be now stored and processed. To make this possible, the Test Descriptor needs to contain empty fields directly related to the deployment information filled by the NODS. In a similar way, once the test results are available, they can be feedback to the NODS using TMF ServiceTest to be stored in the result repository.

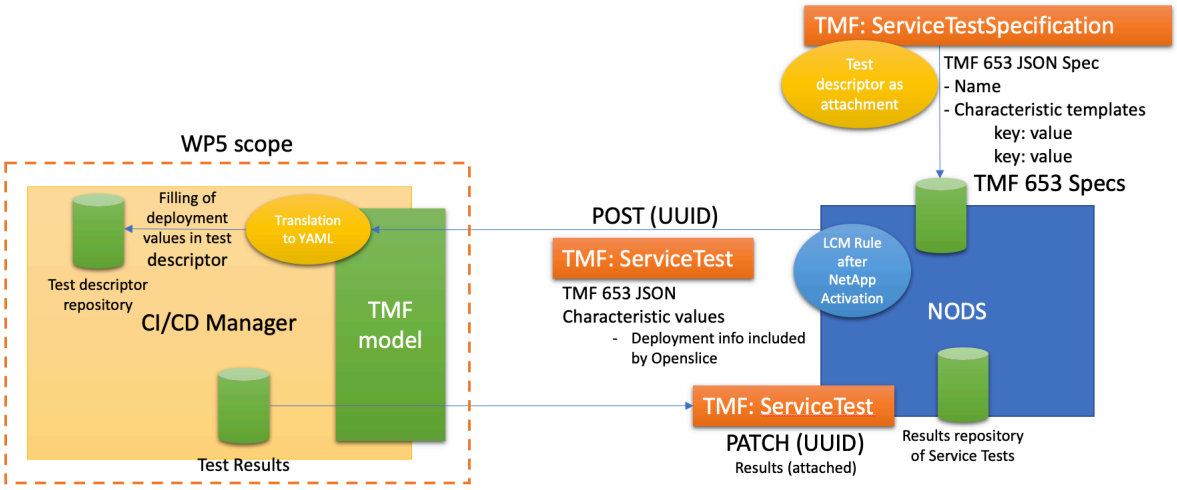


Figure 9 - Onboarding and Handling of the Test Descriptor

The Test Descriptor has been designed to be easily understandable and simple to develop for newcomers to 5G and virtualization technologies. In this way, it is designed to sequentially describe the testing process. By doing so, we ease both the readability for developers or experimenters and the processing from the NODS and the CI/CD Service. Moreover, this structure of Test Descriptor fits into the workflow adopted by Jenkins, where the pipelines are also established in a sequential way. Thus, new 5G experimenters will be able to relate the test description to the traditional test methodologies.

The descriptor's model is divided into three parts that represent the three phases of the testing process: (i) setup, (ii) execution, and (iii) validation. In the setup phase, all the preparatory configurations are performed, as well as the deployment of the custom test VNFs. All the test cases that will be used to validate a NetApp are described here and information related to the type and scope of the test cases are provided as well. Finally, every test case includes parameters in the form of key-value pairs that are later directly passed as arguments to the test files themselves. In the execution phase, the test cases are grouped in batches which are classified by scope. Each test in a batch will be executed together. In this part, the descriptor includes data about the number of instances of each test case or the start delay. The results are gathered and validated in the validation section.

In the following, an example of the first version of the Test Descriptor is presented. Note the fields that will be filled with the instantiation and deployment information gathered from 5GASP NODS. This Test Descriptor defines the tests to validate the first NetApp of the project: the Virtual On-Board Unit (vOBU). As mentioned before, it is the first version and implementation of the Test Descriptor model designed for the 5GASP project. Thus, it is constantly being evaluated and subject to change thanks to the feedback received from the project partners.

```
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:
      # Pltform-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
        file: "testBandwidth" # CI/CD Service will retrieve the test with that name for the repository
        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
      file: "plr"
      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]
  # 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
          start_delay: 0
        - execution_id: 2
          name: bandwidth_test
          testcase_ids: [2]
          instances: 1
          start_delay: 0
    - batch_id: 2
      scope: operational
      executions:
      - execution_id: 1
        name: packet_loss_ratio_test
        testcase_ids: [3,4]
        instances: 1
        start_delay: 0
      - execution_id: 2
        name: custom_vnfs
        testcase_ids: [5]
        instances: 1
        start_delay: 60
# Validate outputs of the test cases
  validation:
    - validation_id: 1
      execution_id: 5
      file: "vnf_log.txt"
```

Code Block 1 - Testing Descriptor

4.3 Tests execution overview

The test execution stage is triggered by Jenkins after it gathers the Robot Framework test files from the LTR. Jenkins starts by setting some environment variables which will be used in the Robot Framework tests. These variables are defined according to the information present on the Testing Descriptors. Figure 10 presents a portion of a Testing Descriptor, where several parameters are defined, and its mapping in the Jenkins pipeline configuration.

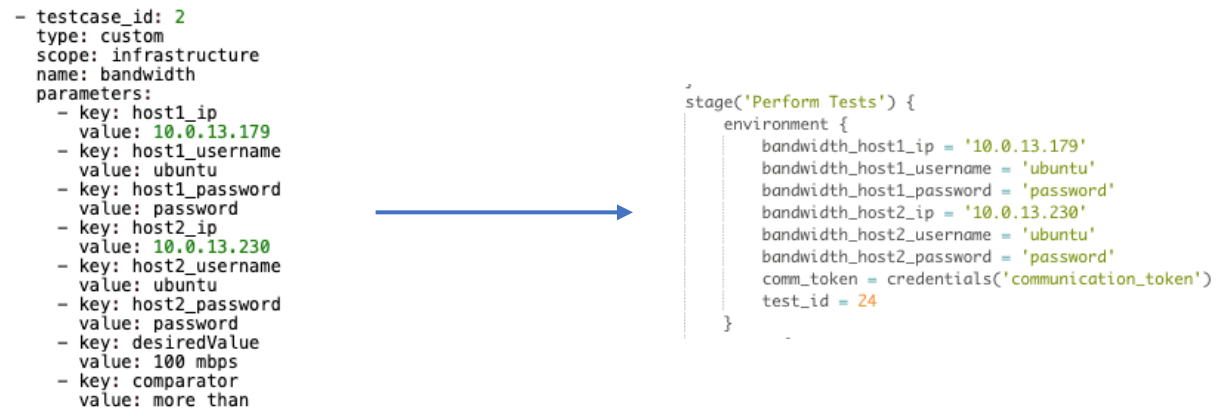


Figure 10 - Mapping of testing variables from the Testing Descriptor to the Jenkins pipeline configuration

After the establishment of these environment variables, the CI/CD Agent is ready to execute the Robot Framework test files. Robot Framework uses Python to perform the business logic behind each test, where Jenkins only needs to use Python to execute the test files.

The Robot Framework language allows to create very flexible tests, since it is possible to define keywords adapted to our requirements and to import our own Python code as libraries to provide further functions. In our case, the logic of the test will be defined in Python code. Whereas the Robot Framework test that uses the imported code as library will be in charge of validating the final result.

Since the arguments for the tests will be specified in the Test Descriptor, it will be necessary to use the descriptor as a variable file. Robot Framework supports the usage of variable files, these arguments can be passed to the methods of the imported Python libraries.

When Jenkins executes the Robot Framework tests, it also defines which logs should be stored and where. By default, Robot Framework creates three different files with the outputs of the testing process: log.html, report.html, and output.xml.

Firstly, the log.html file contains details about the executed test cases in HTML format. It has a hierarchical structure showing the test suite, test case and keyword details. The file contains

timestamps as well and is useful for checking the results step by step in detail. Figure 11 shows an example of this type of file.

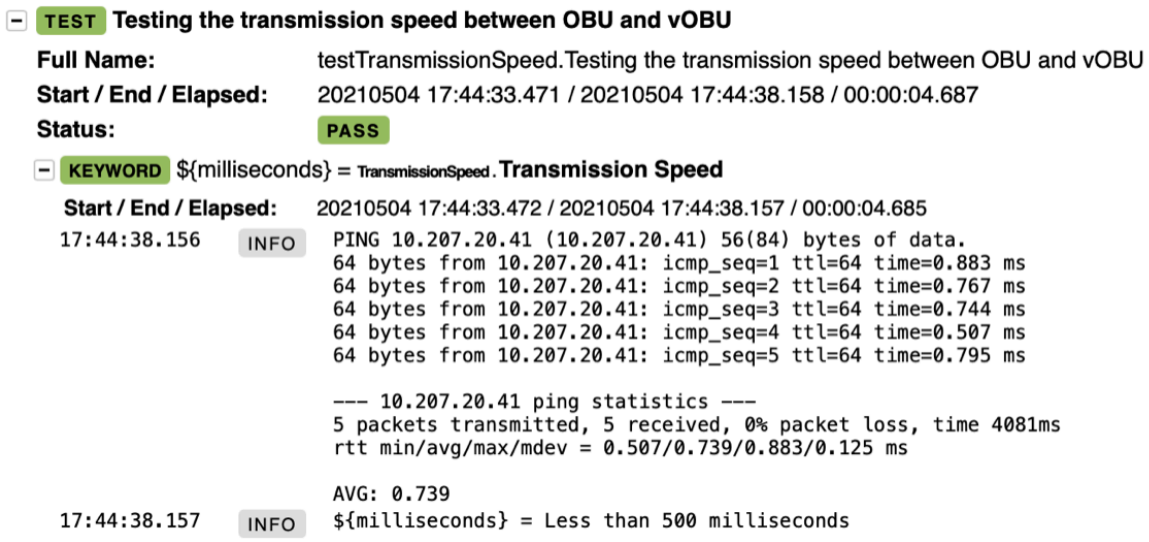


Figure 11 - Content of the log.html file

Secondly, the report.html file contains an overview of the test execution results in HTML format. It acts as a general summary of the test results. If both log and report files are generated, the report has links to the log file for easy navigation.

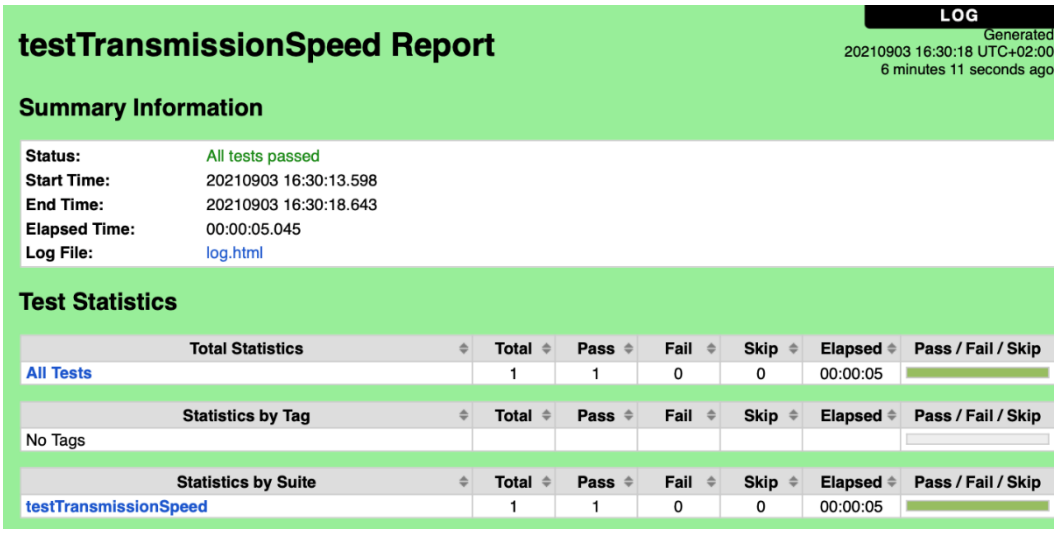


Figure 12 - Content of the report.html file

Finally, the output.xml file contains all information about the execution in XML format. Robot Framework uses it internally to create the log and report files.

```
<?xml version="1.0" encoding="UTF-8"?>
<robot generator="Robot 4.0.1 (Python 3.8.10 on linux)" generated="20210903 16:30:13.597" rpa="false" schemaversion="2">
<suite id="s1" name="testTransmissionSpeed" source="/home/daniel/Escritorio/testTransmissionSpeed.robot">
<test id="s1-t1" name="Testing the transmission speed between OBU and vOBU">
<kw name="Transmission Speed" library="TransmissionSpeed">
<var>${milliseconds}</var>
<msg timestamp="20210903 16:30:18.641" level="INFO">PING 10.207.20.41 (10.207.20.41) 56(84) bytes of data.
64 bytes from 10.207.20.41: icmp_seq=1 ttl=64 time=0.991 ms
64 bytes from 10.207.20.41: icmp_seq=2 ttl=64 time=0.418 ms
64 bytes from 10.207.20.41: icmp_seq=3 ttl=64 time=0.481 ms
64 bytes from 10.207.20.41: icmp_seq=4 ttl=64 time=0.374 ms
64 bytes from 10.207.20.41: icmp_seq=5 ttl=64 time=0.428 ms

--- 10.207.20.41 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4054ms
rtt min/avg/max/mdev = 0.374/0.538/0.991/0.228 ms

AVG: 0.538</msg>
<msg timestamp="20210903 16:30:18.641" level="INFO">${milliseconds} = Less than 500 milliseconds</msg>
<status status="PASS" starttime="20210903 16:30:13.735" endtime="20210903 16:30:18.641"/>
</kw>
<kw name="Should Be Equal" library="BuiltIn">
<arg>${milliseconds}</arg>
<arg>Less than 500 milliseconds</arg>
<doc>Fails if the given objects are unequal.</doc>
<status status="PASS" starttime="20210903 16:30:18.641" endtime="20210903 16:30:18.642"/>
</kw>
<status status="PASS" starttime="20210903 16:30:13.735" endtime="20210903 16:30:18.642"/>
</test>
<status status="PASS" starttime="20210903 16:30:13.598" endtime="20210903 16:30:18.643"/>
</suite>
<statistics>
<total>
<stat pass="1" fail="0" skip="0">All Tests</stat>
</total>
</robot>
```

Figure 13 - Content of the output.xml file

As shown above, Robot Framework generates useful and meaningful reports about the test execution and its results. The logs are formatted in standardized markup languages, and they can be easily integrated in any platform. In this way, this solution can be easily incorporated with the Results Visualization Dashboard to visualize the test results. Although solutions more suited to the project vision can be implemented, these reports are very useful as a first approach and can be later enhanced or used as a source for future developments.

5 Conclusions

One of 5GASP's objectives is to provide the automation of testing and validation, and lowering the cost associated with testing and certification of a NetApp. In this deliverable, the initial methodology has been proposed for the architecture and aspects of 5GASP CI/CD implementation. This methodology has been developed by using past, test-focused projects, such as 5GTANGO and 5G-VINNI as foundation pillars. Thus, the ideas followed by the proposed methodology are evolved from the different approaches that have been made in the past.

The initial methodology discussed in the document follows the requirements of the architecture defined in WP2. At the same time, it benefits from the enhancements being proposed and developed in the context of WP3 and WP4. Moreover, the presented testing and validation processes conform to the preparatory steps that connect with the NetApp onboarding and integration preliminary phases proposed in WP3 and the NetApp design and development initial workflows proposed in WP4.

For that purpose, the 5GASP CI/CD Service is described and defined on higher-level. An initial design and dissection of the goals and features of the CI/CD Service are also presented.

In addition, an initial description of the CI/CD Service architecture is provided with a detailed definition of the steps, components, and roles. The interactions and communication are addressed on the internal CI/CD Service interface level and on the external level to 5GASP NODS.

Finally, the discussion is oriented towards the initial testing and validation approach that SMEs will follow. It describes what type of tests will be automated for NetApps and how NetApps will be validated via 5GASP CI/CD Service, using a Testing Descriptor. Also, an example of the application of the methodology is showcased using the vOBU NetApp, in section 4.2.

The next steps for WP5 will focus on the enrichment of this defined methodology. D5.2 will define the open APIs according to this methodology and D5.3 will develop the open-source automation tools for the automotive and PPDR verticals. At the same time, the iteration over this methodology will be based on the feedback from the multiple SMEs participating in the project.

In conclusion, the main focus of these iterations and enhancements will be automating the 5GASP CI/CD Service to certificate both NetApps and the facilities in the 5G ecosystem market.

6 References

[1] TMF Forum, "TMF653 Service Test Management API User Guide v4.1.0," TMF Forum.

[2] D. Rangel, DevOps: Learn One of the Most Powerful Software Development Methodologies FAST AND EASY!.

[3] B. Sayadi et al, "Cloud-Native and Verticals' services".*5G-PPP Whitepaper.*

[4] N. Railić and M. Savić, "Architecting Continuous Integration and Continuous Deployment for Microservice Architecture," in *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*.

[5] S. Mysari and V. Bejgam, "Continuous Integration And Continuous Deployment PipelineAutomationUsing Jenkins Ansible," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*.

[6] Harness, "Drone vs Jenkins," [Online]. Available: https://harness.io/jenkins-vs-drone/. [Accessed 01 September 2021].

[7] Knapsack Pro, "Jenkins vs Drone," [Online]. Available: https://knapsackpro.com/ci_comparisons/jenkins/vs/drone. [Accessed 091 09 2021].

[8] SONATA, "SONATA," [Online]. Available: https://www.sonata-nfv.eu/. [Accessed 17 September 2021].

[9] T. Soenen, S. V. 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.

[10] 5G-VINNI, "5G-VINNI," [Online]. Available: https://www.5g-vinni.eu/. [Accessed 17 September 2021].

[11] 5GTANGO, "5GTANGO," [Online]. Available: https://www.5gtango.eu/. [Accessed 31 August 2021].

[12] 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 *European Conference on Networks and Communications (EuCNC)*, 2018.

[13] 5GTANGO, "D4.2 final release of the service validation sdk toolset," 2019.

[14] 5G-EVE, "5G-EVE," [Online]. Available: https://www.5g-eve.eu/. [Accessed 17 September 2021].

[15] 5GinFIRE, "5GinFIRE," [Online]. Available: https://5ginfire.eu/. [Accessed 31 August 2021].

[16] 5GPPP, "5GASP," [Online]. Available: https://5g-ppp.eu/5gasp/. [Accessed 31 August 2021].

[17] Robot Framework, "Robot Framework," [Online]. Available: https://robotframework.org/. [Accessed 31 August 2021].

[18] C. Singh, N. S. Gaba, M. Kaur and B. Kaur, "Comparison of Different CI/CD Tools Integrated with Cloud Platform," in *9th International Conference on Cloud Computing, Data Science & Engineering*.

[19] Digital Ocean, "CI/CD Tools Comparison: Jenkins, GitLab CI, Buildbot, Drone, and Concourse," [Online]. Available: https://www.digitalocean.com/community/tutorials/ci-cd-tools-comparison-jenkins-gitlab-ci-buildbot-drone-and-concourse. [Accessed 01 09 2021].