Testbed installation and configuration

MORPHEMIC

Deploying and configuring a test environment to validate MORPHEMIC platform developments and use-cases

H2020-ICT-2018-2020
Leadership in Enabling and Industrial Technologies: Information and Communication Technologies

Grant Agreement Number
871643

Duration
1 January 2020 –
31 December 2022

www.morphemic.cloud

Deliverable reference
D5.6

Date
30 March 2021

Responsible partner
Engineering Ingegneria Informatica

Editor(s)
Daniele Pavia

Reviewers
Yiannis Verginadis
Katarzyna Materka

Distribution
Public

Availability
www.morphemic.cloud

Executive summary

This deliverable provides a detailed description of the software and the tools used to set up MORPHEMIC's testbed. More specifically, this document details what the testbed is and how it is used to deploy, test and ultimately validate both the platform developments and the use-case applications. This document is laid out following a top-down approach: the first chapter provides an introduction that defines the scope of the analysis, the target audience and a brief overview of its structure. The second chapter is focused on the architecture of the testbed along with the FiWare Lab and the NREC Cloud infrastructures, upon which the testbed infrastructure has been built. The third chapter, "Testbed Setup", details the technical procedures that went into creating and configuring the testbed: the instantiation of the necessary virtual machine, the requirements for the deployment of the MELODIC instances leveraged the initial use-case testing, details on the development instances of some of the recently introduced MORPHEMIC components. While the procedure of deploying MELODIC is better documented elsewhere, some troubleshooting-related details have been included in the Appendix A, MELODIC SETUP in order to have a quick reference of the post-installation activities that were performed when deploying the platform instances. Chapter 4 explains which procedures were followed to ensure that the software deployed within the testbed is actually working, while also including a brief overview of the initial use-case testing performed via the testbed by our partners. Next up in line is Chapter 5 which briefly summarises an analysis on the security of the testbed from a technical perspective, the outcomes of which will be included elsewhere as a contribution on improving MORPHEMIC security. Chapter 5 also contains the temporary security measures that have been enforced on the testbed while the project keeps improving upon its security standards. The document closes with some final considerations on the status of the testbed and the next steps that have been planned for the future, giving particular emphasis to the introduction of the ProActive Scheduler, which will have an impact on the relationship between the testbed lead node and the target infrastructures upon which the use-cases are deployed. Finally, the reader will find an addendum to the document under the guise of an appendix that documents some of the troubleshooting steps and fixes applied when dealing with issues arisen when deploying MELODIC on the testbed.

Author(s)

Daniele Pavia (ENG), Ciro Formisano (ENG)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871643

Revisions

Date	Version	Partner	Description
01.02.2021	Draft (0.1)	ENG	First Draft
15.02.2021	Draft (0.2)	ENG	Second Draft
07.03.2021	Draft (0.3)	ENG	Third Draft
16.03.2021	Draft (0.4)	ENG	Fourth Draft
23.03.2021	Draft (0.5)	ICCS	1st Internal Review
31.03.2021	0.7	ENG	Further addressing of feedback and layout fixes
02.04.2021	0.8	7bulls	2nd Internal Review
06.04.2021	0.9	ENG	Addressed 2nd Internal Review comments, ready for PMB submission
21.06.2021	1.0	ENG/ISW/CHUV	Addressed several comments from PMB members, added contributions from use-case partners, revised the document structure and improved its readability
02.09.2021	1.1	ENG	Addressed more comments, rebased document on newer layout, minor tweaks to styling and readability
21.09.2021	1.2	ENG/ICON/7Bulls	Completed the Backup Testbed Validation P4.4

Table of Contents

Index of tables

Index of figures

1 Introduction

1.1 The scope

This document is intended as a reference on the activities performed on the MORPHEMIC testbed, both to create the environment itself and the actions performed on it. The focus is on the environment where the MORPHEMIC platform and the project use-cases are deployed and tested. Specifically, it describes the infrastructures composing the testbed environment, its architecture and the procedures that were followed to setup it up. Additionally, information on the development instances of two new MOPRHEMIC components that are hosted within the testbed are included. This document contains a description of the actions performed to validate the testbed, including a brief overview of the initial use-cases deployment plus some considerations on the security of the testbed infrastructure. In particular, during the first year of the project (Y1), a preliminary analysis on the security capabilities provided by the MELODIC platform was performed. Such an analysis is aimed at providing useful information and proposing security-oriented improvements to the components of MELODIC that will be included in the MORPHEMIC platform. This deliverable will include a very brief summary of this activity and an explanation of the temporary security measures enforced on the testbed. The details of this analysis will convey, instead, into the deliverable *D4.2 Security design and implementation* [2].

1.2 Intended audience

This deliverable is intended for those involved in the software quality assurance process and their outcome:

- mandatory for test teams and architects:
  - the test team needs to know what the test case creation process is, what the life cycle of the test case is, and which elements the test case includes
  - architects must check whether the test cases are consistent with the (system) specifications
- recommended for developers – they should know what the life cycle of the test case is and how the system will be tested
- optional for the rest of the project members.

1.3 Document Organization

This document is organized in the following manner:

- Chapter 1 gives a brief introduction to the deliverable and its intended audience.
- Chapter 2 explains the testbed infrastructure architecture, its components and their interactions, while also offering some details on the Cloud environments on which the testbed has been deployed
- Chapter 3 gives a detailed report on how the testbed has been setup, how FiWare has been used to instantiate the VMs that host the services, how the MORPHEMIC components were deployed and configured and how the services were validated to ensure that everything is functioning as expected
- Chapter 4 offers some insight on the testing activities performed on the software deployed within the testbed lead node
- Chapter 5 gives an overview of the security issues of the MORPHEMIC instances hosted by the FiWare Lab infrastructure, briefly looking at their weaknesses and explaining the temporary measures adopted to mitigate the associated risks for the testbed
- Chapter 6 contains some final considerations on the status of the deliverable and its future evolution as part of the MORPHEMIC project development
- Chapter 8, Appendix A offers some details on the deployment of the MELODIC platform on the testbed, sharing some of the challenges and the troubleshooting steps that went into this activity

2 Testbed Architecture

Briefly describing it, the aim of the MORPHEMIC testbed is to deploy, test and validate the software releases of MORPHEMIC, including updates to its core functionalities and its components, and to be leveraged in order to deploy, test, and validate MORPHEMIC's use-case applications, either when deployed via MORPHEMIC or via other means. When looking at the testbed architecture, we can find two major components:

- the virtualization infrastructure called lead node, which hosts various MORPHEMIC and MELODIC platform instances plus a MOPRHEMIC development instance and some in-development components that have yet to be integrated
- additional virtualization infrastructures, such as public or private Clouds, that are provided by the use-case partners and which can be associated to each partner instance as deployment environments - called deployment nodes.

Please note that the terms *lead node* and *deployment nodes* are specifically used in this document to the describe the testbed architecture. At the time of writing, the testbed comprises two virtual infrastructures directly managed and hosted by the consortium: FiWare[1], which hosts the testbed lead node, and NREC[2], which has a role as the "fallback" deployment node. Additional deployment nodes may be associated to the instances hosted on the lead node, either offering virtualised or non-virtualised resources - such as, for example, bare-metal environments. More details on the interaction between the testbed lead node and its deployment nodes are provided in Section 2.2, Architecture Description. Please note that, at the time of writing, the NREC Cloud infrastructure is presented as fallback deployment node due to the fact that use-case partners are expected to provide their own means to access the appropriate infrastructure they wish to deploy their use-cases on. This is usually a public Cloud service among those that are supported by MORPHEMIC. Therefore, NREC acts as a backup deployment node that is leveraged either when a partner cannot provide his own means to access such an environment or to be leveraged for other purposes such as the testing of components that are not deployed via MORPHEMIC.

The use-case applications, which need a preliminary deployment in order to get a baseline upon which the benefits introduced by MORPHEMIC will be evaluated (*D6.2 Validation framework design* [3] and *D6.3 Use cases definition and preparation* [4]), exploit the testbed for testing and validation as further documented in Chapter 4 Testbed Validation. In order to speed up this activities, three instances of the MELODIC platform, one per use-case partner, have been deployed on the testbed. MELODIC will provide an efficient deployment mechanism for the use-case applications and a baseline upon which it will be possible to evaluate the benefits introduced through Polymorphic and Proactive Adaptation, which are the most significant improvements introduced by MORPHEMIC. Since MORPHEMIC exploits MELODIC at its core, this is approach eases the testing of the initial use-case releases as they are developed.

In the final phase of approval of this Deliverable, a major problem impacted the FiWare based infrastructure hosting the testbed lead node. Considering that the testbed is critical for the project activities, and considering that the lead node cannot be automatically replaced (such as the deployment nodes), the partners considered the possibility to include a *backup lead node* infrastructure to allow quick recovery in case of failures avoiding to slow down the experiments. This setup is, at the time of writing, being implemented. Once the FiWare node will be available again, the testbed lead node is going to be brought back to it.

2.1 Architecture Description

MORPHEMIC is constituted by a number of services tied together with the common goal of offering smart multi-Cloud support, being capable of intelligently deploying diverse application components with different characteristics within given constraints and requirements, from budgetary limits to specialised hardware (see *D4.3 Selection design and implementation of integration layer* [5] for reference). While MORPHEMIC boasts a microservice architecture, the typical deployment sees each instance deployed on a pretty capable - in terms of hardware resources - virtual machine. Each MORPHEMIC instance is then configured to gain access to one or more environments that offer either physical of virtualised resources so as to deploy the project use-cases by leveraging one of its most relevant features, the

[1] https://www.fiware.org/about-us/
[2] https://www.uio.no/english/services/it/hosting/iaas/

Polymorphic Adaptation. The testbed architecture reflects these characteristics in its present embodiment. As such, the testbed architecture presents one node - called here the lead node - that hosts the MORPHEMIC platform instances and any number of additional infrastructures – called from here on, deployment nodes - required for use-case deployment.
In the most essential form of interaction, testbed users willing to deploy and test their use-case via MORPHEMIC connect to their own platform instance hosted within the testbed lead node, upload their CAMEL model and let the platform deploy their use-case application - within the given service-level objectives and optimisations - on the deployment node that is selected according to the MORPHEMIC reasoning process. This workflow reflects a real-world MORPHEMIC scenario, further compounding the usefulness of the testbed as a mean to validate the use-case development. For more information on the subject of use-cases, the reader is encouraged to refer to the industrial requirements analysis in *D6.1 Industrial requirements analysis* [7].
An exemplified overview that abstracts the testbed architecture is shown in Figure 1 Testbed Architecture Overview. Here we can identify the two main blocks of the testbed architecture: the lead node and the deployment nodes.

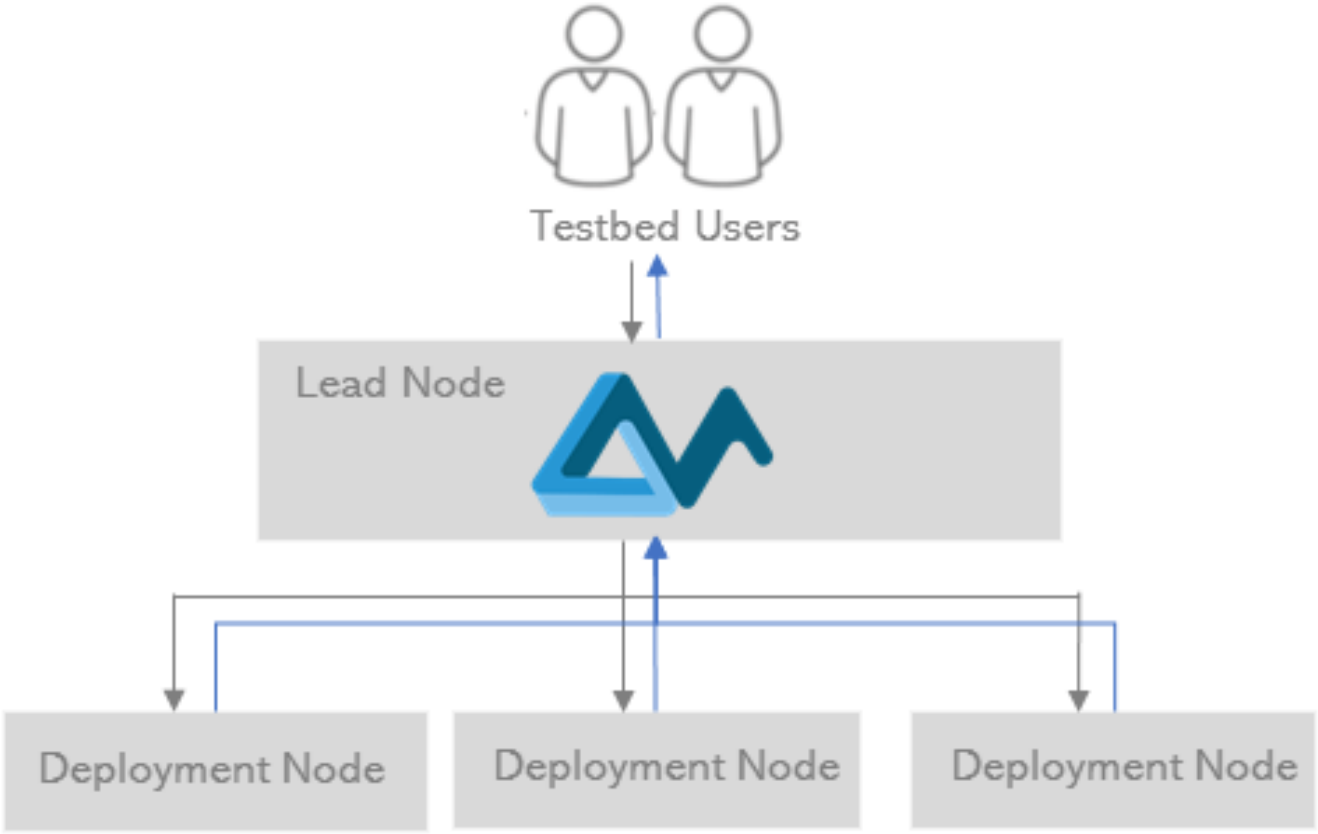

Figure 1 Testbed Architecture Overview

As previously said, the testbed lead node runs on FiWare, or, more specifically, on the FiWare Lab infrastructure, while, at the time of writing, the supported Cloud deployment nodes are the OpenStack-based NREC provided by the University of Oslo, Amazon Web Services, Google Cloud Platform and Microsoft Azure.
The interactions happening within the MORPHEMIC testbed between the users, the lead node, the partner's MORPHEMIC instances and the deployment nodes are exemplified in Figure 2 Testbed Architecture Components.

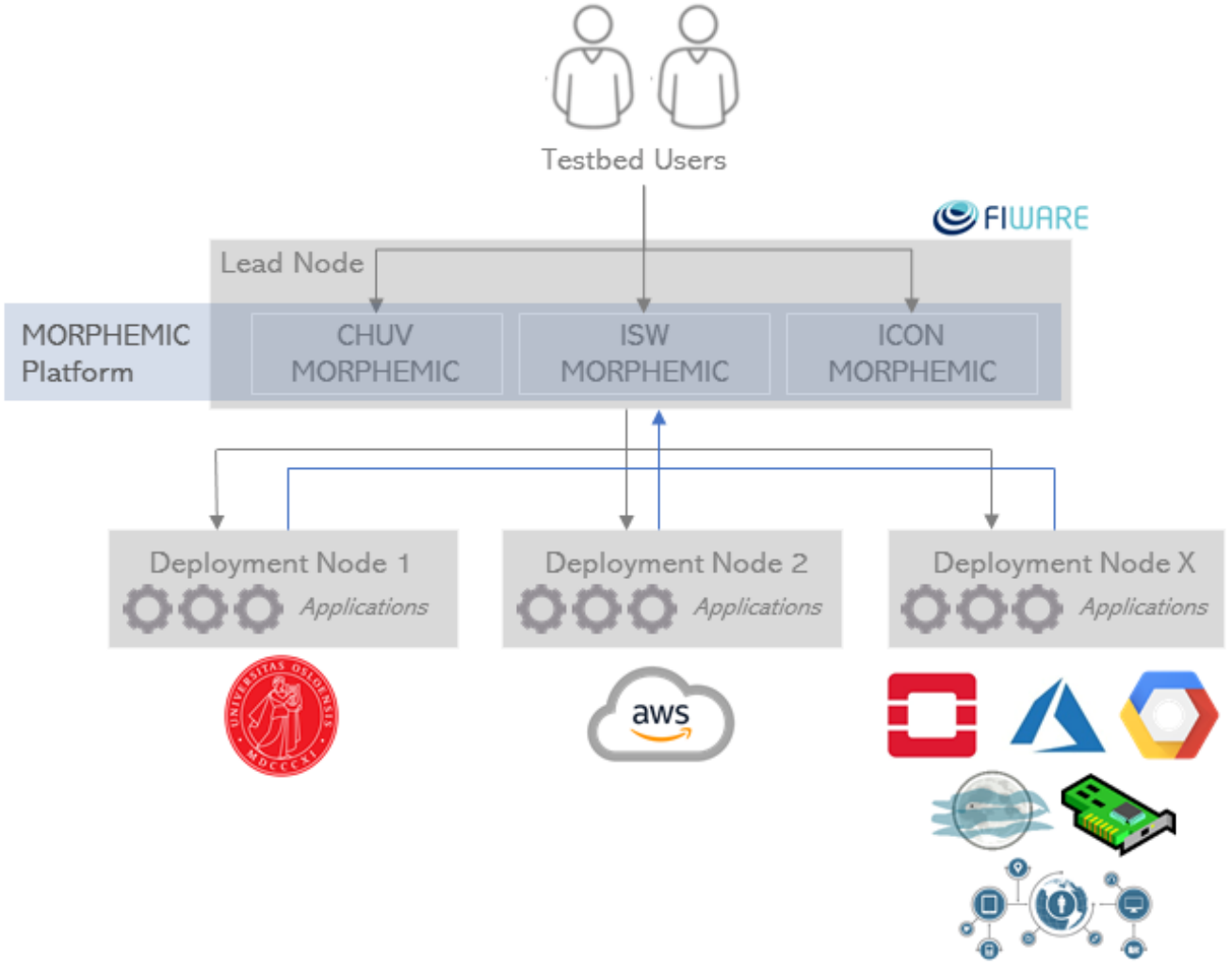

Figure 2 Testbed Architecture Components

One important aspect of MORPHEMIC which marks a relevant evolutionary step from its predecessor MELODIC, is that while the latter already offers multi-Cloud support via its Cloudiator[3] components, the former will extend these capabilities by replacing Cloudiator with the Proactive Scheduler[4] [8, Ch. 3][5, Ch. 3], thus introducing support for other deployment nodes such as FPGAs, bare-metal, Fog and Edge computing infrastructures. This is a critical aspect when it comes to enabling the Polymorphic Adaptation feature, which is one of the defining aspects of the project – see *D3.3 Optimized planning and adaptation approach* [6]. Figure 2 Testbed Architecture Components also depicts the support for such diverse deployment nodes, since future MORPHEMIC developments are planned to deliver on the capabilities required to support those environments. It is worth noting that the testbed does not directly offer the hardware support that is needed enable some of those technologies via the private infrastructures provided by the consortium, such as FPGA, GPU acceleration and so on. These will be supported instead by adding suitable target nodes that possess such capabilities. For example, FPGA support might be included by leveraging AWS specific offerings for such hardware.
While considering user interaction with the testbed, use-case partners are not the only actors. Testbed Administrators can access the underlying platform management by authenticating to a VPN gateway, while MORPHEMIC developers can establish either a direct connection via SSH or a WebSSH connection to the testbed machines via MELODIC WebUI, enabling the former to manage the infrastructure and the latter to interact with the software hosted therein. This enables privileged users to, for example, analyse logs, fix issues, test MORPHEMIC components, restart instances and so on. It should be noted that, as further explained in the following chapters of this deliverable, some of the services published by the testbed instances hosted within the testbed infrastructure offer their own web user interfaces. These

[3] https://github.com/cloudiator
[4] https://proactive.activeeon.com/

Web UIs are expected to be publicly available in order to access their respective functionalities. In particular, MORPHEMIC, as part of its offerings, publishes several web interfaces. First and foremost, through the main Web UI that use-case partners users access to deploy their applications, then through other interfaces offered by MORPHEMIC components such as Grafana[5], WebSSH[6], etc. Furthermore, some of the services exposed by MORPHEMIC may be accessed by remote instances deployed via the platform itself. For example, artefacts deployed by the ICON use-case connect to the EMS service provided by their MORPHEMIC instance. The testbed lead node enables use-case partners to gain access to these services by publishing their web interfaces via Security Groups, thus allowing them to access the expected functionalities and the tools that are critical to exploit their respective MOPRHEMIC instances and that of the deployed use-case applications. Please refer to Figure 3 Testbed Access for an overview of this workflow.

While hosting several instances of the MORPHEMIC platform, the testbed support also extends to the development of individual components that will be integrated in MORPHEMIC. This is an offering of the testbed towards the developers of MORPHEMIC so that they can leverage an infrastructure at no cost in order to develop, deploy and test their component. At the time of writing, the lead node is hosting one development instance of the Web Crawler and one of the Downloader, with plans to have both deployed in conjunction with each MORPHEMIC instance once the undergoing integration work will be completed. The Web Crawler is a MORPHEMIC component that provides the tools required to crawl open-source software repositories in order to identify suitable projects according to MORPHEMIC requirements, meanwhile the Downloader works in concert with the Web Crawler by fetching remote artifacts based on data mined. For more information on the Web Crawler and the Downloader, the reader is encouraged to refer to *MORPHEMIC Deliverable D3.1 Software, tools, and repositories for code mining* [1]. As an example of what has been achieved by employing the testbed, this particular Web Crawler instance has been used to deploy, develop and test the software component. Formal testing of the Web Crawler has been conducted by accessing its APIs that have been published as part of the testbed Web Interfaces access layer. The testbed additionally hosts a MORPHEMIC development instance, managed by 7Bulls, in order to test the current builds of the software, with plans to leverage this instance to conduct the first use-case demonstrations.

An overview of the way that MORPHEMIC users, developers and testbed administrators interact with the testbed and the instances hosted within can be found in Figure 3 Testbed Access.

[5] https://grafana.com/
[6] https://en.wikipedia.org/wiki/Web-based_SSH

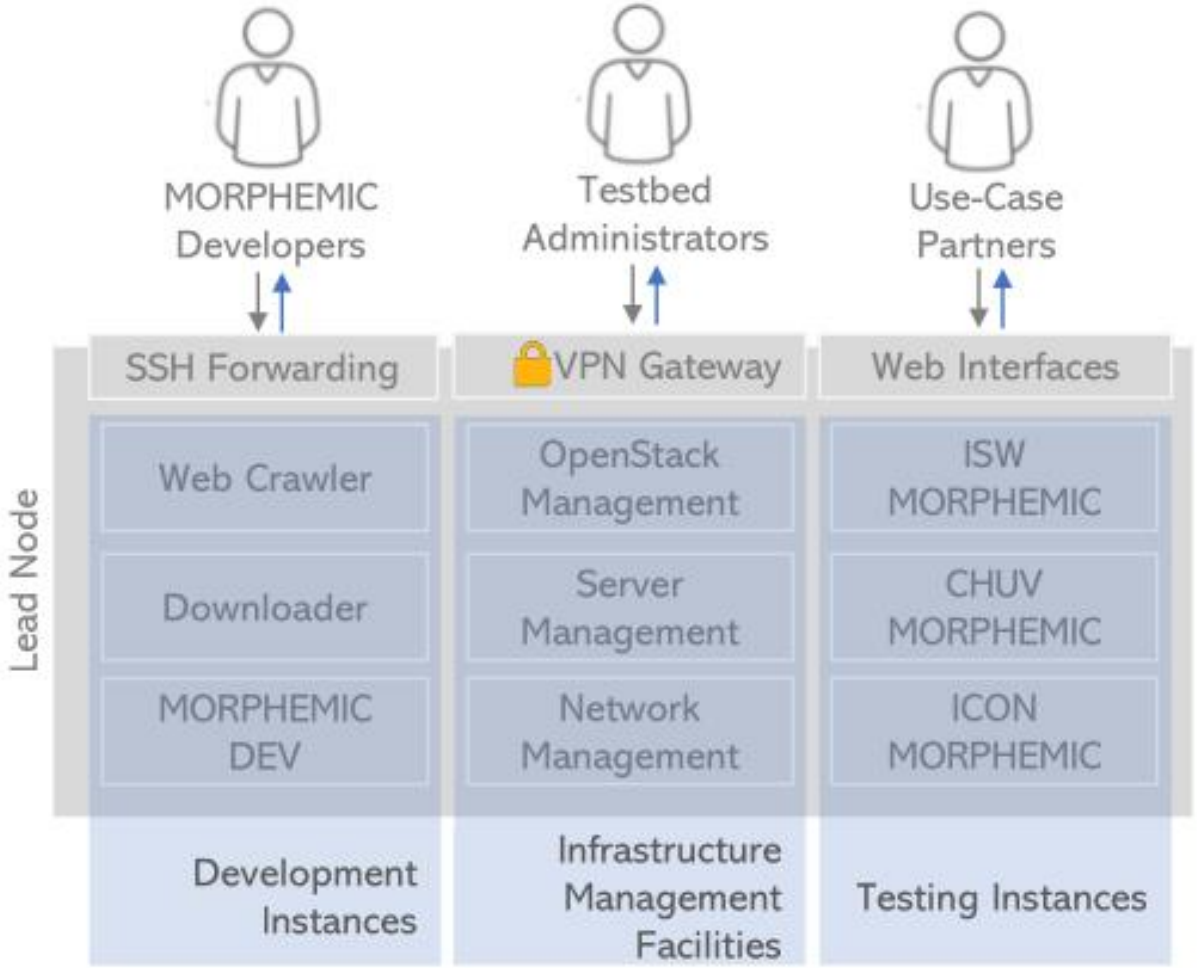

Figure 3 Testbed Access

While looking at Figure 3 Testbed Access, the reader can identify the following components:

- SSH Forwarding, leveraged by MORPHEMIIC developers to directly access via SSH the development instances:
  - Web Crawler
  - Downloader
  - MORPHEMIC-DEV
- VPN Gateway, which encrypts network traffic and enables direct connection to the underlying infrastructure while limiting access to authenticated testbed administrators. This gateway routes traffic toward the infrastructure management facilities, such as:
  - OpenStack management - since FiWare is an OpenStack-based infrastructure, testbed administrators can directly interact with the underlying Cloud platform
  - Server management, enabling administrators to connect directly to the Cloud infrastructure nodes (via ILOM, O.S., etc.)
  - Network Management, in order to let the administrators inspect and configure networking
  - VPN Gateway, encrypts traffic and enables direct connection to the Virtual Machines that host the MORPHEMIC instances
- Web Interfaces enable use-case partners to access the services exposed by the testing instances via the infrastructure security groups and firewall – see Table 2 MELODIC Security Group

It is worth noting that the testbed is available to host any other in-development component that is yet to be integrated to MORPHEMIC, should the opportunity arise. For a full list of the MORPHEMIC software components and their interaction within the platform architecture the reader should refer to *D4.1 Architecture of pre-processor and proactive reconfiguration* [8].

2.2 Cloud Infrastructures

This chapter offers a brief description of the Cloud Infrastructures involved in the testbed. Both the FiWare and the NREC infrastructures are, as described below, based on the community maintained of OpenStack, therefore free and open-source software. It should be noted that, despite the existence of support for AWS, Google Cloud Platform, Azure as deployment nodes, this chapter will not delve into describing these proprietary platforms as it is considered out of the scope of this deliverable. Furthermore, there is plenty of documentation on the subject of AWS[7], Google Cloud Platform[8] and Azure[9] architectures available online. Also, the use of a specific public cloud resource is a responsibility of each use-case partner and not of the testbed itself. Thus, the deployment nodes can vary and could depend both on the use-case requirements but also on the ability of the platform to fetch the right public cloud offers and select the most suited among those for application deployment purposes. The latest subsection contains a brief description of the backup lead node infrastructure.

2.2.1 FiWare

As explained in its official website[10], FiWare "is a curated framework of open-source platform components to accelerate the development of smart solutions". From a technical standpoint, FiWare is a federation of OpenStack-based Cloud instances that share a common authentication end point. The testbed lead node is hosted on the FiWare Lab Cloud infrastructure, based on OpenStack Kilo, which is currently part of the federated network of infrastructures that compose FiWare. FiWare is currently steered by the FiWare foundation, a non-profit organisation that has been established in 2016 as a way to move forward after a successful research project by the same name, founded by the European Commission as part of the Horizon 2020 initiative.

Figure 4 FiWare Lab Infrastructure summarises the architecture of the FiWare node hosted by Engineering that contains the testbed Lead Node and its MORPHEMIC platform components.

[7] Further information on the topic of AWS services and their architecture can be found here: https://aws.amazon.com/it/blogs/architecture/
[8] https://cloud.google.com/gcp/
[9] https://docs.microsoft.com/en-us/azure/architecture/browse/
[10] https://www.FiWare.org/about-us/

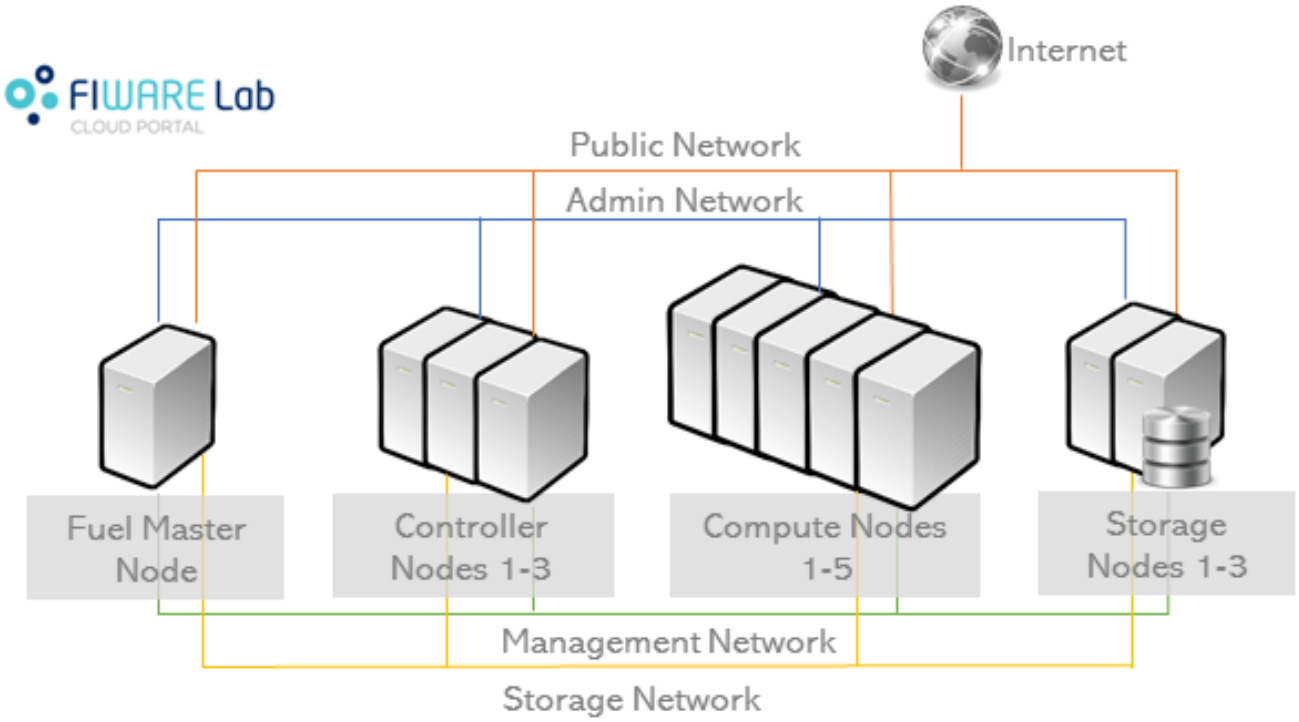

Figure 4 FiWare Lab Infrastructure

The main components that constitute the FiWare Lab infrastructure Cloud platform are:

- Fuel Master Node – this node has been employed to deploy the actual OpenStack instance, it offers a number of tools to create custom OpenStack deployments
- Controller Nodes – these nodes host all the OpenStack components that are required to administer the OpenStack instance, its resources and its VMs. One node acts as the main, while the other two are redundant and kept ready in case the main controller fails
- Compute Nodes – these nodes are dedicated to the actual execution of the VMs
- Storage Nodes – as the name implies, these nodes offer the storage space required to host the VMs virtual disks. These nodes employ a shared filesystem in order to pool together the storage space that is available to the OpenStack instance

For reference, here are included the hardware specifications of the Fiware FiLab infrastructure:

Table 1 FiWare FiLab Hardware Specifications

Node	Model	CPU	RAM	Storage	Network
Fuel Master	Centos 7-based Virtual Machine	4 Cores	8 GB	80GB	2 network cards at 1.0 Gbps
Controller	Dell PowerEdge M630	2x Intel Xeon E5-2670 v3 30 MB cache @2,30 GHz , 12 cores with 24 threads	384 GB ECC	2x 300GB Disks in RAID 1	8x network cards at 10 Gbps
Compute	Dell PowerEdge M630	2x Intel Xeon E5-2670 v3 30 MB cache @2,30	384 GB ECC	2x 300GB Disks in RAID 1	8x network cards at 10 Gbps

		GHz , 12 cores with 24 threads			
Storage	Dell Inc. PowerEdge R730xd	CPU: 2x Intel Xeon Processor E5-2620 v3 15M Cache, 2.40 GHz, 6 cores with 12 threads	128 GB	2x 300GB Disks in RAID 1, 17x 1,1 TB Disks in RAID 6	4x network cards at 10 Gbps, 2 network cards at 1.0 Gbps

In terms of hardware resources 45 cores, 288 GBs of RAM, 740GBs of storage and 7 public Ips are currently allocated to the project, plus access to an internal 1Gb network that connects the physical servers. These resources are in use at the time of writing in order to enable the hosting of the MORPHEMIC lead node VMs, with more resources available should the need arise.

2.2.2 University of Oslo NREC Cloud Platform

The Norwegian Research and Education Cloud, in short NREC, is a cloud infrastructure for research and education, provided in collaboration between the University of Oslo and the University of Bergen with additional sponsorships from NeIC (Nordic e-Infrastructure Collaboration).

NREC is an Infrastructure-as-a-Service (IaaS) cloud infrastructure that enables users from various Norwegian universities or colleagues to spawn standardised servers and storage instantly, as needed, via a web-based self-service.

NREC has been in production since 2016 and is currently providing cloud infrastructure for several high-profile academic projects, including CERN's ALICE and ATLAS experiments. Hardware is located exclusively on-premise in Oslo and Bergen and services are developed locally and almost entirely based on open-source software and open standards, making NREC a more transparent alternative to commercial cloud providers. NREC is a community cloud aiming to provide a modern, flexible and secure IT infrastructure, tailored to the needs of the research and higher education sector.

NREC is built on OpenStack, which is a large framework for building flexible, elastic and modern cloud services. Within the frame of this service, the user is given the freedom to choose the operating system and application stack, the maintenance of which is the user's responsibility.

As operating systems several Linux distributions (Cent-OS, Debian, Fedora, Ubuntu) and one Windows version are available to choose from. The available versions of each distribution change over time as new releases are published and older ones phased out.

Hardware resources for the virtual machine instances can be picked from a set of currently five pre-defined "flavors", ranging between 1 virtual CPU with 512 MB RAM to 4 virtual CPUs with 16 GiB RAM.

These relatively humble per-machine resources are in-line with OpenStack's philosophy. OpenStack is designed to be horizontally scalable. Rather than switching to larger servers, users are supposed to procure more servers and simply install additional identically configured services.

Depending on what is needed for the specific use-case, block storage space can be assigned to an instance or shared between multiple instances.

Users can choose between public or private IPv6 and combined IPv4/IPv6 network configurations and can provide firewall rules for ports or IP-address ranges as needed.

NREC also supports several ways of adding records into the Domain Name System (DNS) to enable a more convenient public access to hosted instances than plain IP addresses.

Running instances can be accessed on the operating system level via SSH or – on Windows instances only – Remote Desktop protocol (RDP).

The whole experience from setting-up an instance, starting and stopping it is provided as a self-service using an easy-to-use web-based user interface that is also prepared for configuration scripts and custom images for easier deployment of identical instances.

Support of High-performance computing (HPC) within NREC is currently in a beta-state. In contrast to normal instances HPC instances are handled in a different way in regard to underlying hardware and usage policies due to the different

nature of HPC applications. HPC instances use 32 or 48 core AMD EPYC processors, while normal instances use various models and generations of Intel processors. In contrast to normal instances, which may have to share resources, CPU and memory for HPC are exclusive to the instance they are allocated to. Available "flavors" for instances range from 8 virtual CPUs with 30 GiB RAM to 64 virtual CPUs with 240 GiB RAM (subject to change due to the beta phase).
Another beta programme in NREC offers the use of Virtual GPU accelerated instances. vGPU instances use NVIDIA Tesla V100 or P40 GPUs with Intel Xeon processors, depending on the data centre used. The GPU is shared between 2-3 instances and "flavors" range from 2 virtual CPUs with 8 GiB RAM to 8 virtual CPUs with 32 GiB RAM (subject to change due to the beta phase).
The Numascale cluster at the University of Oslo was provided by Oslo-based company Numascale AS, which specializes in the development of unique node controller hardware technology that enables a cost-effective way to share CPUs, memory and caches in clustered nodes and act as one machine. The cluster at UiO uses Numascale's Numachip1 hardware to unite 68 nodes into one cluster with a total of 408 AMD Opteron 6174 processors and 1 TB of RAM. The operating system used is CentOS. The goal of bringing the Numascale cluster together with NREC is to make it easier and more convenient for researchers to work with the cluster's resources. At the time of writing, preparatory work on the cluster is ongoing, with the installation of OpenStack as one of the next steps.
The integration of the Numascale machine into NREC's self-service user interface will enable quick and easy set-up and use of the cluster's computing power while keeping administrative and maintenance efforts low.

2.2.3 Backup Lead Node

Engineering D.HUB[11] is part of Engineering Group and offers several innovative solutions, among which Cloud Infrastructure services have crucial importance. The services provided by D.HUB are consolidated *commercial products*, including support and specific *Service Level Agreements*. On this sense, while the FiWare Lab is considered an exploitable asset in MORPHEMIC [10], the D.HUB infrastructure is used by Engineering R&D Group as internal customer. This consideration defines the hierarchy of the two lead node infrastructures: FiWare is the main one, as strategic exploitable item, D.HUB is the backup to support the MORPHEMIC's activities if needed.
D.HUB offers a managed service for a diverse array of Cloud infrastructures, either off and on-premises, including OpenStack, VMWare and AWS. The main requirement of a backup infrastructure is to become available when needed with minimum effort. Due to its extensive usage during the development of MELODIC and MORPHEMIC, the Cloud infrastructure that required the minimal effort has proven to be AWS: for this reason, AWS has been chosen as the backup infrastructure for the Testbed lead node.
The backup lead node infrastructure must be able to replace the virtual machines used on FiWare lab to host the instances of MELODIC/MORPHEMIC. At the time of writing this Deliverable, it has been decided that each VM hosting a backup instance of MELODIC/MORPHEMIC should provide the following features:

- Instance Type: m5a.4xlarge
- vCPUs: 16
- Memory: 64GB
- Network Performance: Up to 10 Gigabit
- Volume Size: 100GB.

The current version of MORPHEMIC support one instance per use-case: this means that the minimal backup lead node should be capable of hosting three virtual machines with the aforementioned characteristics, in order to offer a suitable temporary replacement for the FiWare infrastructure in the event that a critical failure occurs.
Moreover, should the backup Testbed *lead node* fail, NREC offers yet another possibility to host temporary MORPHEMIC instances, enabling use-case partners to directly deploy to their AWS/Azure/OpenStack accounts. This effectively doubles the possibilities for having a working environment should the main Testbed *lead node* fail.

[11] https://www.eng.it/en/who-we-are/engineering-group/engineering-dhub

3 Testbed Setup

In this section, the steps that are required to setup the MORPHEMIC testbed are explained in-depth. The objective of this chapter is both to explain how the testbed was deployed and to facilitate the reproduction of a similar setup on other premises. This chapter details the testbed installation processes, including the process of instantiating the required virtual machines on the FiWare Lab, deploying MORPHEMIC components, configuring the environment and running tests to verify that the software has been successfully deployed. It should be noted that, since the testbed VPN gateway is a generic OpenVPN instance hosted on a CentOS 7 Virtual Machine, this part of the setup process will not be detailed as it is considered trivial – therefore, it will not be included in this deliverable. Should the reader be interested in knowing more on the subject of deploying an OpenVPN server on CentOS 7, detailed instructions may be found elsewhere[12]. It is also worth noting that this chapter does not include the installation instructions for MELODIC. The reader should instead refer to the MELODIC official installation and deployment documentation instead[13]. More information on the subject is also included in Chapter 8 Appendix A.

3.1 Lead Node Setup

In order to deploy the software required to execute the project use-cases via testbed, three different virtual machines, one per use-case partner, have been instantiated. Each of these VMs host one MELODIC instance. This setup has been deemed necessary due to the intentional lack of multi-tenancy support in MELODIC. This configuration will be retained even after each MELODIC instance will be replaced with MORPHEMIC, since that is the intended user model for MORPHEMIC. It should be noted that each MORPHEMIC instance user will be able to deploy multiple applications. The process of deploying virtual machines on FiWare requires having access to a valid account and a tenant with a set of pre-allocated resources. Thus, access to FiWare resources requires a registered user[14].

Access to the FiWare Lab that hosts the testbed can happen by visiting the login page[15], called Cloud Portal. Once a user has gained access to the FiWare user interface, it is advisable to begin the testbed setup by creating the security groups that will be necessary in order to allow external access to the services that will be deployed. Also, the generation of SSH keypairs[16] is required. As the name implies, SSH keypairs will grant SSH access to the VMs themselves. By default, every VM created on a FiWare node can be accessed exclusively by leveraging a private SSH key. It is therefore critical to keep any generated SSH keypair backed up in a safe location in order to avoid losing access to the VMs.

Figure 5 FiWare Interface Overview offers a sample of the Cloud Portal user interface that will be used for the activities explained in this paragraph.

[12] For example: https://www.digitalocean.com/community/tutorials/how-to-set-up-and-configure-an-openvpn-server-on-centos-7
[13] Official MELODIC documentation may be found here: https://MELODIC.cloud/get-started/
[14] Potential FiWare users should apply here for registration
[15] https://cloud.lab.FiWare.org/auth/login/
[16] See here for more information on the subject

In order to create a new keypair, the user needs to locate the “Access & Security” section of the “Overview” section.

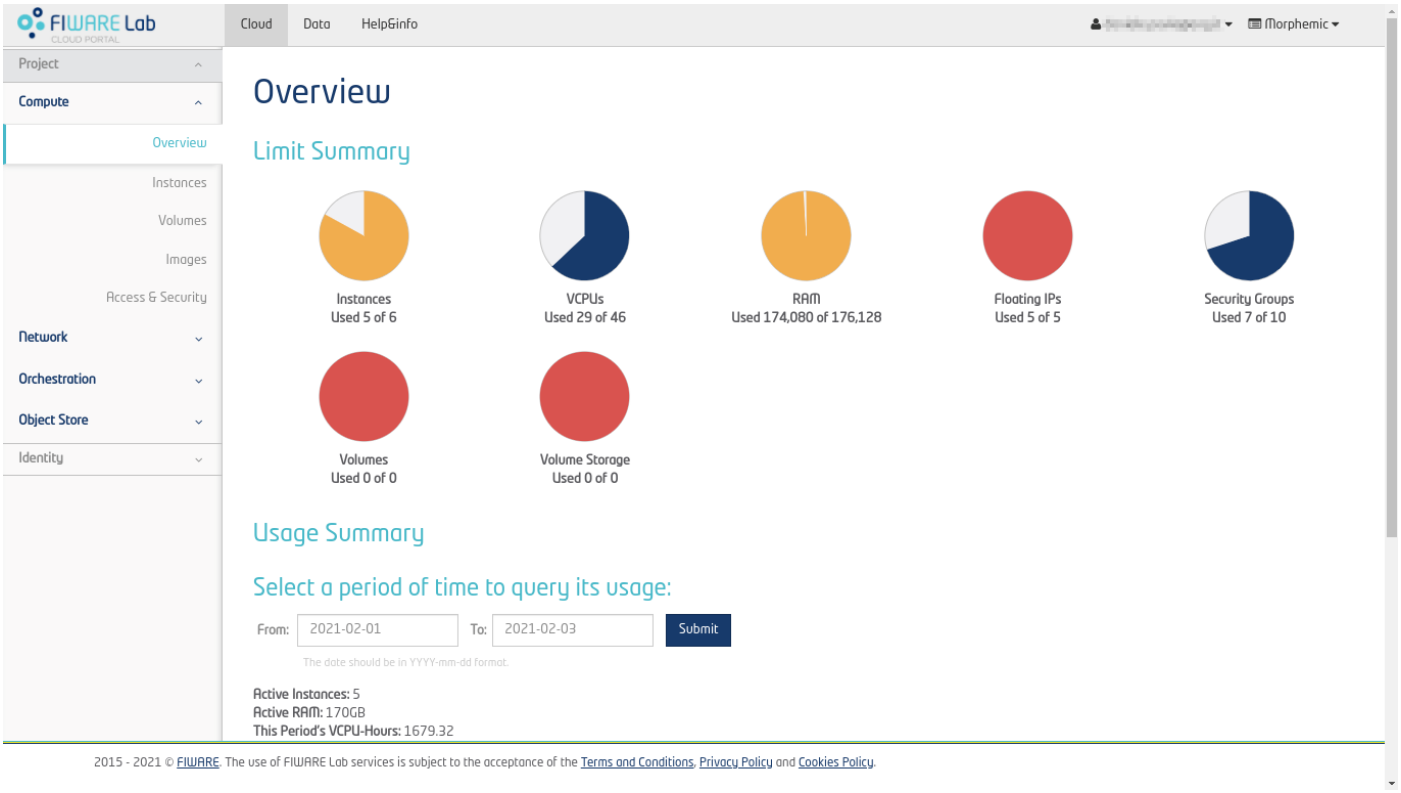

Figure 5 FiWare Interface Overview

Once there, the user follows the “Create a New Keypair” wizard under the “Key Pairs” tab to create the keypair that will be used to access the VMs. SSH keypairs are used within the MORPHEMIC testbed in order to gain access to each VM via a remote session, typically for administrative purposes.

Meanwhile, security groups act as a sort of port forwarding between the private IP of the VM, allocated on an internal OpenStack network, and its public IP, also known as floating IP. Only network ports explicitly defined in a security group that is assigned to a specific VM are publicly available. Therefore, it is necessary that every service that needs to be exposed on a public network - i.e., the internet - has its network ports added to a security group, which, in turn, must be assigned to the correct VM; otherwise, the services might not be available or exhibit unexpected behaviour.

In order to create a new security group, users should locate the “Access & Security” tab under the “Overview” section and access the “Create Security Group” functionality under the “Security Groups” tab. There is a “default” security group that is assigned to every VM by default. This security group enables traffic on network port number 22, therefore allowing access to the VM via SSH. Users that optionally decide to leverage the VPN gateway may remove this security group from the instantiated VMs in order to limit public exposure of the SSH service. In fact, SSH is often abused with random brute-force attacks, hence denying public access to this service helps improving the security of the VM. Please note that this document assumes that outbound network traffic is not restricted; therefore, only inbound traffic rules will be specified.

The following Table 2 MELODIC Security Group details how the security group for the MELODIC instances has been created according to the requirements of the exposed services.

Table 2 MELODIC Security Group

Direction	Type	Protocol	Port Range	Service
Ingress	IPv4	TCP	80	Melodic UI frontend
Ingress	IPv4	TCP	443	Melodic UI frontend SSL
Ingress	IPv4	TCP	8088	ESB REST API
Ingress	IPv4	TCP	8095	Camunda UI

Direction	Type	Protocol	Port Range	Service
Ingress	IPv4	TCP	8181	Spark
Ingress	IPv4	TCP	8998	Spark
Ingress	IPv4	TCP	7077	Spark
Ingress	IPv4	TCP	38000	Spark
Ingress	IPv4	TCP	38100	Spark
Ingress	IPv4	TCP	38200	Spark
Ingress	IPv4	TCP	38300	Spark
Ingress	IPv4	TCP	38400	Spark
Ingress	IPv4	TCP	38500	Spark
Ingress	IPv4	TCP	8080	Cloudiator WebUI
Ingress	IPv4	TCP	4001	etc registry
Ingress	IPv4	TCP	9000	Cloudiator REST API
Ingress	IPv4	TCP	33034	Cloudiator rmi registry
Ingress	IPv4	TCP	61610-61619	ActiveMQ event broker ports
Ingress	IPv4	TCP	2222	Baguette server port
Ingress	IPv4	TCP	1099	ActiveMQ JMX connector port
Ingress	IPv4	TCP	8111	EMS REST API
Ingress	IPv4	TCP	8078	Melodic UI backend
Ingress	IPv4	TCP	2036	CDO Server
Ingress	IPv4	TCP	3077	JWT
Ingress	IPv4	TCP	2121	webssh
Ingress	IPv4	TCP	3000	Grafana
Ingress	IPv4	TCP	8123	mq-http-adapter/UI

Once both keypairs and security groups are created, the next step is to actually instantiate the VMs that will host the platform instances. Each VM will be deployed using a specific flavour depending on the system requirements of each component installed.

To instantiate a VM on FiWare, the user needs to select the "*Launch Instance*" button placed under *Compute -> Instances*. This executes a new instance wizard where the user selects the sizing - also known in OpenStack as *flavour* - network, security groups and keypairs according to their requirements. This procedure has been repeated for each use-case partner MELODIC and for the MORPHEMIC-DEV VM. Once the VMs are created, the final step before the actual deployment of MORPHEMIC is to associate to each VM to floating IP. This enables the running services exposed via specified security groups to be available from the internet.

Once every VM is ready and fully functional, both MELODIC and the in-development components have been deployed as depicted in the following paragraphs.

3.2 MELODIC Deployment

The installation procedure that explains the process to deploy MELODIC is already publicly available at the official melodic.cloud[17] website; therefore, it will not be included in this deliverable. This paragraph contains MELODIC software requirements as a reference for what has been actively deployed on the testbed, while Appendix A delves more into how the instances were configured to work with NREC as a deployment node and the means to overcome the challenges that were faced when deploying MELODIC within FiWare.

[17] Official MELODIC installation Guide: https://MELODIC.cloud/wp-content/uploads/2019/10/MELODIC-installation-guide.pdf

In terms of requirements, deploying MELODIC on the testbed has shown that, when it comes to VM sizing, it is advisable to allocate a higher amount of RAM than suggested in the software requirements found inside the official documentation. All testbed instances of MELODIC have been deployed with 64GB of RAM to avoid memory saturation, failure of containers ungracefully halted by the OOM killer and general slowdowns due to the "swappiness" caused by memory exhaustion. In future updates to testbed software, an improvement is expected with the first release of the MORPHEMIC platform, where the Cloudiator portions of MELODIC has been superseded by the ProActive Scheduler, together with other developments such as the deprecation of Spark, which will be stripped from the platform due to the dynamic development of its framework. It has been deemed that the proper solution was not to couple MORPHEMIC to a specific version of the Spark framework and, in case of a Spark based application, the Spark framework would still be deployed using the proper configuration in the application's CAMEL model. Furthermore, there is work undergoing to reduce the size of the docker containers that are used to deploy MORPHEMIC components by tailoring the containers to the project needs instead of reusing more generic images that include software that is not strictly required by the components themselves. Overall, it is expected that these changes will drive memory requirements down.

Once MELODIC has been installed, users will be able to actually deploy the in-development MORPHEMIC use cases only after one or more Cloud providers are configured to host the use case application instances. As explained in Chapter 2.1 Architecture Description, MORPHEMIC's testbed is configured to leverage NREC as the fallback deployment node; however, partners that are able to target either AWS or any other supported infrastructure such as OpenStack, Google Cloud Platform and Azure, are expected to do so in order to execute and validate their use-cases. More information on Cloud provider configuration in MELODIC can be found in Appendix A.

3.3 Web Crawler Deployment

This paragraph gives a step-by-step guide on how the Web Crawler development instance has been deployed on the testbed. It is worth noting that the procedure explained here reflects the state of the Web Crawler as a single component that has yet to be fully integrated with the MORPHEMIC platform. In the future, as the platform development progresses, the Web Crawler will be distributed together with MORPHEMIC as a fully integrated component. As MORPHEMIC components are distributed via Docker images, it is expected that the Web Crawler will be redistributed under such form, in that it will be installed and configured via the same MORPHEMIC installation procedure that deploys the rest of the platform. Once the Web Crawler will be tightly integrated to the platform, actions that will be detailed later in this chapter, such as installing dependencies like MariaDB – a step that will not be required as the Web Crawler will share the same MORPHEMIC SQL database instance with the other components - and compiling Python libraries from their sources will be taken care of automatically at deployment time. Nonetheless, the following procedure is still worth including both as it is, at the time of writing, the only way of properly deploying the Web Crawler just as it has been instantiated within the testbed, and as a reference for the work that will need to be automated during the future development of the platform, hence the editor's will to include this guide in this deliverable. It is worth noting that a similar process can be followed by use-case partners to install the use-cases and standalone applications, ergo not deployed via MORPHEMIC, making it also relevant to them.

The Crawler is mostly written in Python and uses a MariaDB instance to store its data. Deploying and using the Web Crawler is essentially a matter of installing its software dependencies, setting up its database, pulling its code from a repository, customising its configuration and, finally, executing it. As it will be seen during the installation procedure, since the Crawler is based on Python 2.7, a few of its dependencies are legacy libraries that either need to be installed at a specific tagged version - usually by employing the pip package manager – or compiled from source. For this reason, the following guide makes use of a Python virtualenv[18] in order to segregate the Web Crawler working environment from that of the operative system in order to avoid the clashing of conflicting versions of the same libraries, thereby preventing from generating undesired issues for both the crawler and the hosting VM Operative System.

When executed, the Web Crawler spawns four different Python processes, each with a specific job:

- Orchestrator - as the name implies, this process manages the execution of the Crawler tasks

[18] See https://docs.python-guide.org/dev/virtualenvs/ for more information on Python virtualenvs

- Notifier - internally notifies when there are batches of data that have been integrated
- Repository Crawler - this process is devoted to data scraping from various supported code repositories
- Pserve - a simple Python-based web server used to expose the Web Crawler API

The software and hardware requirements to successfully deploy and execute the crawler are depicted in Table 3 Web Crawler Software Requirements and Table 4 Web Crawler Hardware Requirements.

Table 3 Web Crawler Software Requirements

Operative System	Development Environment	Database
Ubuntu 18.04	Python 2.7 Python Pip Python Virtualenv	MariaDB

Table 4 Web Crawler Hardware Requirements

Hardware Requirements	CPU	RAM	Storage
Minimal	1 CPU core	2GB of RAM	200GB of storage space
Recommended	4 CPU cores	4GB of RAM	1TB or more

Once the VM has been set up according to the hardware and software requirements, the user should proceed with the installation of the Web Crawler by authenticating to the VM with an administrative account and executing the following steps:

– Install dependencies and enable MariaDB, then secure the instance:

```
apt install -y mariadb-server mariadb-client libmariadb-dev-compat gcc libpython2.7-dev libmariadbclient-dev
systemctl enable --now mariadb
mysql_secure_installation
```

– Create the *markos* user:

```
useradd -d /opt/markos -m markos
```

– Switch to the *markos* user and create a python virtualenv. This step is required in order to isolate the Python libraries that will be installed from those of the operative system in order to avoid issues due to conflicting versions. Once created, the user should activate it:

```
su markos
cd $HOME
virtualenv markos-virtualenv
source markos-virtualenv/bin/activate
```

– Clone the Web Crawler Git repository and setup the development environment, downloading the required dependencies. Please note that users willing to access the Web Crawler Gitlab repository need to ask Engineering for access:

```
git clone https://production.eng.it/gitlab/MORPHEMIC/markos-modified.git
cd markos-modified/web-crawler/api/
python setup.py develop
```

– Setup additional python dependencies:
– RDFAlchemy:

```
cd /opt/markos/markos-modified/
git clone https://github.com/gjhiggins/RDFAlchemy
cd RDFAlchemy/
```

```
pip install isodate
pip install pyparsing
python setup.py install
```

– DOAPFiend:

```
cd /opt/markos/markos-modified/web-crawler/lib/doapfiend/doapfiend-0.3.3/
python setup.py install
```

– Pyramid:

```
pip install waitress==1.2.1
pip install cornice==3.5.0
pip install venusian==1.2.0
pip install pyramid
```

– MySQL-python:

```
pip install MySQL-python
```

- Create a MariaDB database for the Web Crawler:

```
mysql -u root -p < /opt/markos/markos-modified/web-crawler/database/createCrawlerDB.sql
```

- Create a MariaDB user for the Web Crawler to connect to the database:

```
mysql -u root -p
 - CREATE USER 'markos'@'localhost' IDENTIFIED BY '******';
 - GRANT ALL ON markosN2_db2.* to 'markos'@'localhost';
 - FLUSH PRIVILEGES;
```

Populate additional database tables:

```
mysql -u markos -p -D markosN2_db2 < flossmole_apache.sql
mysql -u markos -p -D markosN2_db2 < flossmole_eclipse.sql
```

– Create additional paths required by the Web Crawler:

```
mkdir -p /opt/markos/markos-modified/data/data-for-doap-work /opt/markos/markos-modified/data/flossmole/gh
/opt/markos/markos-modified/web-crawler/logs/ /opt/markos/markos-modified/web-crawler/www
```

Once the procedure has been fully accomplished the Web Crawler is installed. The last step required in order to run the crawler is to configure it, according to the user needs, by editing the following file:

```
/opt/markos/markos-modified/web-crawler/config
```

For more information on the topic of Web Crawler configuration please refer to deliverable *D3.1 Software, tools, and repositories for code mining* [1]

3.4 Downloader Deployment

MORPHEMIC, as a multi management cloud system, provides two types of adaptation: (1) the adaptation of the resource allocated to an application (scaling operation), (2) modification of the architecture of the managed application. By architecture modification we mean the migration amongst application variants. The latter is a composition of an application form (VM, Docker container, Serverless, HPC) and the execution unit (CPU, GPU, FPGA). In order to enable the second type of adaptation application's profile must be created where all different variants will be discovered. Using the camel, the application's owner can define some variants according to its knowledge. However, it is important to establish a semi-automatic mechanism for discovering other possible variants so that the application described in the camel model can present a satisfying result in terms of performance and cost. The MORPHEMIC approach regarding the architecture adaptation consists of using open repositories for grouping application code according to their form and execution hardware. An important task for enabling this functionality consists of downloading a huge number of

repositories for creating component's clusters. This task is being executed by the Downloader, one of the main components of the Profiler.
The Downloader reads a local database (SQLite) containing the repository's information, then initiates the download process. For improving the overall download speed, many repositories can be downloaded simultaneously. The current version includes a basic analyser which extracts features from source code for being stored in a local database (MongoDB). After the analysis, the repository folder is deleted. MongoDB is deployed as a docker container component. Please refer to the following Table 5 Downloader Hardware Requirements and Table 6 Downloader Software Requirements for more insight on the Downloader requirements:

Table 5 Downloader Hardware Requirements

Hardware Requirements	CPU	RAM	Storage
Minimal	4 CPU core	8 GB of RAM	100GB of storage space
Recommended	8CPU cores	16 GB of RAM	200 GB

Table 6 Downloader Software Requirements

Operative System	Development Environment	Database
Ubuntu 18.04	– Python 3.6 – Python Pip – Docker – Docker compose – NLTK – GIT	MongoDB, SQLite

Downloader installation procedure:

- python installation

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3 python3-pip
python3 -m install gitpython nltk pydantic pymongo
```

- docker environment installation

```
sudo apt-get update
sudo apt-get install  apt-transport-https ca-certificates curl gnupg lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
sudo curl -L https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

- Docker MTU configuration

```
Edit the docker.service system file /etc/systemd/system/multi-user.target.wants/docker.service
ExecStart=/usr/bin/dockerd -H fd:// --mtu=1400 --containerd=/run/containerd/containerd.sock
systemctl daemon-reload
systemctl enable –now docker
```

4 Testbed Validation

The scope of this chapter is to document the activities that went into verifying that the MELODIC instances and MORPHEMIC in-development components that have been deployed within the testbed lead node are actually functioning as expected. The next paragraphs report on the activities performed after deploying the three MELODIC instances that are available for use-case validation, the first testing steps taken by our use-case partners and the first test run of the Web Crawler and Downloader development instances. While these validation actions do not represent a formal testing assessment of each instance involved, they can be considered "good enough" in order to ascertain that the core functionalities of the deployed services are effectively working as intended.

4.1 MELODIC deployment Validation

Once every MELODIC instance has been deployed on the testbed, a test run has been performed to ensure that the platform was capable of deploying applications using a CAMEL model. This step reproduces the kind of operations that MORPHEMIC partners execute on the testbed to validate their use-case models.
The following procedure is reported here the way it has been executed on one of the MELODIC instances, but the same procedure has been successfully repeated on each instance. In this example the testbed instance CHUV-MORPHEMIC will be used, which has public IP address of 217.172.12.245.
In order to execute this test run, a specific CAMEL model, *TwoComponents_UiO.xmi*[19], provided by 7Bulls, has been used. This model deploys an application with two components, one MariaDB instance and one application that connects to said database instance. As common with CAMEL, the provided model provides a VM image to instantiate, a number of application components to deploy and several constraints that the system needs to take into account, making it a useful all-around test application.
As a first step, the user validating this deployment has logged to the CHUV-MORPHEMIC instance in order to obtain a token from the MELODIC backend service by connecting to https://217.172.12.245:8078. Once the token has been successfully obtained, and the user has logged in on the MELODIC UI at https://217.172.12.245, the test file has been submitted through the deployment tab by following the wizard, after which the execution of the test application has been requested. After the MELODIC instance went through all the deployment stages and the application up and running, a connection to the instantiated VMs has been established in order to verify that the dummy executable was running and connecting to the MariaDB instance. The test application was found successfully executing as intended, thus the instance was confirmed as working properly.
Since a few issues were encountered before MELODIC could successfully deploy applications when hosted within the testbed lead node, some troubleshooting steps that helped identifying and correcting such issues have been documented in Appendix A, Troubleshooting MELODIC.

4.2 Web Crawler Deployment Validation

This paragraph details how the Web Crawler development instance has been validated after the initial deployment in order to ensure that the software is working as expected. Coincidentally, the reader will also be exposed to a few basic operations with the Web Crawler. Should any other question about the Web Crawler arise, the reader is advised to refer to the deliverable *D3.1 Software, tools, and repositories for code mining* [1] for more information on this subject.

In order to verify the correct installation of the Web Crawler it is advisable to execute at least one test run with at least one supported forge, among GitHub[20], Sourceforge[21] and others, enabled. To this end, the user validating the instance needed to login to the relative VM, specify which remote repositories the crawler should scrape for metadata, start the crawler and wait for it to do one full run. The Web Crawler instance was capable of downloading and parsing content from remote repositories and generating the description of a project (DoaP) entries relative to the scraped projects. In short, a DoaP is a type of object that contains information about a project. DoaPs are generated by the Web Crawler at the end of a run by leveraging the information collected from remote project repositories, which is scraped and then

[19] The test application XMI file can be downloaded here
[20] https://github.com/
[21] https://sourceforge.net/

merged. If, after a run, the Web Crawler generates any number of DoaPs, it is safe to assume that the deployment is working as expected.

It should be noted that by design, once started, the Web Crawler will always keep running unless explicitly stopped. Since the scope of this paragraph is to verify that a Web Crawler instance has been properly deployed, only one full run has been executed, followed by the issuing of a command to halt the crawler processes. Furthermore, while the following procedure does not differ if executed after multiple runs, this test run was the first execution of the Web Crawler after its deployment.

In order to test the Web Crawler deployment, the following steps were executed:

- Logging in to the Web Crawler VM, switching to the markos user and activating the Python VirtualEnv:

```
su markos
cd $HOME
source markos-virtualenv/bin/activate
```

- Connecting to the Web Crawler database using the password of the Crawler's DB user:

```
mysql -u markos -p -D markosN2_db2
```

- Quering the DoaP table for the overall number of generated DoaPs:

```
mysql -u markos -p -d markosN2_db2
```

Note: since the Web Crawler has not run before, the result looked like this:

```
+----------+
| count(*) |
+----------+
|   0      |
```

-

- Exiting the MariaDB session, starting the Web Crawler for the first time and letting it run, at this point it was noted that no errors were displayed at any time:

```
cd /opt/markos/markos-modified/web-crawler/
./startCrawler |tee logs/crawler.log
```

After letting the Web Crawler run, it queried and downloaded data from the configured remote repositories, extracted and parsed the contained metadata and finally merged this data and generated the relative DoaP entries. Depending on the configured time range and the type and number of sources, the amount of time required for a whole run to be completed could go from less than an hour to up to any number of days. In order to understand when to halt the process, users should wait until the crawler logs the information that there are "*no batches left that need to be integrated*", as it has been done during this test run.

Once the first run had ended, DoaP entries were generated as expected. Since code repositories supported by the Web Crawler are renown services that host a significant number of projects, the result of the "count" query executed on the DoaP table was expected to provide any value starting from a few thousand records. For reference, the result obtained was:

```
select count(*) from Doap;
+----------+
| count(*) |
```

```
+----------+
|  38824 |
+----------+
```

Since during the test run no errors have been logged by the Web Crawler and DoaP entries have been correctly generated, it is safe to assume that the software is working as intended and that the deployment has been successful.

4.3 Downloader Deployment Validation

After having installed all dependencies required for the deployment, the Downloader process has been started. MongoDB, the database leveraged by the downloader, has been launched as a docker container using the docker-compose file defining its configuration.

The validation of docker installation has been performed by executing `docker-compose version` and `docker version`. The version was displayed without throwing any error message.

MongoDB has been executed by running sudo docker-compose up -d on the root folder of the Downloader project.
By executing `sudo docker ps -a`, the terminal displayed the MongoDB container in a "running" status.
The Downloader has then been started by executing `python3 src/app.py`.

As expected, the Downloader created a temporary folder at */tmp*, then begun downloading repositories. The progress and the extraction information have been printed in the terminal without any error.
The correctness of the process execution has then been confirmed by verifying the data stored by the Downloader inside its MongoDB database by executing:

```
sudo docker exec -it mongodb mongo
use classifier
db.tokens.find()
```

The terminal showed features from all projects which have been extracted as expected, thereby validating the deployment of the Downloader.

4.4 Backup Testbed Validation

This paragraph details the assessment of the Backup Testbed performed in order to confirm that everything is working as intended and to ascertain that the Backup Testbed is fully operational in the event of failure of the main Testbed facility. In order to validate the testbed, the guidelines for the testing methodology, the test cases and the testing environment as presented in D4.4 Test Strategy[11, Cpv. 3.3] (*Acceptance Environment)* and D4.5 Test cases and testing [9] have been followed.

More specifically, Table 7 Backup Testbed, list of test cases performed enumerates the test cases executed on the Backup Testbed, which constitute a subset of the test cases presented in D4.5. All the test cases enumerated in this table have been performed with positive results.

Table 7 Backup Testbed, list of test cases performed

KEY	Name	Priority	Group
MORPHTEST-64	1.5 Morphemic - Deployment of a two-component application on one Cloud Provider	Medium	Initial deployment
MORPHTEST-68	1.5 - T3.1 [T] Install Morphemic with Proactive	Medium	Initial deployment

KEY	Name	Priority	Group
MORPHTEST-73	1.5 - Performance model	Medium	Metric management
MORPHTEST-72	1.5 - Persistent storage	Medium	Metric management
MORPHTEST-74	1.5 - TEST COMMUNICATION with EMS REST endpoints	Medium	Metric management
MORPHTEST-75	1.5 - MetaSolver notifies EMS for a CP model update	Medium	Metric management
MORPHTEST-76	1.5. - Proactive notifies EMS for a new VM and COMMUNICATION with EMS ActiveMQ	Medium	Metric management
MORPHTEST-96	1.5 - T1.8 [F] Assure that wrong cloud configuration forbids retrieval of Node Candidates	Medium	Initial deployment
MORPHTEST-97	1.5 - T1.7 [T] Successfully retrieve Node Candidates for AWS	Medium	Initial deployment
MORPHTEST-99	1.5 - T1.5[T] Create VM on AWS resources	Medium	Initial deployment
MORPHTEST-58	T3.1 [P] Install MORPHEMIC with Proactive	Medium	Initial deployment
MORPHTEST-49	T2.1 [P] Testing the operation of Melodic testbed.	Medium	Initial deployment
MORPHTEST-45	UC-ICON-1 Deployment of HelloWorld Camel - Stomp.py to create worker	Medium	Reconfiguration
MORPHTEST-46	UC-ICON-2 Test ActiveMQ server client deployment	Medium	Initial deployment
MORPHTEST-47	UC-ICON-3 run different base images for scheduler and worker	Medium	Initial deployment
MORPHTEST-48	UC-ICON-4 check requirements storage option	Medium	Initial deployment
MORPHTEST-14	UC-ISW-T-01: CPU/RAM utilization metric acquisition testing	Medium	Metric management
MORPHTEST-9	T1.8 [N] Assure that wrong cloud configuration forbids retrieval of Node Candidates	Medium	Initial deployment
MORPHTEST-3	T1.2[P] Installation and deployment of two-component application on AWS	Medium	Initial deployment
MORPHTEST-8	T1.7 [P] Successfully retrieve Node Candidates for AWS	Medium	Initial deployment
MORPHTEST-6	T1.5[P] Create VM on AWS resources	Medium	Initial deployment

Once the formal assessment of the Backup Testbed has been completed, ICON ran their use-case leveraging the Backup Testbed MORPHEMIC instance hosted at NREC, which, in turn, has deployed the workload on their own AWS infrastructure as per provider configuration. Specifically, ICON tested a CFD use-case by:

- uploading the use-case Camel model
- deploying the application on ICON's own AWS infrastructure

- restarting the MORPHEMIC instance and redeploying the Camel model for a second run

At the beginning, the executed scenario included a scheduler and a single worker component, where an iconCFD Platform web frontend is deployed on the scheduler component in order to upload geometries and setup CFD simulation projects - which are then run on the deployed worker instances. The first part of the test was executed correctly, with the use-case deployed exactly as expected. Then, the reconfiguration of the deployment was tested by filling up the job queue. That was done in order to verify that MOPRHEMIC could automatically reconfigure the deployment in order to meet the given deadlines and, as expected, new workers were created filling up the job queues and, ultimately, all the given jobs were executed. The MOPRHEMIC instance was then restarted and the whole test repeated. Everything went according to the expectations and the test has been fully completed with success.

Therefore, the conclusions from the executed test were:

- the Backup Testbed MORPHEMIC instance was capable of successfully deploying the use-case on the selected provider
- the reconfiguration capabilities were working as intended and the resources were scaled appropriately given the workload

Figure 6 ICON Use-Case running on the Backup Testbed shows the iconCFD frontend tracking the state of the workers during the use-case execution.

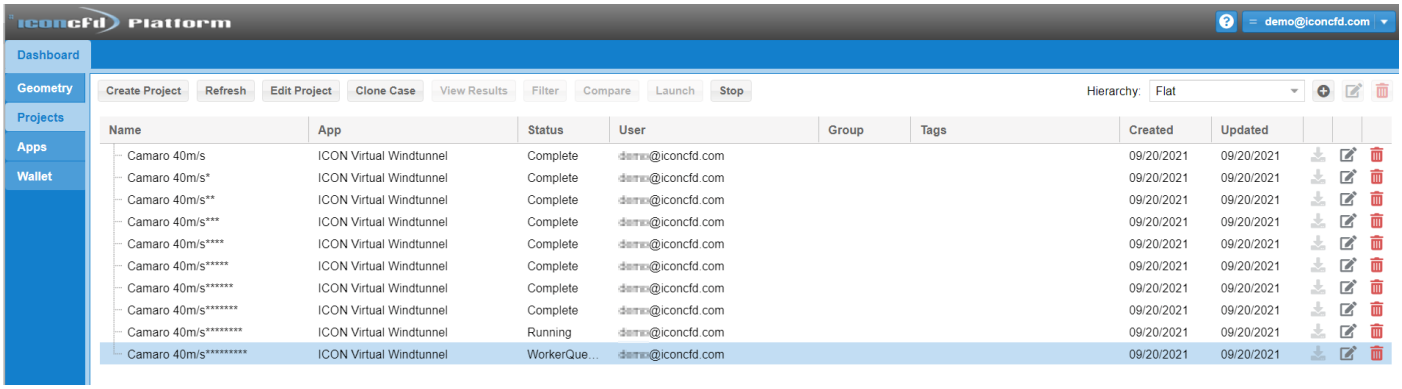

Figure 6 ICON Use-Case running on the Backup Testbed

4.5 Use-Case Deployment Validation

While at the time of writing use-cases partners are still working on having their use-cases working on MORPHEMIC, some partial testing was successfully performed by ICON and ISW by leveraging the testbed MELODIC instances. More specifically ICON was able to deploy, via their FiWare testbed instance, two ActiveMQ components - which are part of their use-case, having them instantiated by leveraging AWS as target provider. ISW deployed a fully operational component of their use case SD-RAN application (CU-CP component) on the AWS cloud provider as deployment node by means of their MELODIC instance deployed at MORPHEMIC testbed. The CU-CP component was able to connect to the 5G core network and conduct a Registration Procedure mimicking User Equipment connecting to the 5G system. However, despite the software being deployed as expected, it was not possible to access the VMs via WebSSH due to issues that are still under investigation. ISW used a workaround to this issue by allowing a password login in order to have remote, password based, SSH access to the instance. This approach allowed to verify the deployment correctness and to conduct the necessary tests.

5 Security Considerations

Every infrastructure is exposed to many different security risks, from denial of service to data leaking, malware, data ransom and other threats. Just as many are the ways for malevolent parties to exploit the weaknesses exposed by any infrastructure, penetrating its defences via non secured services that are publicly exposed, social engineering, lack of adherence to best practises or by leveraging a non-adequate software architecture. Taking the social factor aside, it is possible to define the grade of an infrastructure security as something that is closely related to the security level of the software deployed within.

In order to assess and confirm that the security levels required for an enterprise-grade environment were met when deploying the testbed, a security analysis has been performed on both the hosted software instances and their publicly exposed services. This work has been performed strictly on the software components of the MELODIC instances that have been deployed within the testbed, thus excluding the use-case applications that are delivered via the platform itself. Furthermore, this analysis does not concern the security of the Cloud providers that host the testbed or upon which the use-cases are executed; it is, therefore, limited in scope to the MELODIC software platform itself. The analysis also includes an assessment on the security of the development instance of the Web Crawler since, at the time, it was the only other future MORPHEMIC software component that was deployed on the testbed.

The criteria that have been followed for the analysis were the following:

- Has the software any architectural weakness that might render it exploitable?
- Does the software expose any exploitable weakness? Is it exposed publicly?
- Is there any sensible data stored in an unsafe manner/without any security measures?

The outcomes of this security analysis have been provided as input for other project-related security work which will be included in *D4.2 Security design and implementation* [2]. Hence, the data and considerations that have sparked from such analysis will be found there. Therefore, the reader is advised to refer to that deliverable should there be a need for more detailed information on this topic.

To summarise the outcome of the security analysis, several issues have been found when scrutinising the components that have been deployed within the infrastructure. These weaknesses range from the usage of deprecated libraries, to data leaking, to minor and medium tier exploitable risks, up to a non-optimal number of services that are published on the internet, thereby increasing the attack surface available to malevolent parties. These are some of the key security elements that were identified and require the attention of the consortium. Several measures have been proposed to mitigate the risks and increase the security level of the software. However, there were some concerns on the security of the FiWare Lab infrastructure while waiting for the measures included in deliverable *D4.2 Security design and implementation* [2] to be fully evaluated and implemented.

Software security may at times feel as something unsubstantial or hard to grasp in its most tangible essence; however, during the testbed lifetime, there has been some anecdotal evidence on how critical it is to secure services that are publicly exposed on the internet. In fact, on the fifth of December 2020, the Web Crawler VM has been compromised in what was most probably an automated attack by a botnet. It is important to note that the vulnerability that was used as a vector for the attack was not exposed by the Web Crawler itself, but rather from a misconfigured Tomcat instance that was deployed on the VM for development purposes. This Tomcat 8 installation was deployed with an administrative UI that was using an unsafe username/password combination to authenticate the instance administrator. This administrative UI was not deployed with the stock Tomcat 8 package; rather it was added at a later time to ease the developer's work. This led to the exploitation of the service and to malware gaining privileged access to the VM. The malevolent process then installed a cryptominer and started replicating itself and deploying other malware via the compromised Tomcat instance. This abnormal behaviour went unnoticed for around two days, when the malware was finally identified and the security team begun to deal with it with the appropriate response. Luckily, when doing a post-mortem analysis on the VM, no horizontal movement of the malware was identified, meaning that the rest of the VMs on the same subnet, including the MELODIC instances, went untouched. Once the threat had been thoroughly studied, the VM was destroyed and recreated in order to host another instance of the Web Crawler, this time without any misconfigured service.

This episode further confirms that, when taking into consideration the most appropriate ways to defend the testbed infrastructure, the most effective approach is to avoid to publicly expose any service that does not strictly need to be published in order for the platform to function properly, while also trying to minimise the overall number of possible entry points in order to reduce the surface that can be attacked by malevolent entities. It was then decided that, until the security of the software deployed on the lead node could be improved up to the point where only a minimal number of services are exposed as safely as possible, access to the testbed instances would be on-demand. This means that all the testbed services are shut down when the partners are not actively executing tests to further their use-case development and validation. To further clarify, at the time of writing the instances are available and being actively accessed by our partners as they are testing their use cases. It is expected that, with the first release of MORPHEMIC, the security improvements that will be implemented will justify a re-evaluation of the risks, which will also include another security assessment of the hosted instances. This will hopefully lead to a less restrictive approach to the availability of the testbed instances without increasing the security risks for the infrastructure.

6 Conclusions and next steps

In this deliverable the testing environment for MORPHEMIC has been described. In the first part, the testbed architecture is presented, defining the testbed from a logical perspective: focus has been given to the interactions happening between the infrastructures that compose the testbed and the methods that use-case partners, MORPHEMIC developers and testbed administrators have at their disposal to interact with the hosted instances and the testbed infrastructure. Then, the two Cloud infrastructures that compose the testbed lead node (FiWare) and default deployment target (NREC) have been briefly described, briefly summarising the technologies involved and the hardware resources at disposal of the project. Additionally, the backup lead node, which has been activated during late August 2021 due to a temporary issue with the FiWare node, has been presented as the NREC/AWS-based backup plan to host the testbed lead node while the infrastructure reparations are ongoing.

The testbed purpose is to test and validate the MORPHEMIC's components, the use-cases application components and the platform developments. Since the integration between the MORPHEMIC platform and the ProActive Scheduler is still undergoing development, a baseline upon which partners can begin testing their use-cases has been provided by deploying three instances of the MELODIC platform. Considering that MELODIC is at the hearth of MORPHEMIC, this solution has proven to be a solid choice while enabling the use-case partners to test their in-development use-cases, as explained in Chapter 4 Testbed Validation. Each platform instance has been tested and confirmed to be functional by deploying a test application under the supervision of 7Bulls and leveraging NREC as the deployment node. Moreover, two use-case partners have confirmed to be able to run at least a part or a less complex version of their use-cases.

Chapter 3 Testbed Setup offered some technical details, including the activities performed to set up the testbed and the new MORPHEMIC components that are in development and yet to be integrated. The actual process of configuring MELODIC on FiWare has been included in Appendix A, MELODIC SETUP. While the steps required to deploy MELODIC are not in the scope of this deliverable, a guideline on the issues encountered, the troubleshooting procedures and the solutions that have been adopted have been documented for future reference, especially since the technical challenges that have been solved might resurface even when future releases of MORPHEMIC will be deployed.

In Chapter 5 Security Considerations, some considerations regarding the security of the testbed and the instances hosted within have been presented, mentioning the security analysis performed on the aforementioned subjects. This activity, which has proven useful for the improvement of the security aspects of the software and the safety of the infrastructure, is briefly mentioned in order to provide the context within which the specific mitigation actions, basically the restricted instance availability policy, have been adopted - while the full coverage of the security analysis performed on the testbed will converge into D4.2 Security design and implementation [2], which will offer a more comprehensive approach to the evolution of the software security within MORPHEMIC.

In conclusion, while the setup of the testbed has proven to be challenging, mainly due to the act of performing a software deployment in a mixed research/enterprise environment, several useful lessons have been learned. The actual state of the testbed shows that all the main components are ready in order to proceed with the deployment, testing and validation of both the MORPHEMIC platform and its use-cases. In the meantime, use-cases development and validation is ongoing, with the first tests being executed on the testbed, while future updates are being planned according to

MORPHEMIC developments. This includes replacing each testbed MELODIC instance with MORPHEMIC release 1.0, thus enabling the execution, testing and validation of both the project use-cases and of MORPHEMIC itself in a scenario that is close to the real world. Once this update will be in place and with future developments of the Polymorphic Adaption, the testbed will gain a wider support for deployment nodes, thanks to the addition of the ProActive Scheduler which will further increase the number of supported Cloud providers and other technologies such as FPGAs, edge, fog and bare-metal.

7 References

[1] M. A. Di Girolamo, C. Formisano, M. Różańska, G. Horn, K. Krytikos, e A. Taherkordi, «D3.1 Software, tools, and repositories for code mining». dic. 31, 2020.
[2] C. Formisano, «D4.2 Security design and implementation». [due in Month 18].
[3] P. Skrzypek, M. Katarzyna, e M. Semczuk, «D4.3 Selection design and implementation of integration layer». giu. 30, 2021.
[4] C. Formisano, R. Gdowski, A. Latypova, F. Kherif, e S. Geller, «D6.1 Industrial requirements analysis». ago. 31, 2020.
[5] M. Różańska *et al.*, «D4.1 Architecture of pre-processor and proactive reconfiguration». mar. 19, 2021.
[6] M. D. Moghadam, «D3.3 Optimized planning and adaptation approach». apr. 30, 2021.
[7] F. Kherif, R. Gdowski, S. Geller, C. Formisano, e A. Latypova, «D6.3 Use cases definition and preparation». dic. 31, 2020.
[8] A. Raikos, «D6.2 Validation framework design». apr. 30, 2021.
[9] M. Byra, «D4.5 Test cases and testing».
[10] I. Matranga, «D8.2 Initial IPR and Exploitation Plan». giu. 30, 2021.
[11] A. Wyszomirska, «D4.4 Test Strategy». gen. 01, 2020.

8 Appendix A

8.1 MELODIC SETUP

While setting up the testbed Cloud providers, the configuration of NREC as the default Cloud provider proved to be challenging. This was partly due to a lack of documentation on the subject of adding an OpenStack-based Cloud provider to MELODIC and partly due to the presence of monitoring software deployed within the FiWare Lab network which was dropping legit traffic due to false positives. It is therefore deemed valuable to include the steps taken to configure NREC as the fallback deployment node Cloud provider as a reference on what has been done to bring the testbed to a working state. It is important to note that, when adding NREC or every other Cloud provider that is OpenStack-based, all the parameters required to proceed with the configuration are contained within the OpenStack RC file[22] that is either offered by the Cloud provider itself or downloadable once the user is authenticated to the OpenStack instance web UI, namely Horizon.
Once logged into the MELODIC instance that has to be configured, the user needs to access the "Provider Settings" tab and edit the openstack4j configuration. Figure 7 MELODIC OpenStack Provider Configuration is included as a reference of the configuration page.

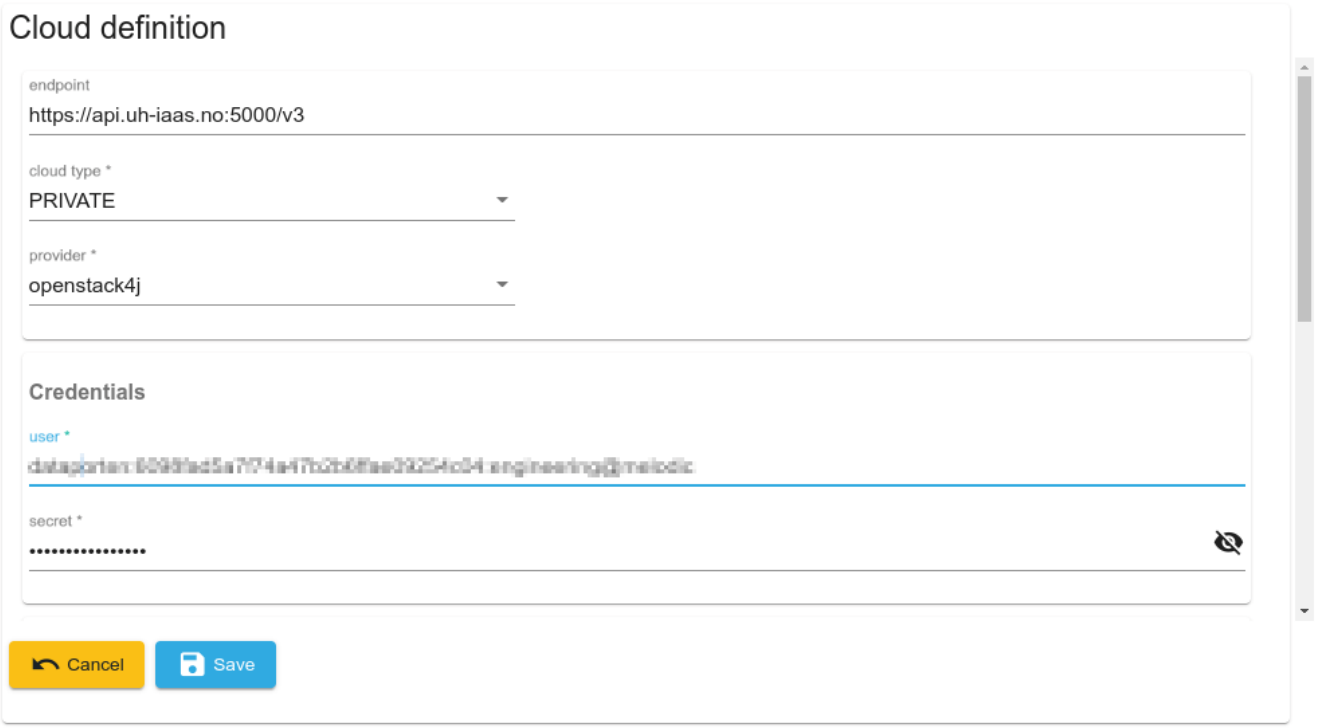

Figure 7 MELODIC OpenStack Provider Configuration

In order to setup an OpenStack-based Cloud provider, the configuration has to be accomplished by using these parameters:

- Endpoint: the OpenStack Keystone API public URL
- Cloud type: PRIVATE
- Provider: openstack4j
- Credentials: this string is divided in three parts separated by a colon symbol (":")
    - OS_PROJECT_DOMAIN_NAME = the human readable name of the OpenStack project
    - OS_TENANT_ID = the actual internal ID of the project
    - OS_USERNAME = the username of the account used to authenticate to the remote endpoint

[22] https://docs.openstack.org/zh_CN/user-guide/common/cli-set-environment-variables-using-openstack-rc.html

- Secret: the password of the account that is accessing the remote endpoint
- Node group: a name for the group inside which the VMs will be instantiated
- Cloud properties: here the user can specify several openstack4j properties and their respective values. In this case, the following parameters where used:
    - Sword.openstack4j.defaultNetwork= the id of the default OpenStack VM network to leverage
    - Sword.providerId.blacklist= an exclude list that specifies OpenStack entities that will be ignored when deploying a new application

Once the user hits the Save button the Cloud provider configuration is done. This means that the MELODIC instance is finally ready to deploy.

8.2 Troubleshooting MELODIC

This section contains some of the most meaningful issues and their relative fixes that were found during the deployment of the MELODIC instances that were instantiated for our use-case partners. These have been included as future reference.

8.2.1 Network Issues/Grafana not starting

One notable thing when deploying MELODIC on the testbed is that FiWare Lab nodes have a custom MTU setting specified on their network interfaces. This setting is slightly below the typical MTU value, namely 1450 instead of 1500. This particular network setting has caused several erratic network behaviours when MELODIC has been deployed on the lead node due to a non-optimal interaction between Docker network bridges, upon which Docker containers that encapsulate MELODIC services communicate, and the virtualised network interface of the hosting VM. This behaviour usually manifested in randomly failing network connections when web traffic using SSL encrypted protocols such as HTTPS was involved, causing various issues including the failure of MELODIC's Grafana service deployment. In order to avoid said issues it is necessary to configure Docker to run at a lower MTU than the value specified on the network interface of the hosting VM, upon which Docker will setup its network bridges.
The way this has been achieved on the testbed is by going through the following procedure:

- login on the VM as root
- install Docker: `apt install docker`
- edit the Docker systemd service file: `/etc/systemd/system/multi-user.target.wants/docker.service`
- replace the ExecStart entry as follows: `ExecStart=/usr/bin/dockerd -H fd:// --mtu=1400 --containerd=/run/containerd/containerd.sock`
- Have systemd reload the service file and, finally, enable Docker:
    - `systemctl daemon-reload`
    - `systemctl enable –now docker`
- Deploy or re-deploy MELODIC

8.2.2 Spark failing to start/crashing at runtime

Another issue that was detected when deploying MELODIC on the testbed was a failing Spark instance. While Spark is planned to be removed by the time MORPHEMIC will reach the first release, at the time of writing it is still being deployed as part of MELODIC. It is therefore useful to note that, unless the system Linux kernel is configured with a high vm.max_map_count, Spark will possibly fail either at start-up or at runtime.
In order to avoid this issue, it is necessary to setup the system as follows:

- edit the sysctl.conf file: `vim /etc/sysctl.conf`
- Either replace or insert the following vm.max_map_count value: `vm.max_map_count=262144`
- Have the system reload the newly specified value: `sysctl –p`
- Restart MELODIC

8.2.3 VM Storage exhausting over time

Another issue that was experienced when setting up the testbed was caused by MELODIC-related Docker volumes growing in size and filling up the VM filesystem. This happens only at runtime. Upon inspection the issue seems to be caused by one of the Cloudiator components and it seems to commonly manifest when there are repeated discovery-

agent errors. While this issue seems should be fixed once Cloudiator will be removed in favour of ProActive, at the time of writing MELODIC is still deployed with this component as a mean to offer multi-Cloud support.
In order to work around this issue, an administrator user should issue the following command exclusively when MELODIC is running, doing otherwise would break the MELODIC deployment and require a redeployment:

```
sudo docker volumes prune
```

8.2.4 Errors during the "Fetching Offers" phase

During the initial setup of the testbed environment, several issues were causing application deployments through MELODIC to get stuck at the "Fetching Offers" phase. It has been discovered that this issue was related to the networking environment, both due to the interaction with docker (see "Network Issues/Grafana not starting" above) and to security software deployed within the FiWare Lab infrastructure not allowing some specific traffic. It should also be noted that a host of other issues could manifest with the same symptoms, rendering the identification of the culprit rather difficult. It is worth noting that, whilst not providing a direct solution, a simple troubleshooting tip that could ease the identification phase of the issue is to follow the discovery agent container log by accessing the VM and executing the following command:

```
sudo docker tail -f docker_discovery-agent_1
```