Adaptive edge/cloud compute and network continuum over a heterogeneous sparse edge infrastructure to support nextgen applications

Deliverable D6.3

# ACCORDION System Implementation (I)

ACCORDION receives funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 871793

# DOCUMENT INFORMATION

| PROJECT | |
|---|---|
| PROJECT ACRONYM | ACCORDION |
| PROJECT FULL NAME | Adaptive edge/cloud compute and network continuum over a heterogeneous sparse edge infrastructure to support nextgen applications |
| STARTING DATE | 01/01/2020 (36 months) |
| ENDING DATE | 31/12/2022 |
| PROJECT WEBSITE | http://www.accordion-project.eu/ |
| TOPIC | ICT-15-2019-2020 Cloud Computing |
| GRANT AGREEMENT N. | 871793 |
| COORDINATOR | CNR |
| DELIVERABLE INFORMATION | |
| WORKPACKAGE N. \| TITLE | WP6 \| Integration, Pilot Implementation & Evaluation |
| WORKPACKAGE LEADER | OVR |
| DELIVERABLE N. \| TITLE | D6.3 \| ACCORDION System Implementation (I) |
| EDITOR | Emanuele Carlini (CNR) |
| CONTRIBUTOR(S) | Emanuele Carlini (CNR), Alain Vailati (HPE) Jakub Rola (BS), Luca Ferrucci (CNR), Hanna Kavalionak (CNR), Konstantinos Tserpes (HUA), Ferran Diego (TID) |
| REVIEWER | Maria Pateraki (OVR) |
| CONTRACTUAL DELIVERY DATE | 30/04/21 |
| ACTUAL DELIVERY DATE | 30/04/21 |
| VERSION | 1.0 |
| TYPE | Demonstrator |
| DISSEMINATION LEVEL | Public |
| TOTAL N. PAGES | 26 |
| KEYWORDS | |

# EXECUTIVE SUMMARY

This document covers the implementation activities put in place in the ACCORDION consortium to enable a smooth integration of the distributed ACCORDION platform. The first part of the document describes the three core components of the platform, while the second part describes the first integration testing performed on such components.

This document has been prepared during an intensive phase of iteration in the installation and implementation procedure of the components. Further improvements are planned for a comprehensive evaluation of Use Cases that is due at M18.

# DISCLAIMER

ACCORDION (871793) is a H2020 ICT project funded by the European Commission.

ACCORDION establishes an opportunistic approach in bringing together edge resource/infrastructures (public clouds, on-premise infrastructures, telco resources, even end-devices) in pools defined in terms of latency, that can support NextGen application requirements. To mitigate the expectation that these pools will be “sparse”, providing low availability guarantees, ACCORDION will intelligently orchestrate the compute & network continuum formed between edge and public clouds, using the latter as a capacitor. Deployment decisions will be taken also based on privacy, security, cost, time and resource type criteria.

# REVISION HISTORY LOG

| VERSION No. | DATE | AUTHOR(S) | SUMMARY OF CHANGES |
|---|---|---|---|
| 0.1 | 15/03/2021 | Emanuele Carlini (CNR) | Table of Contents |
| 0.2 | 07/04/2021 | Emanuele Carlini (CNR)<br>Hanna Kavalionak (CNR)<br>Alain Vailati (HPE)<br>Jakub Rola (BS) | Added content for Sections 2, 3, and 5 |
| 0.3 | 11/04/2021 | Luca Ferrucci (CNR) | Added content for Integration Testing (Sect 6.) |
| 0.4 | 13/04/2021 | Konstantinos Tserpes (HUA) | Added content for the HUA resources |
| 0.5 | 15/04/2021 | Ferran Diego (TID) | Added the installation of the orchestrators |
| 1.0 (final) | 28/04/2021 | Emanuele Carlini (CNR) | Finalization |

# GLOSSARY

| | |
|---|---|
| EU | European Union |
| EC | European Commission |
| H2020 | Horizon 2020 EU Framework Programme for Research and Innovation |
| K3S | Lightweight Kubernetes |
| CLI | Command Line Interface |
| GUI | Graphical User Interface |
| QoE | Quality of Experience |
| ITM | Intermediate TOSCA Model |
| VIM | Virtual Infrastructure Manager |
| Minicloud | A single instance of the Edge Infrastructure pool framework |
| VM | Virtual Machine |

# TABLE OF CONTENTS

# 1 Relevance to ACCORDION

## 1.1 Purpose of this document

This document illustrates the status of the implementation and integration of the whole ACCORDION platform. To this end, the document contains information on the installation of the macro components of the platform, as well as the initial testing procedures that have been carried out. This deliverable does not enter in the details of the installation for each single software module of the platform, as those have been covered in many details in the following deliverables:

- D3.2 - Edge infrastructure pool framework implementation (I)
- D4.2 - Edge/Cloud continuum management framework implementation (I)
- D5.2 - Application management framework implementation (I)

## 1.2 Relevance to project objectives

This deliverable describes the efforts and the actions performed by the consortium to put together all the pieces of software that have been implemented in the first 16 months of the project. Therefore, its relevance regards all technical objectives of the project, namely:

- Obj1: Maximize edge resource pool
- Obj2: Maximize robustness of cloud/edge compute continuum
- Obj3: Minimize overheads in migrating applications to cloud/edge federations
- Obj4: Realize NextGen applications

## 1.3 Relation to other workpackages

This deliverable focus on the implementation and integration of the platform. Therefore, the content of the document mostly reflects the work done in the technical work packages, namely WP3, WP4, and WP5. The deliverable has an impact on the work package of the Use Case (WP6 - Integration, Pilot Implementation & Evaluation), that are using the ACCORDION platform presented here to test and evaluate the pilot prototypes.

## 1.4 Structure of the document

The document is structured as follows. Section 2 presents the methodology of implementation and integration, including a broad platform overview, the integration actions, and the testing procedure that has been put in place to test this initial version of the system. Section 3, 4, and 5 describe the installation of the three main macro-components of the ACCORDION Platform. Section 6 discusses in details the testing procedure of the integrated platform. Finally, Section 7 concludes the document.

# 2 Introduction

## 2.1 Platform Overview

The first release of the system implementation of ACCORDION is a basic platform resulting from the composition of the work done in the past months by the technical WPs. The platform is composed of three main core components, each identifying a related set of services to support the ACCORDION vision. At the current status, the components offer a basic set of functionalities which is planned to build upon in the next iterations of the project. For the full list of current supported functionalities, please refer to the following deliverables D3.1, D4.1, and D5.1.

Each macro-component has been installed on different hardware in different geographical positions. Following this installation, a testing procedure has been executed to verify the installation and provide a first glance on running the NextGen applications on ACCORDION. This work lays the basis for the platform evaluation that is due to M18 with deliverable D6.5.

The core macro-components of the ACCORDION platforms are the following:

- **Application Management Framework**. Encompasses the work done in WP5, and it contains those methodologies and procedures to assist application developers working with ACCORDION. For this deliverable the most notable function of this macro-component is the hosting of application images and application models.
- **Edge/Cloud Continuum Management Framework.** The decision-making component of the platform. This component contains the software artifacts that resulted from WP4. In the current versions, the most prominent service in this macro-component is the Orchestrators, which takes the decision of where to deploy applications.
- **Edge Infrastructure Pool Framework**. This macro-component manages the pool of available edge resources via a series of dedicated services (developed within WP3) that run on the edge. For this deliverable, the core service is the virtual infrastructure manager, which manages the deployment and orchestration of applications at the edge. A single instance of the Edge Infrastructure pool framework is also referred to as *Minicloud*.

The next sections describe in detail the installation procedure of each component, as well as the computational and network resources on which they have been installed.

## 2.2 Integration actions

The integration of the ACCORDION platform has required a coordinated effort among all partners. To support such effort, the consortium has taken a series of actions that have been taken to assure a smooth integration process.

From an organizational standpoint, a weekly consortium call has been performed since M13 to specifically discuss integration related matters. Relevant operations made during these calls have been the definition of an integration testing strategy (described in Section 6), as well as the definition and monitoring of the technical action to reach the targeted integration. These actions include:

**Implementation of Use Cases.** The services that compose the Use Case applications have been implemented. UC#2 and UC#3 realized their application by using docker containerization. UC#1 used a mix of docker container and windows VM.

**Definition of application models of Use Cases.** This action has required the generation of two documents for the description of an application for each Use Case. The first is a TOSCA-like document that describes the application in general. The second is a YAML document that describe the deployment of the application in the edge/cloud continuum.

**Developments of components and interfaces**. With respect to their definition in D3.1, D4.1, and D5.1 several components have been improved and functionalities have been added. These advancements have been due to a refinement of the interfaces and the management of the flow of information.

**Installation of components.** All the relevant components have been installed into a collection of distributed computational resources. This has required all components to refine the installation procedures written in D3.2, D4.2, and D5.2, as well as a few “hands-on" meetings to debug and provide feedback.

# 3 Edge Infrastructure Pool Framework

## 3.1 Installation procedure

This section describes the procedure that has been followed to install the Edge Infrastructure Pool Framework. The test environment is composed by a single k3S node, which take both the role of master and worker. The ACCORION services (namely: monitoring, edge storage and resources indexing) have been installed on top of such K3S installation.

The installation procedure below assumes a Linux machine and the possession of the gitlab.com access credentials.

The first installation step to verify and install some dependencies with the following command:

```
sudo apt install git curl python3-pip ufw x11-xserver-utils
```

After that it is possible to install all the modules of the Edge Infrastructure Pool. The following sections briefly describe the installations procedures of all the services involved.

### 3.1.1 VIM (Virtual Infrastructure Manager)

To start installation, it is necessary to download installation scripts, all installation scripts are versioned on gitlab.com (see D5.2 for more details).

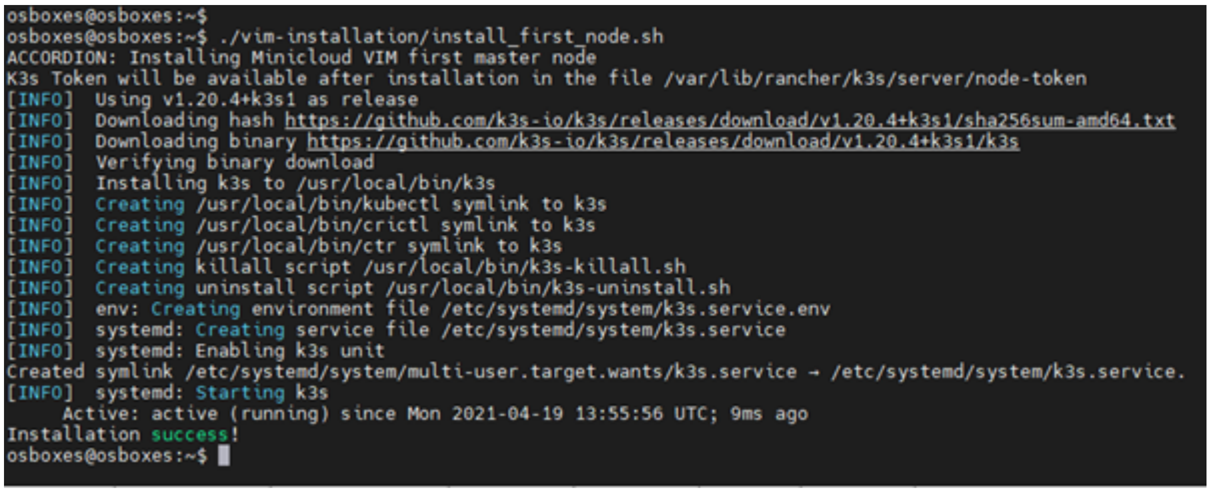
```
$git clone https://gitlab.com/accordion-project/wp3/vim-installation
```

Then, change the execution permission to the installation scripts:

```
$ chmod 775 vim-installation/*.sh
```

The script executed to install K3S master node is install_first_node.sh

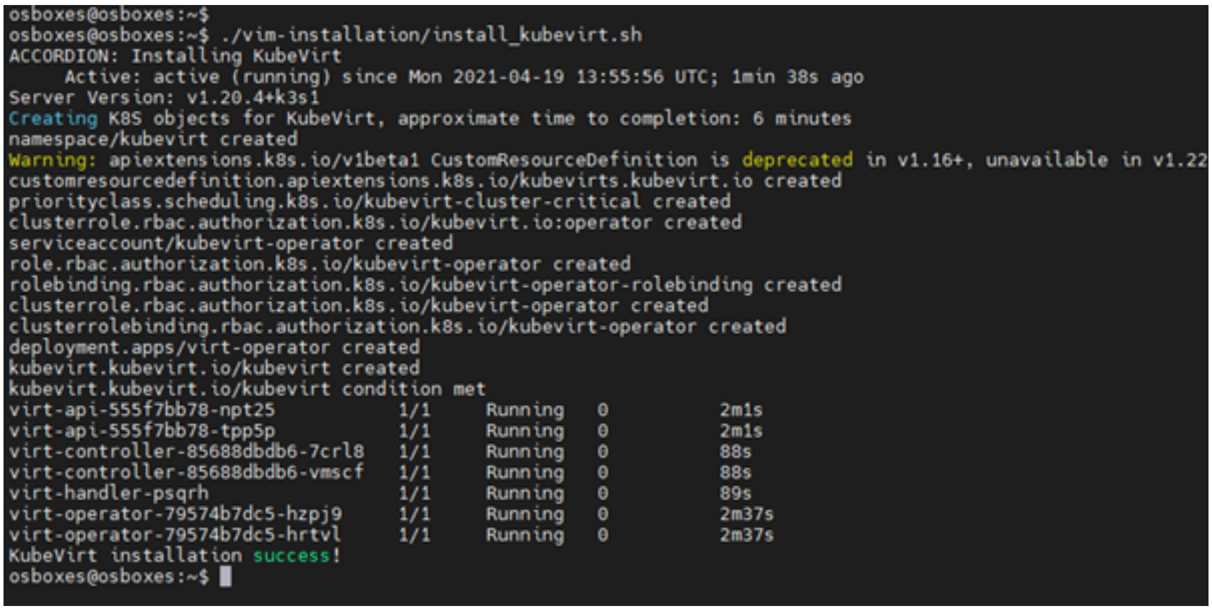
```
$./vim-installation/install_first_node.sh
```



```
osboxes@osboxes:~$
osboxes@osboxes:~$ ./vim-installation/install_first_node.sh
ACCORDION: Installing Minicloud VIM first master node
K3s Token will be available after installation in the file /var/lib/rancher/k3s/server/node-token
[INFO]  Using v1.20.4+k3s1 as release
[INFO]  Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.20.4+k3s1/sha256sum-amd64.txt
[INFO]  Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.20.4+k3s1/k3s
[INFO]  Verifying binary download
[INFO]  Installing k3s to /usr/local/bin/k3s
[INFO]  Creating /usr/local/bin/kubectl symlink to k3s
[INFO]  Creating /usr/local/bin/crictl symlink to k3s
[INFO]  Creating /usr/local/bin/ctr symlink to k3s
[INFO]  Creating killall script /usr/local/bin/k3s-killall.sh
[INFO]  Creating uninstall script /usr/local/bin/k3s-uninstall.sh
[INFO]  env: Creating environment file /etc/systemd/system/k3s.service.env
[INFO]  systemd: Creating service file /etc/systemd/system/k3s.service
[INFO]  systemd: Enabling k3s unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s.service → /etc/systemd/system/k3s.service.
[INFO]  systemd: Starting k3s
     Active: active (running) since Mon 2021-04-19 13:55:56 UTC; 9ms ago
Installation success!
osboxes@osboxes:~$
```

Now as K3S cluster is up and running, one can proceed to install on top of Kubernetes, Kubevirt the framework for virtual machine execution, executing from command line the installation script:

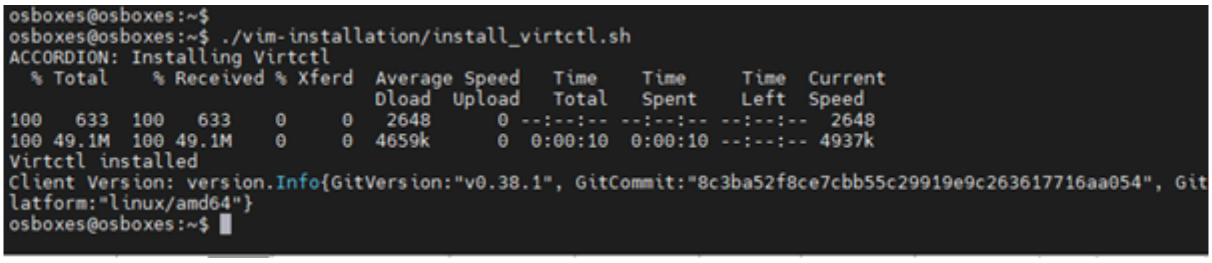
```
$./vim-installation/install_kubevirt.sh
```



```
osboxes@osboxes:~$
osboxes@osboxes:~$ ./vim-installation/install_kubevirt.sh
ACCORDION: Installing KubeVirt
     Active: active (running) since Mon 2021-04-19 13:55:56 UTC; 1min 38s ago
Server Version: v1.20.4+k3s1
Creating K8S objects for KubeVirt, approximate time to completion: 6 minutes
namespace/kubevirt created
Warning: apiextensions.k8s.io/v1beta1 CustomResourceDefinition is deprecated in v1.16+, unavailable in v1.22
customresourcedefinition.apiextensions.k8s.io/kubevirts.kubevirt.io created
priorityclass.scheduling.k8s.io/kubevirt-cluster-critical created
clusterrole.rbac.authorization.k8s.io/kubevirt.io:operator created
serviceaccount/kubevirt-operator created
role.rbac.authorization.k8s.io/kubevirt-operator created
rolebinding.rbac.authorization.k8s.io/kubevirt-operator-rolebinding created
clusterrole.rbac.authorization.k8s.io/kubevirt-operator created
clusterrolebinding.rbac.authorization.k8s.io/kubevirt-operator created
deployment.apps/virt-operator created
kubevirt.kubevirt.io/kubevirt created
kubevirt.kubevirt.io/kubevirt condition met
virt-api-555f7bb78-npt25            1/1     Running   0          2m1s
virt-api-555f7bb78-tpp5p            1/1     Running   0          2m1s
virt-controller-85688dbdb6-7crl8    1/1     Running   0          88s
virt-controller-85688dbdb6-vmscf    1/1     Running   0          88s
virt-handler-psqrh                  1/1     Running   0          89s
virt-operator-79574b7dc5-hzpj9      1/1     Running   0          2m37s
virt-operator-79574b7dc5-hrtvl      1/1     Running   0          2m37s
KubeVirt installation success!
osboxes@osboxes:~$
```

The last log line shows all Kubevirt pods running, and the installation has succeeded. In order to interact with virtual machine execution, install the CLI Virtctl with the relative script:

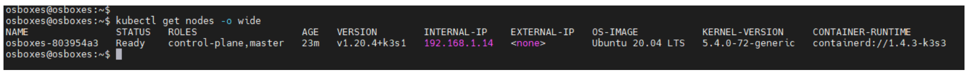
```
$ ./vim-installation/install_virtctl.sh
```



```
osboxes@osboxes:~$
osboxes@osboxes:~$ ./vim-installation/install_virtctl.sh
ACCORDION: Installing Virtctl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   633  100   633    0     0   2648      0 --:--:-- --:--:-- --:--:--  2648
100 49.1M  100 49.1M    0     0  4659k      0  0:00:10  0:00:10 --:--:-- 4937k
Virtctl installed
Client Version: version.Info{GitVersion:"v0.38.1", GitCommit:"8c3ba52f8ce7cbb55c29919e9c263617716aa054", Git
latform:"linux/amd64"}
osboxes@osboxes:~$
```

To check if it is working:

```
$ kubectl get nodes -o wide
```



```
osboxes@osboxes:~$
osboxes@osboxes:~$ kubectl get nodes -o wide
NAME               STATUS   ROLES                  AGE   VERSION        INTERNAL-IP    EXTERNAL-IP   OS-IMAGE           KERNEL-VERSION     CONTAINER-RUNTIME
osboxes-803954a3   Ready    control-plane,master   23m   v1.20.4+k3s1   192.168.1.14   <none>        Ubuntu 20.04 LTS   5.4.0-72-generic   containerd://1.4.3-k3s3
osboxes@osboxes:~$
```

### 3.1.2 Monitoring

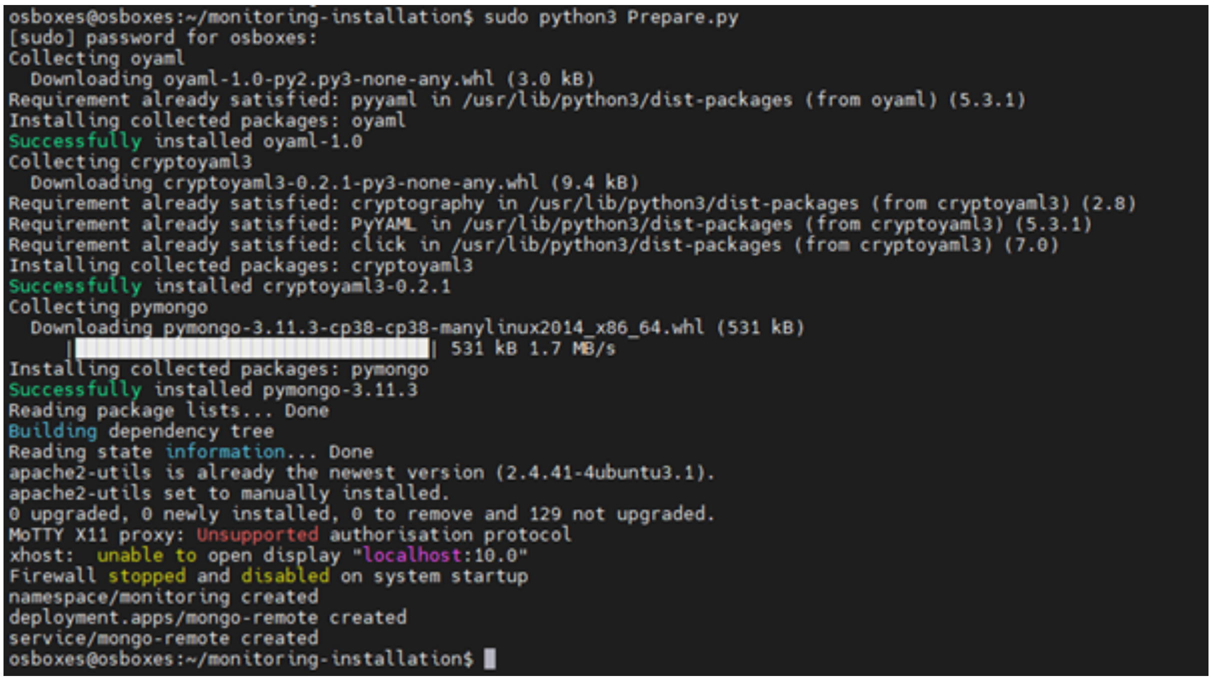
To start installation, is necessary to download installation scripts from Gitlab.

```
$ git clone https://gitlab.com/accordion-project/wp3/monitoring-installation
```

Then, from installation script folder that has been cloned, run the following script to prepare the environment:

```
$ cd monitoring-installation/

$ sudo python3 Prepare.py
```



```
osboxes@osboxes:~/monitoring-installation$ sudo python3 Prepare.py
[sudo] password for osboxes:
Collecting oyaml
  Downloading oyaml-1.0-py2.py3-none-any.whl (3.0 kB)
Requirement already satisfied: pyyaml in /usr/lib/python3/dist-packages (from oyaml) (5.3.1)
Installing collected packages: oyaml
Successfully installed oyaml-1.0
Collecting cryptoyaml3
  Downloading cryptoyaml3-0.2.1-py3-none-any.whl (9.4 kB)
Requirement already satisfied: cryptography in /usr/lib/python3/dist-packages (from cryptoyaml3) (2.8)
Requirement already satisfied: PyYAML in /usr/lib/python3/dist-packages (from cryptoyaml3) (5.3.1)
Requirement already satisfied: click in /usr/lib/python3/dist-packages (from cryptoyaml3) (7.0)
Installing collected packages: cryptoyaml3
Successfully installed cryptoyaml3-0.2.1
Collecting pymongo
  Downloading pymongo-3.11.3-cp38-cp38-manylinux2014_x86_64.whl (531 kB)
     |████████████████████████████████| 531 kB 1.7 MB/s
Installing collected packages: pymongo
Successfully installed pymongo-3.11.3
Reading package lists... Done
Building dependency tree
Reading state information... Done
apache2-utils is already the newest version (2.4.41-4ubuntu3.1).
apache2-utils set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 129 not upgraded.
MoTTY X11 proxy: Unsupported authorisation protocol
xhost:  unable to open display "localhost:10.0"
Firewall stopped and disabled on system startup
namespace/monitoring created
deployment.apps/mongo-remote created
service/mongo-remote created
osboxes@osboxes:~/monitoring-installation$
```

Then copy kubectl config file to script folder and execute Configs.py

```
$ cp -r ../.kube .

$ python3 Configs.py
```

### 3.1.3 ACES (ACCORDION Edge Storage Component)

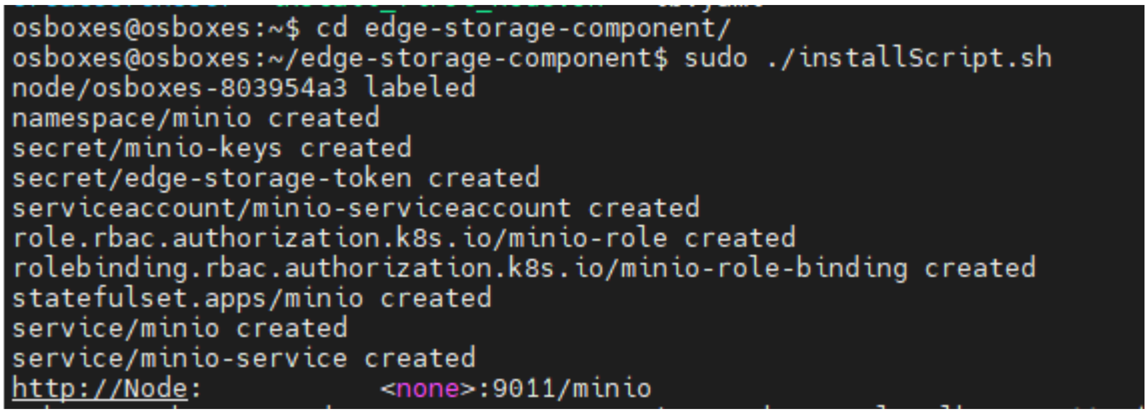
To install edge storage, first as usual, download installation scripts from Gitlab into home directory:

```
$ git clone https://gitlab.com/accordion-project/wp3/edge-storage-component
```

The script will be placed into folder edge-storage-component, move current directory to that give execution permission and start installation of installScript.sh script.

```
$ cd edge-storage-component/

$ chmod 775 installScript.sh
```



```
osboxes@osboxes:~$ cd edge-storage-component/
osboxes@osboxes:~/edge-storage-component$ sudo ./installScript.sh
node/osboxes-803954a3 labeled
namespace/minio created
secret/minio-keys created
secret/edge-storage-token created
serviceaccount/minio-serviceaccount created
role.rbac.authorization.k8s.io/minio-role created
rolebinding.rbac.authorization.k8s.io/minio-role-binding created
statefulset.apps/minio created
service/minio created
service/minio-service created
http://Node:           <none>:9011/minio
```

The execution of the “minio” service can be checked by opening on a browser on the localhost at this url: htpp://localhost:9011

### 3.1.4 RID (Resource Indexing and Discovery)

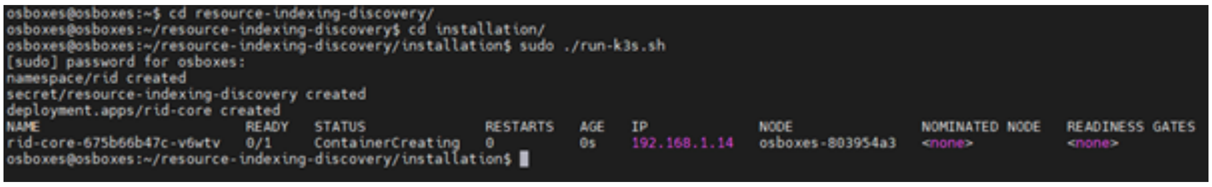
To install Resource Indexing and Discovery, download installation scripts from Gitlab into home directory:

```
$ git clone https://gitlab.com/accordion-project/wp3/resource-indexing-discovery
```

Change current directory and execute installation script run-k3s.sh script

```
$ cd resource-indexing-discovery/installation/

$ sudo ./run-k3s.sh
```



```
osboxes@osboxes:~$ cd resource-indexing-discovery/
osboxes@osboxes:~/resource-indexing-discovery$ cd installation/
osboxes@osboxes:~/resource-indexing-discovery/installation$ sudo ./run-k3s.sh
[sudo] password for osboxes:
namespace/rid created
secret/resource-indexing-discovery created
deployment.apps/rid-core created
NAME                        READY   STATUS              RESTARTS   AGE   IP             NODE                NOMINATED NODE   READINESS GATES
rid-core-675b66b47c-v6wtv   0/1     ContainerCreating   0          0s    192.168.1.14   osboxes-803954a3    <none>           <none>
osboxes@osboxes:~/resource-indexing-discovery/installation$ 
```

## 3.2 Available resources

For this first integration, the Edge Infrastructure Pool Framework has been installed on a single VirtualBox virtual machine, equipped with 16GB of RAM, 2 intel-9 CPUs, and 20GB of disk. From the perspective of Kubernetes, this machine works both as master and worker node. The machine is accessible via a public IP. The machine is under NAT with the host; therefore, for the VIM to be accessible from the Internet port of 6443 of the VIM has been mapped to the port 9443.

# 4 Edge/Cloud Continuum Management Framework

## 4.1 Installation procedure of the Dynamic Orchestrator

In this paragraph is described the procedure that has been followed to install the main component of the edge/cloud continuum Management Framework: the *dynamic orchestrator*. The installation procedure of the other components has been described in deliverable D4.2. The installation procedure below assumes a Linux machine and the possession of the gitlab.com access credentials.

The first installation step is to create a virtual environment and activate the environment:

```
$ cd conda create --name dynamic_orchestrator python=3.6

$ conda activate dynamic_orchestrator
```

To install dynamic orchestrator, first as usual, download installation scripts from Gitlab into home directory:

```
$ git clone https://gitlab.com/accordion-project/wp4/dynamic-orchestrator
```

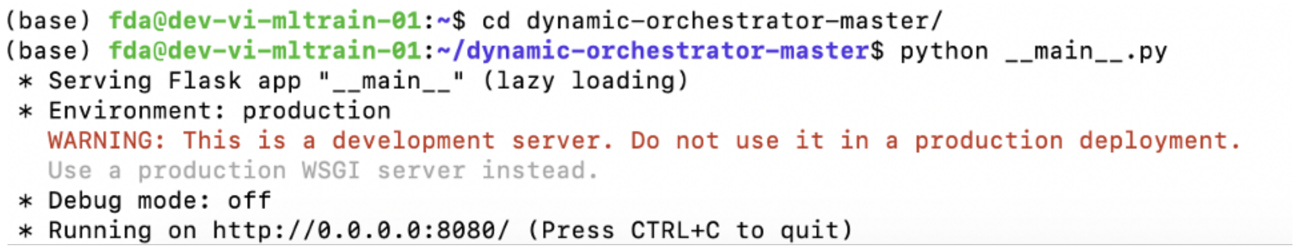
and then install some dependencies with the following command:

```
$ cd dynamic-orchestrator

$ pip install -r config/requirements.txt
```

To check if the installation is working:

```
$ python __main__.py
```



```
(base) fda@dev-vi-mltrain-01:~$ cd dynamic-orchestrator-master/
(base) fda@dev-vi-mltrain-01:~/dynamic-orchestrator-master$ python __main__.py
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

## 4.2 Available resources

For the purposes of the installation of the Management Framework, we employed 5 separate VMs in a private cloud hosted by HUA. The private cloud is featuring 1 modular system with 16 servers (blades), 2 optical switches, which connect the servers and the storage units together with FC. 1 server Oracle Sun Server T4 to serve multicore applications and other requirements. 1 unified storage Oracle Sun ZFS 7320 with a total capacity of 70TBytes.

The framework that manages the entire enterprise infrastructure is oVirt. An open-source distributed virtualization solution, that uses the KVM hypervisor. It was founded by Red Hat as a community project on which Red Hat Enterprise Virtualization is based.

Using oVirt, we reserved a pool of resources for ACCORDION each of which features the following characteristics:

| VM | IP | vCPUs | RAM (GBs) | Disk (GBs) | OS |
|---|---|---|---|---|---|
| ACCORDION-1 | 83.212.240.38 | 2 | 8 | 40 | Ubuntu 20.04.02 LTS |
| ACCORDION-2 | 83.212.240.46 | 2 | 8 | 40 | Ubuntu 20.04.02 LTS |
| ACCORDION-3 | 83.212.240.47 | 2 | 8 | 40 | Ubuntu 20.04.02 LTS |
| ACCORDION-4 | 83.212.240.48 | 2 | 8 | 40 | Ubuntu 20.04.02 LTS |
| ACCORDION-5 | 83.212.240.49 | 2 | 8 | 40 | Ubuntu 20.04.02 LTS |

**Table 1: VMs for the ACCORDION Management Framework**

Each VM is allocated to an Edge/Cloud Continuum Management Framework component, namely:

- **Resource Orchestrator:** which is embedded in a Docker container, which occupies about 50MB of RAM, a VCPU and about 300MB of storage. It needs a host based on Linux (or Windows with the WSL2.0 framework installed).
- **Network orchestrator:** which runs on a docker containers, we are using ubuntu 18.04 LTS 64bits, I believe 20.04 LTE should be ok. At this stage of the project, we only require 1 vCPU, an additional 1GB of RAM and 2 GB of storage.
- **Security component:** which runs on Docker containers (SonarQube + PostgreSQL DB + NGINX). The environment used for private tests is Ubuntu 20.04 LTS (VM) + docker v. 20.10.3 + docker-compose v. 1.28.2. RAM: 2GB RAM is required (and at least additional 1GB for the OS). Only 64-bit systems are supported. For the storage, about 2GB for the installation + the storage needed for the DB data.
- **Privacy component:** which needs a VM similar to the one above
- **Resilience component:** which needs a VM with at least 4 GBs RAM, 2 vCPUs and 20GBs storage. Ideally, the component requires a GPU too, in order to support online training scenarios but due to the lack of such infrastructure and in alignment with the component development plans, this will be delivered in the next iteration.

# 5 Application Management Framework

The detailed description of components that are developing in the scope of WP 5 is in the deliverable 5.1. The detailed information of the installation procedure and the user manual is available in the deliverable 5.2. Table 1 presents the summary of the two above mention deliverables.

| **Component** | **Short description** | **Technologies** | **Deployment** | **API/ exposing resource** |
|---|---|---|---|---|
| Application Model Artefact | Provide application models that use extended TOSCA grammar to describe application components, their actions and their relationships | not applicable | not applicable | D5.1 section 2 |
| Converter | Converter convert AMA TOSCA file to required k3s configuration files in yaml format | PYTHON 3 | It is the library used inside the Compute resource orchestrator | D5.2 section 2.1 |
| QoE validator | | PYTHON 2.7+ | It is the library used inside the Compute resource orchestrator | D5.2 section 3.1 |
| Application Bucket | Store provided by the ACCORDION user source code, application images, and deployment requirements | Spring Boot, MongoDB | It is deployed at the Bluesoft server http://82.214.143.119:31725 | The API is described with OpenAPI standard in yaml file[1] |
| Web-based GUI | Visual portal for the administrators and developers for | Angular, web technologies | In progress | In progress |

| | | | | |
|---|---|---|---|---|
| | interacting with the Accordion Platform | | | |
| Internal GitLab instance | It provides the repository functionality and act as resource and authorization server in OAuth 2 | Open source multi technologies component | It is deployed at the Bluesoft server http://82.214.143.119:31730/ | GitLab community documentation[2] |

Table 1. Summary of the services composing the Application Management Framework

# 6 Testing the Integration

This section describes the testing procedures to check the correctness and basic functionalities of the ACCORDION platform. Every test consists of executing the start application scenario of one of the three Use Cases of ACCORDION. The tests are not designed to stress the internal functionalities of the various components for this integration phase of the ACCORDION project. Such functionalities will be tested in the next phases of the development of the ACCORDION framework. Basically, the testing procedure consists in starting an application loaded into a repository into the edge, such that all the ACCORDION stacks is involved. The next sections provide a description of the testing environment as well as a detailed discussion of the actual testing procedure.

## 6.1 Testing Environment

The start application scenario is preceded by the deployment of the application scenario, which had been executed offline. Before the execution of tests, the Docker containers for the components of each Use Case had been stored and validated inside the Application Bucket (see Section 5). At the current stage, the platform only supports the fetching of docker images. The support to Virtual Machines is going to be added in the next releases of the platform.

Mirroring the macro-components described in Section 2.1, the ACCORDION Platform is then composed of three different logical locations:

- The *Application Management Framework,* which contains the Application Bucket service (mentioned above).
- Several public servers on which are deployed the *Orchestrator* components.
- A single *Minicloud*, on which is installed an instance for each Edge Infrastructure Pool Framework component. The Minicloud is composed of a single Kubernetes K3s cluster with a single node and is installed on a different host with respect to the Continuum Management Framework.

The testing procedure assumes every core component to be correctly configured, installed and their requirements fulfilled to obtain a ready-to-start service. The installation procedure of each component and the hardware and software resources on which they are deployed are described in sections 3, 4 and 5, respectively.

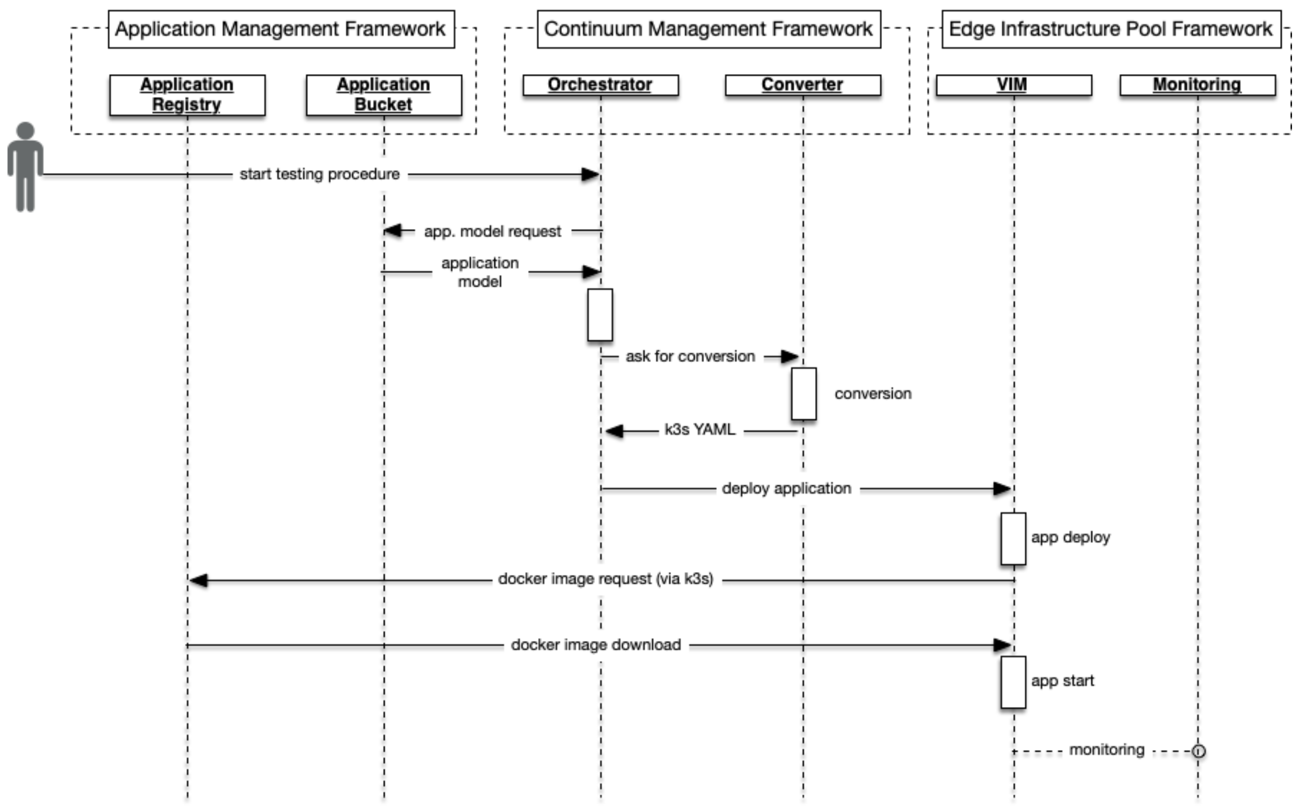


**Figure 1: Sequence diagram of the testing procedure**

## 6.2 Testing procedure

The testing procedure follows the sequence diagrams shown in Figure 1. The first step is to submit the start application request to the *Orchestrator* component. It is executed through the following POST operation over an extension of the HTTP Rest interface described n D4.2 section 2, at the following endpoint:

```
$ POST /orchestrator/appmodel/startapp?name=<UC_app_nome>
```

The query string parameter `name` must be one of the values in the set { ’ovr’, ‘orbk’, ’plexus’}, each referring to the application of the Use Cases 1, 2 and 3 of ACCORDION, respectively.

The returned json-formatted messages and codes are the following:

- *200 {'message': 'Application with the submitted name has been deployed successfully’}* when the application has been correctly deployed
- 500 {'message': <specific_error_text_message>}: the application has not been correctly deployed. Possible reasons could be:
    - Application image does not exist
    - The deployment on the VIM is failed due to a Kubernetes error
    - Connection error with the VIM or the Application Bucket

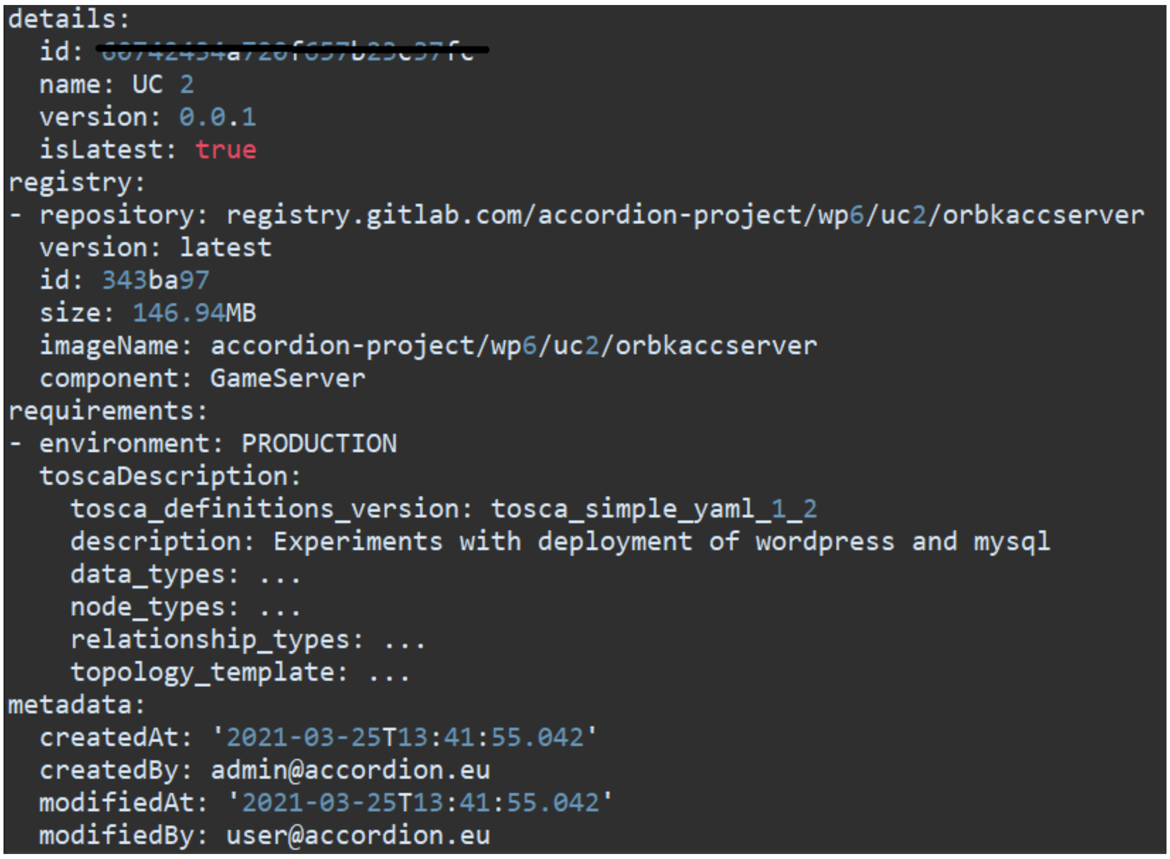
The *Orchestrator* has a table (hardcoded in this initial version) to map the app name into a private application identifier (`UC_app_id`), which is an alphanumeric string value that the Application Bucket uses to uniquely identify internally the application to be deployed. This identifier is submitted as a parameter for the following GET request to the Application Bucket to fetch the application model:

```
$ GET POST http://82.214.143.119:31725/application?id=<UC_app_id>
```

Alternatively, an alternative REST (which is currently under implementation) call indicates directly the application owner name, the scenario and a flag which is set to True to fetch the latest version of the component images of the Use Case; if the flag is set to False, a parameter to indicate the version must be specified in the call. Since the tests only perform the start application scenario, the parameter to specify it is not needed.

The returned json-formatted messages and codes are the following:

- `200`: The request is successful. In Figure 2 is shown the returned json-formatted messages for the 'orbk' Use Case, translated in Yaml.
- `404 {'Could not find the application model with id: <UC_app_id>'}`: The request failed. The application model for the requested ID does not exist in the application bucket.



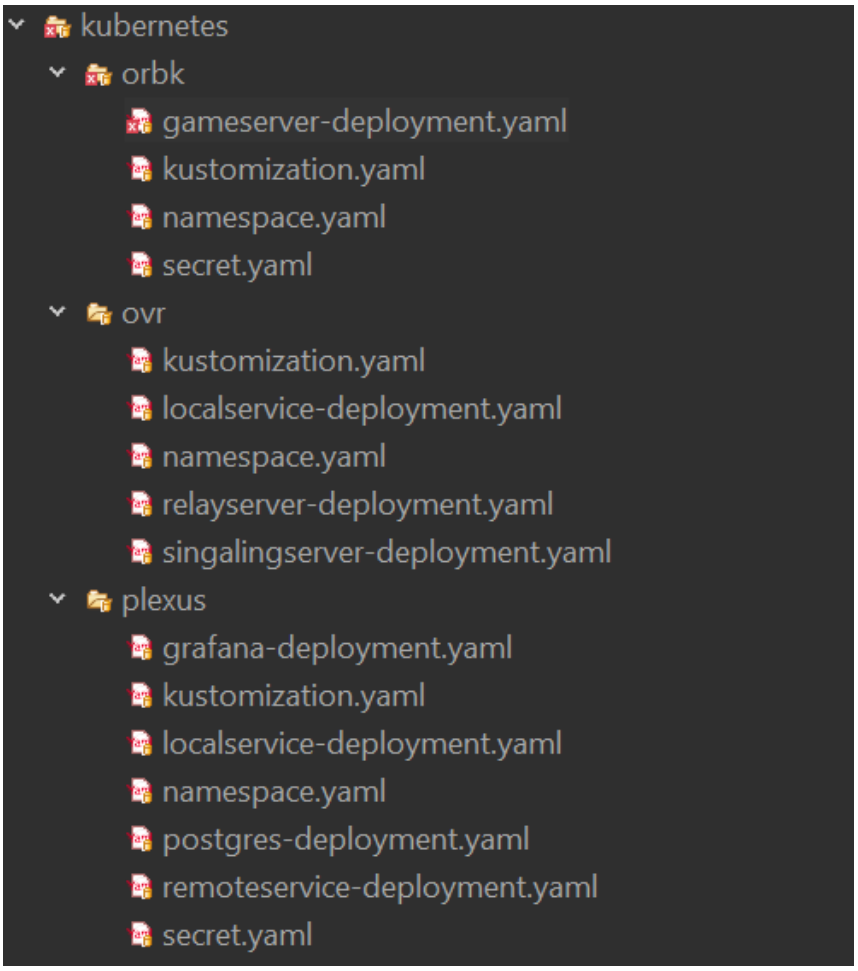
```
details:
  id: [redacted]
  name: UC 2
  version: 0.0.1
  isLatest: true
registry:
- repository: registry.gitlab.com/accordion-project/wp6/uc2/orbkaccserver
  version: latest
  id: 343ba97
  size: 146.94MB
  imageName: accordion-project/wp6/uc2/orbkaccserver
  component: GameServer
requirements:
- environment: PRODUCTION
  toscaDescription:
    tosca_definitions_version: tosca_simple_yaml_1_2
    description: Experiments with deployment of wordpress and mysql
    data_types: ...
    node_types: ...
    relationship_types: ...
    topology_template: ...
metadata:
  createdAt: '2021-03-25T13:41:55.042'
  createdBy: admin@accordion.eu
  modifiedAt: '2021-03-25T13:41:55.042'
  modifiedBy: user@accordion.eu
```

Figure 2: returned message from the Application Bucket for the 'orbk' Use Case

The document contains four different sections, where the **registry** and **requirements** are the most important sections for test purposes:

- **details**: base information about application; in particular, id, name and version number.
- **registry**: information about the stored images in the git repository. There is a *repository* section for each different application component image.
- **requirements**: the application model for the start application scenario of the requested Use Case. It is a TOSCA Yaml Simple profile model, partially modified for the tests to simplify the process of the Converter component and to satisfy requirements of Use Cases. We refer to this part as the *Intermediate TOSCA Model* (ITM). ITM is grouped by target environment: the start application scenario is represented by the PRODUCTION environment, as shown in Figure 2. The ITM is contained in the *toscaDescription* key.
- **metadata**: additional information about the record creation and modification

The document is submitted to the *Converter* component. Its function is to convert the ITM into a set of K3s configuration files that are supported by the *Virtual Infrastructure Manager* component (*VIM*) and that describe resources and mandatory actions to fetch images from the Application Bucket and deployed the various components of the selected Use Case over the K3S cluster. Figure 3 shown the output K3s YAML files for the three Use Cases.

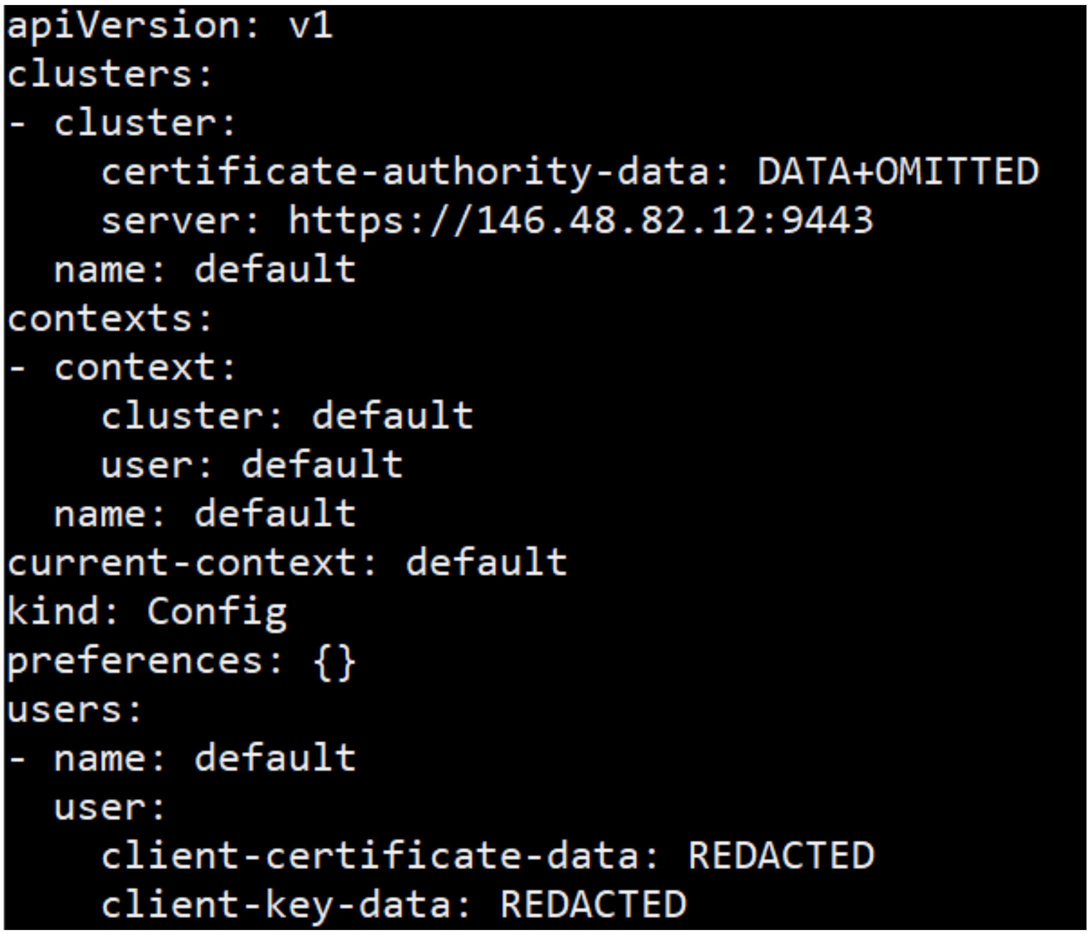


```
kubernetes
  orbk
    gameserver-deployment.yaml
    kustomization.yaml
    namespace.yaml
    secret.yaml
  ovr
    kustomization.yaml
    localservice-deployment.yaml
    namespace.yaml
    relayserver-deployment.yaml
    singalingserver-deployment.yaml
  plexus
    grafana-deployment.yaml
    kustomization.yaml
    localservice-deployment.yaml
    namespace.yaml
    postgres-deployment.yaml
    remoteservice-deployment.yaml
    secret.yaml
```

Figure 3: K3s Yaml files produced by the Converter component for each Use Case

All the output K3S files are stored locally inside the filesystem of the container of the *Orchestrator* component and not exposed externally.

The next step is to deploy all the application components on the *VIM*. In order to perform this action, the *Orchestrator* component needs to communicate remotely with the *VIM* component. To this end, the Python Kubernetes standard client library has been integrated in the *Orchestrator* component, which is able to communicate with the Master Node and, in particular, the API server of a remote Kubernetes installation. A particular configuration file is needed to access the remote *VIM*. Such configuration file is generated before starting the tests, and reflect the characteristics of the particular Kubernetes installation. Figure 4 shows an example of this configuration file.

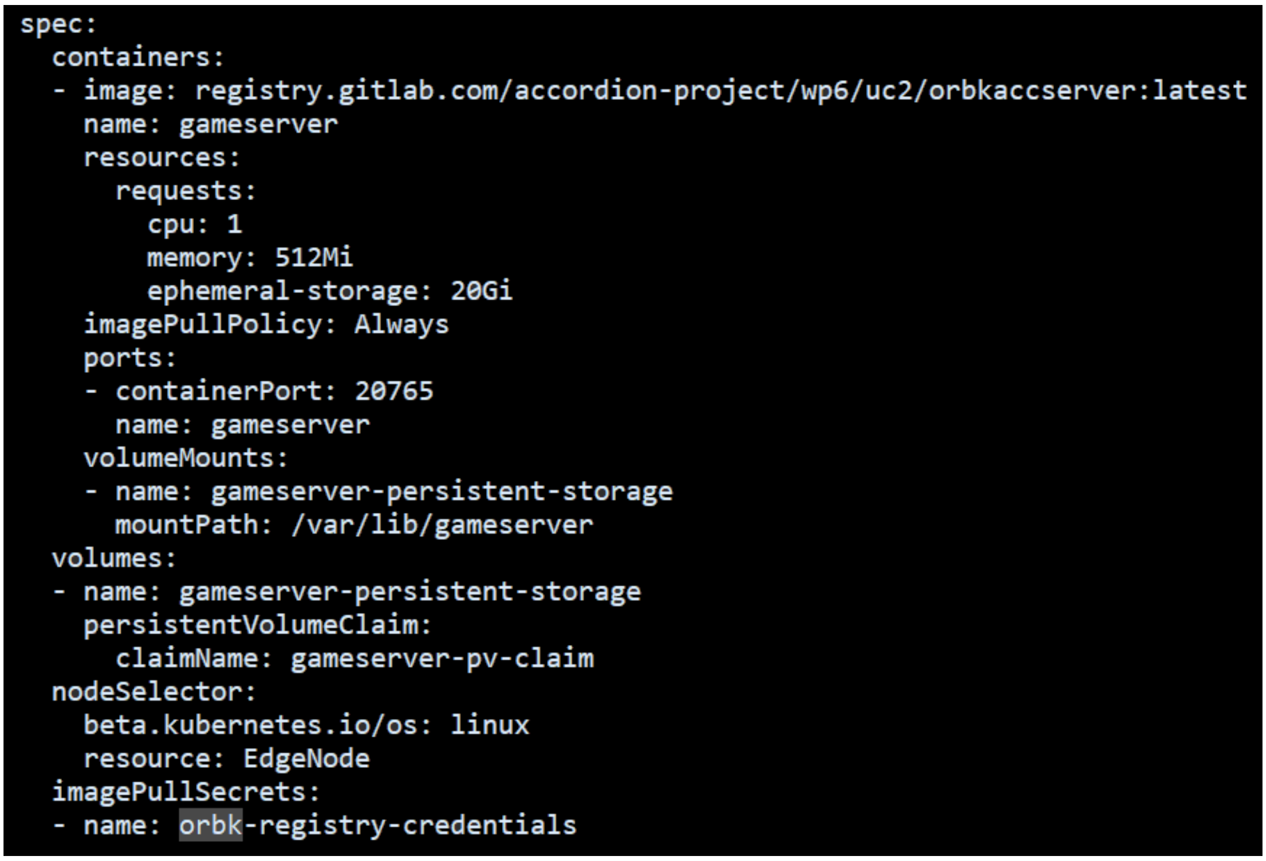


```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://146.48.82.12:9443
  name: default
contexts:
- context:
    cluster: default
    user: default
  name: default
current-context: default
kind: Config
preferences: {}
users:
- name: default
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

Figure 4: configuration file to access the remote Kubernetes cluster installation of the VIM

The configuration file specifies the credentials needed to access the remote Kubernetes installation and in particular the IP address of the Master node, specified in the **server** key. Such configuration is then loaded by the Kubernetes client library and all the YAML files of the tested Use Case are submitted to the *VIM* by invoking specific library function calls.

After the completion of this step, the *VIM* starts to deploy the application component images over the Kubernetes cluster on the Minicloud, interpreting and executing the submitted YAML files. To perform this action, *VIM* needs to access the *Application Registry* to fetch and download the Docker container or VMs images of the application components: such operation is performed automatically by exploiting the functionalities of Kubernetes to download Containers from a registry represented by a git repository. In Figure 5 is shown a part of the YAML K3s configuration file needed to deploy an instance of the **Game Server** service component over the Minicloud.



```
spec:
  containers:
  - image: registry.gitlab.com/accordion-project/wp6/uc2/orbkaccserver:latest
    name: gameserver
    resources:
      requests:
        cpu: 1
        memory: 512Mi
        ephemeral-storage: 20Gi
    imagePullPolicy: Always
    ports:
    - containerPort: 20765
      name: gameserver
    volumeMounts:
    - name: gameserver-persistent-storage
      mountPath: /var/lib/gameserver
  volumes:
  - name: gameserver-persistent-storage
    persistentVolumeClaim:
      claimName: gameserver-pv-claim
  nodeSelector:
    beta.kubernetes.io/os: linux
    resource: EdgeNode
  imagePullSecrets:
  - name: orbk-registry-credentials
```

Figure 5: K3s deployment descriptor for the Game Server of the ORBK Use Case

After fetching the needed application component images, the *VIM* component starts the application components.

Finally, after the deployment of the application, the *Monitoring* starts to collect metrics about the running applications services (i.e., *pods*), by using the Prometheus infrastructure. Figure 6, show the trend of the total amount of memory used by the game server of the ORBK Use Case over a period of two days.

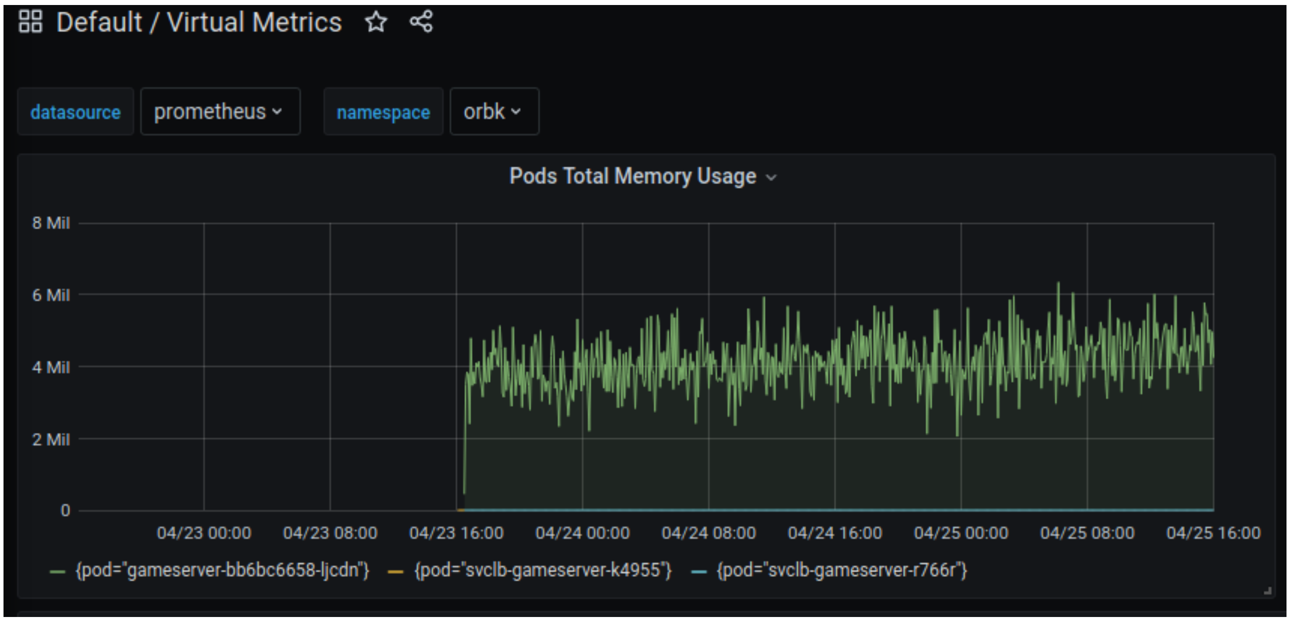


Figure 6. Total memory timeseries for the ORBK game server service

# 7 Conclusion and Next Steps

This document describes the status of the installation and integration of the ACCORDION Platform. At the current stage, the platform can fetch resources from an application repository, convert them into a specific format, and pass them to an execution environment. The platform can serve the purpose of an initial testing of the Use Case. Further integration actions are planned and will be performed in the close future. These actions include:

- The improvement of the interfaces of the current installing services, following the feedback received by the Use Case pilot testing.
- The installation of multiple *Minicloud* instances, possibly on different hardware configurations including System-on-Chip clusters (i.e., Raspberry Pi).
- New testing integration procedures (possibly automatic), which also include all the other services in the ACCORDION Platform.

Finally, the next version of this deliverable is expected for M31.