ACCORDION

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

Deliverable D3.2

Edge infrastructure pool framework 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
DOCUMENT INFORMATION	
WORKPACKAGE N. | TITLE	WP 3 | Edge infrastructure pool framework
WORKPACKAGE LEADER	HPE
DELIVERABLE N. | TITLE	D3.2 | Edge infrastructure pool framework implementation (I)
EDITOR	Lorenzo Blasi (HPE)
CONTRIBUTOR(S)	Ioannis Korontanis (HUA), Vangelis Psomakelis (ICCS), Hanna Kavalionak (CNR), Marco Di Girolamo (HPE), Alain Vailati (HPE), Felipe Huici (NEC), Patrizio Dazzi (CNR)
REVIEWER	Mateusz Kamiński (BSOFT)
CONTRACTUAL DELIVERY DATE	28/2/2021
ACTUAL DELIVERY DATE	28/2/2021
VERSION	V1.5
TYPE	Demonstrator
DISSEMINATION LEVEL	Public
TOTAL N. PAGES	49
KEYWORDS	Resource Management, Resource indexing, Resource Virtualization

EXECUTIVE SUMMARY

This document is the accompanying report documenting the software components that are released as part of ACCORDION Deliverable D3.2 and explains how to install and use them. The report includes the description of the first implementation provided by ACCORDION for the Edge infrastructure pool framework (or Edge minicloud), developed by the tasks inside WP3. The implemented minicloud model can include only resources located in a single site and typically owned by a single provider. The developed framework, representing one of the key innovations realized by ACCORDION, puts together the different components that altogether implement the functionalities requested to deploy and operate federated miniclouds inside the ACCORDION environment. It allows to locate and assign edge resources to the client applications of ACCORDION, ensuring they are duly registered, tracked, monitored and that the system is able to react whenever needed to ensure the quality of service to users of client applications. The framework implements the architectural guidelines and requirements set by WP2, will be further integrated with the higher (orchestration) layer components developed by WP4 and, once integrated, will run the Pilot use case applications proposed by WP6.

Whereas deliverable D3.1 extensively describes the research work carried on by WP3 to investigate and find out the best possible solutions to realize the different modules, D3.2 provides the actual implementation of the framework components, complemented by the present document. This document includes a section for each delivered software component: Edge minicloud VIM (sect. 2), Monitoring (sect. 3), Resource indexing and discovery (sect. 4), Edge storage (sect. 5), Unikraft (sect. 6). Each section describes how the related component has been actually implemented in the demonstrator (adopted software baseline, configurations and possible customizations), how it has been packaged and deployed and provides a user guide with all the needed directions to install and use the component in the ACCORDION system. It also includes information about the software license scheme adopted for the component and/or any of its constituent submodules. The ultimate purpose of this report is to make the reader potentially able to install and operate the ACCORDION framework software, besides understanding how it is structured and what packages make up its foundation.

The Edge minicloud is currently partially integrated as its components already run on the provided VIM baseline platform (K3s plus KubeVirt). Work for the next months includes increasing the integration level of the components and integrating them with components from other ACCORDION Work Packages, typically the WP4 Orchestrators. This integration phase will also allow analyzing the traffic that crosses the provider's firewall and tuning its configuration. Finally, the minicloud will be tested by running applications from the WP6 use cases, to evaluate if the provided functionalities and performance are suitable to satisfy the requirements and to indicate which improvements are needed. The next minicloud version will aim at maximizing users' expectations' satisfaction, based on the validation feedbacks, and may include additional components as indicated in the ACCORDION architecture. We will also explore the possibility of implementing a minicloud across multiple sites.

The deliverable D3.2, with both its prototypal and report constituents, represents the first release of the framework in the ACCORDION lifecycle. A second release will occur at project month 29 (D3.4), and the final one will be provided at project month 36 (D3.6). The second and third release will take advantage of the learnings collected during the two pilot phases.

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.

This document contains information on ACCORDION core activities. Any reference to content in this document should clearly indicate the authors, source, organisation and publication date.

The document has been produced with the funding of the European Commission. The content of this publication is the sole responsibility of the ACCORDION Consortium and its experts, and it cannot be considered to reflect the views of the European Commission. The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

The European Union (EU) was established in accordance with the Treaty on the European Union (Maastricht). There are currently 27 members states of the European Union. It is based on the European Communities and the member states' cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice, and the Court of Auditors (http://europa.eu.int/).

REVISION HISTORY LOG

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
V0.1	Created ToC	20/1/2021	Lorenzo Blasi (HPE)
V0.2	Monitoring section	11/2/2021	Ioannis Korontanis (HUA)
V0.3	Introduction and conclusions, list of figures and list of tables	13/2/2021	Lorenzo Blasi (HPE)
V0.4	First version of Edge storage subsection	15/2/2021	Vangelis Psomakelis (ICCS)
V0.5	Resource Indexing and Discovery	16/2/2021	Hanna Kavalionak (CNR)
V0.6	Exec Summary	16/2/2021	Marco DI Girolamo (HPE)
V0.7	Updated version of Edge storage subsection	16/2/2021	Vangelis Psomakelis (ICCS)
V0.8	VIM Component subsection	16/2/2021	Alain Vailati (HPE)
V0.9	Editing	16/2/2021	Lorenzo Blasi (HPE)
V1.0	Released for internal review	16/2/2021	Lorenzo Blasi (HPE)
V1.2	Unikraft subsection	22/2/2021	Felipe Huici (NEC)
V1.4	Merge of updates after internal review. Section on VIM moved just after the introduction	25/2/2021	Lorenzo Blasi (HPE)
V1.5	Final editing	25/2/2021	Lorenzo Blasi (HPE)
V1.6	Final typesetting	28/2/2021	Patrizio Dazzi (CNR)

GLOSSARY

ACES	ACCORDION Edge Storage
AI	Artificial Intelligence
API	Application Programming Interface
ARM	Advanced RISC (Reduced Instruction Set Computing) Machine
CPU	Central Processing Unit
DHT	Distributed Hash Table
DNS	Distributed Naming Service
DoA	Description of Action
EC	European Commission
EU	European Union
Gb	Gigabit
GB	Gigabyte
GPU	Graphical Processing Unit
H2020	Horizon 2020 EU Framework Programme for Research and Innovation
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service
ID	IDentifier
IP	Internet Protocol
K8s	Kubernetes
KB	Kilobyte
MB	Megabyte
PC	Personal Computer
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RDS	Resource Discovery System
REST	Representational State Transfer

TCP	Transmission Control Protocol
TOSCA	Topology and Orchestration Specification for Cloud Applications
UDP	User Datagram Protocol
VIM	Virtual Infrastructure Manager
VLAN	Virtual Local Area Network
VM	Virtual Machine

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Relevance to ACCORDION

1.1 Purpose of this document

The present document is the accompanying report documenting the software that is released as part of ACCORDION Deliverable D3.2. The document describes each released software component and explains how to install and use it.

1.2 Relevance to project objectives

The components described in this Deliverable make up a first version of what is indicated in the DoA as an "Edge minicloud". In the model implemented so far, each minicloud includes resources owned by a single provider and located in a single site.

The project Objective related to the work reported in this document is Objective 1: "Maximize edge resource pool size". The sub Objectives (Enablers) listed in the DoA for this Objective are addressed by WP3 Tasks and their related components as indicated in the following Table 1.

Table 1: Relationship between Enablers, Tasks and delivered Components

Enabler	Sub-Objective	Task	Component	Section
O1-E1	Resource monitoring & characterization	Task 3.1	Monitoring component	3
O1-E2	Resource indexing and discovery	Task 3.2	Resource indexing and discovery	4
O1-E3	Edge storage	Task 3.3	Edge storage	5
O1-E4	Hybrid elasticity	Task 3.4	Unikraft	6
		Task 3.5	Minicloud VIM	2

1.3 Relation to other work packages

The software released as part of this Deliverable is a product of WP3, but the work done has been also guided by WP2 (D2.1 - User requirements, D2.3 - Architecture design) and WP7 (D7.5 - Technoeconomic analysis). The Edge minicloud, composed of results documented in this report, is expected to be used by the ACCORDION Orchestrators developed by WP4 and once integrated, will run the Pilot use case applications proposed by WP6.

1.4 Structure of the document

The document is organized with a section for each delivered software component. Each section provides the following subsections:

- Component description: includes a technical description of the component, specification of any reused open-source components, and any prerequisite to be satisfied.
- Package information: describes the structure and content of the delivered package
- Installation instructions: explains how to install and start up the software

- User manual: provides details on how to use the software
- Licensing information: indicates the license applicable to the delivered component, plus version and license of any included open source components and libraries

Section 2 is about the Edge minicloud VIM component produced by Task 3.5.

Section 3 is about the Monitoring component produced by Task 3.1.

Section 4 is about the Resource indexing and discovery component produced by Task 3.2.

Section 5 is about the Edge storage component produced by Task 3.3.

Section 6 is about the Lightweight virtualization system (Unikraft) produced by Task 3.4.

2 VIM Component

2.1 Component description

This component is dedicated to the management of hardware and software infrastructure, in order to permit execution and management of workloads of three kinds: Docker[1] container, virtual machine and unikernel. Vim offers all services needed to run workloads like network connections, persistent volumes, kernel functions etc. The VIM choose for Accordion is named K3s[2]: a Kubernetes[3] distribution tailored for edge computing by Rancher team and released to the open-source community. This orchestrator customization keeps compatibility with mainstream Kubernetes but has modifications for optimizing installation on the edge. K3s reorganize Kubernetes agent/server processes running on hosts: there is only one process "k3s-server" on master's node and only one process "k3s-agent" on worker creating a Kubernetes cluster, and lots of packages are not included in the simplified installation to keep as light as possible. More details can be found on the K3s project website[2].

On top of K3s cluster will be installed KubeVirt[4], a Kubernetes extension that can run a virtual machine, KubeVirt leverage Custom Resources Definitions (CRD) to add virtual machine (VM) and virtual machine instance (VMI) to Kubernetes objects.

2.2 Package information

The current version of the VIM installation scripts can be cloned with the following command (credentials are needed, which could be available upon request):

```
$ git clone -b v1.0.0 https://gitlab.com/accordion-project/wp3/vim-installation.git .
```

The released package is composed of five installation Shell scripts. These scripts will download necessary packages from the Internet network.

- install_first_node.sh
- install_secondary_master.sh
- install_worker.sh
- install_kubevirt.sh
- install_virtctl.sh

2.3 Installation instructions

VIM installation is performed via Linux CLI scripts that must be executed with sudo grant, also execution permission could be necessary for the environment once copied the selected installation script to the host.

[1] https://www.docker.com/
[2] https://k3s.io/
[3] https://kubernetes.io/
[4] https://kubevirt.io/

To grant execute permission you can use chmod as below:

```
$ chmod 744 install_first_node.sh
```

In our VIM installation procedure, the first step is to install the K3s cluster starting from the first master node.

Execute `install_first_node.sh` script, the sudo password will be requested during installation, as shown in Figure 1 below:

```
osboxes@osboxes:~$ ./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
[sudo] password for osboxes:
[INFO]  Using v1.20.2+k3s1 as release
[INFO]  Downloading hash https://github.com/rancher/k3s/releases/download/v1.20.2+k3s1/sha256sum-amd64.txt
[INFO]  Downloading binary https://github.com/rancher/k3s/releases/download/v1.20.2+k3s1/k3s
[INFO]  Verifying binary download
[INFO]  Installing k3s to /usr/local/bin/k3s
[sudo] password for osboxes:
[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 Tue 2021-02-16 08:24:56 UTC; 48ms ago
Installation success!
osboxes@osboxes:~$
```

Figure 1: First VIM master node installation

Result of this installation is the first cluster node working, this is a K3s master with etcd cluster initialized and ready to accept join request from other nodes. The second result is the node token, the key needed to log into the cluster. As can be seen in the figure, the K3s token[5] is placed in the file: `/var/lib/rancher/k3s/server/node-token` and kept for the next installation steps.

If high availability is required, a second master and possibly a third master node must be installed. On those nodes' shell prompt, with sudo privilege, execute the script `install_secondary_master.sh`, passing two mandatory parameters: K3s token and the IP address of the first master installed, as shown in Figure 2.

[5] See https://rancher.com/docs/k3s/latest/en/quick-start/

```
osboxes@osboxes:~$ sudo ./install_secondary_master.sh K10c448315d9a4af637377c4e0f5342d35a70eaa2fb60182da33ea1f4350600658f::server:e561
 192.168.1.14
ACCORDION: Installing Minicloud VIM secondary master node
[INFO]  Using v1.20.2+k3s1 as release
[INFO]  Downloading hash https://github.com/rancher/k3s/releases/download/v1.20.2+k3s1/sha256sum-amd64.txt
[INFO]  Downloading binary https://github.com/rancher/k3s/releases/download/v1.20.2+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 Tue 2021-02-16 10:26:22 UTC; 9ms ago
Installation success!
osboxes@osboxes:~$
```

Figure 2: Additional VIM master node installation

Once all master nodes are installed, it's time for any worker nodes to join the cluster. On each worker node, with sudo privileges, execute the script `install_worker.sh` passing two mandatory parameters: K3s-token and the IP address of the first master installed. A third parameter called *skipKubeVirt*, if present, will disable all KubeVirt system checks in the installation script. These checks verify if the host OS can run KVM virtual machines. If you have multiple worker nodes you can choose to dedicate some of them only to containers: the *skipKubeVirt* parameter should be used when installing these container-only nodes.

```
osboxes@osboxes:~$ ./install_worker.sh  K10a433a62d408161cb708fe6773715c3756b77ca6ad8c86acdc868c6b29a3cca48::server:daadf2676d74f0c950c9ca87b9ee3d71 192.168.1.14
ACCORDION: Installing Minicloud VIM worker node
Reading package lists... Done
Building dependency tree
Reading state information... Done
cpu-checker is already the newest version (0.7-1.1).
0 upgraded, 0 newly installed, 0 to remove and 123 not upgraded.
Reading package lists... Done
Building dependency tree
Reading state information... Done
libvirt-clients is already the newest version (6.0.0-0ubuntu8.5).
0 upgraded, 0 newly installed, 0 to remove and 123 not upgraded.
[INFO]  Using v1.20.2+k3s1 as release
[INFO]  Downloading hash https://github.com/rancher/k3s/releases/download/v1.20.2+k3s1/sha256sum-amd64.txt
[INFO]  Downloading binary https://github.com/rancher/k3s/releases/download/v1.20.2+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-agent-uninstall.sh
[INFO]  env: Creating environment file /etc/systemd/system/k3s-agent.service.env
[INFO]  systemd: Creating service file /etc/systemd/system/k3s-agent.service
[INFO]  systemd: Enabling k3s-agent unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s-agent.service → /etc/systemd/system/k3s-agent.service.
[INFO]  systemd: Starting k3s-agent
     Active: active (running) since Tue 2021-02-16 11:37:29 UTC; 12ms ago
K3S Installation success!
```

Figure 3: VIM worker node installation (for both containers and VMs)

After the installation of all worker nodes, the K3s cluster is complete and it's time to install KubeVirt.

From a master node console run the script `install_kubevirt.sh`, it will install all Kubernetes object for KuberVirt engine and will instantiate it.

```
osboxes@osboxes:~$ sudo ./install_kubevirt.sh
[sudo] password for osboxes:
ACCORDION: Installing KubeVirt
     Active: active (running) since Tue 2021-02-16 13:46:38 UTC; 22min ago
Server Version: v1.20.2+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+
n
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
error: timed out waiting for the condition on kubevirts/kubevirt
```

Figure 4: Kubevirt installation

Depending on HW resources this process can take several minutes. Even if, as in Figure 4, a time out occurs, the startup is going on and can be verified after the script termination. To check if startup is complete, just query the K3s cluster for Pods status in the KubeVirt namespace as in the following Figure 5:

```
osboxes@osboxes:~$ kubectl get pod -n kubevirt
NAME                               READY   STATUS              RESTARTS   AGE
virt-api-78db556598-8wv8b          1/1     Running             0          2m7s
virt-api-78db556598-g66mb          1/1     Running             0          2m7s
virt-controller-f9c8d7986-4ll9x    1/1     Running             0          87s
virt-controller-f9c8d7986-82mq5    1/1     Running             0          87s
virt-handler-dzc7f                 0/1     ContainerCreating   0          87s
virt-handler-f2vft                 0/1     ContainerCreating   0          87s
virt-handler-jx6vc                 0/1     ContainerCreating   0          87s
virt-operator-648dbdd989-4mdp6     1/1     Running             0          3m30s
virt-operator-648dbdd989-g748k     1/1     Running             0          3m30s
osboxes@osboxes:~$ kubectl get pod -n kubevirt
NAME                               READY   STATUS    RESTARTS   AGE
virt-api-78db556598-8wv8b          1/1     Running   0          8m21s
virt-api-78db556598-g66mb          1/1     Running   0          8m21s
virt-controller-f9c8d7986-4ll9x    1/1     Running   1          7m41s
virt-controller-f9c8d7986-82mq5    1/1     Running   0          7m41s
virt-handler-dzc7f                 1/1     Running   0          7m41s
virt-handler-f2vft                 1/1     Running   0          7m41s
virt-handler-jx6vc                 1/1     Running   0          7m41s
virt-operator-648dbdd989-4mdp6     1/1     Running   0          9m44s
virt-operator-648dbdd989-g748k     1/1     Running   1          9m44s
osboxes@osboxes:~$
```

Figure 5: Kubevirt pods status

To manage VMs KubeVirt provides its own command-line tool: *virtctl*[6]. Installation of virtctl on the master node can be done with the script `install_virtctl.sh` as shown in Figure 6 below.

```
osboxes@osboxes:~$ sudo ./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   6459      0 --:--:-- --:--:-- --:--:--  6459
100 46.1M  100 46.1M    0     0  4684k      0  0:00:10  0:00:10 --:--:-- 4977k
Virtctl installed
```

Figure 6: Virtctl installation

You can check the installed tool by running:

```
$ ./virtctl version
```

2.4 User manual

K3s is a Kubernetes CFCN distribution, therefore user manuals can be found on the K8S official site (https://kubernetes.io) for the majority of cases, while for details on customizations please refer to Rancher documentation website (https://k3s.io/).

KubeVirt user manual can be found at official site: https://kubevirt.io/user-guide/docs/latest/welcome/index.html. KubeVirt has been started and is mainly supported by RedHat, which uses it to support VMs within RedHat OpenShift. Therefore some documentation on KubeVirt can also be found in OpenShift manuals and blog sites. For example the list of command line parameters for virtctl can be found in an OpenShift document[12].

2.4.1 K3s Hello World tutorial

In this section, there are instructions on how to run a simple "Hello World" example on a K3s cluster. This example will deploy an NGNIX container[7] with a web page that shows details on the web requests.

To start this tutorial you need a running K3s cluster with at least one master node (installation instructions can be found in section 2.3).

First, create a basic deployment "hello-node" pointing to nginx demo container image. This image runs an NGINX webserver with a webpage showing HTTP request details. The needed K8S objects can be created using *kubectl*[8], a tool installed for root user during the K3s installation process:

```
$ kubectl create deployment hello-node --image=nginxdemos/hello

deployment.apps/hello-node created
```

Check if the deployed Pod is in running state:

[6] https://docs.openshift.com/container-platform/4.2/cnv/cnv_users_guide/cnv-using-the-cli-tools.html
[7] https://hub.docker.com/r/nginxdemos/hello/
[8] https://kubernetes.io/docs/reference/kubectl/overview/

```
$ kubectl get pods

NAME                           READY   STATUS    RESTARTS   AGE
hello-node-7bc5b6d85d-qfcsj    1/1     Running   0          3s
```

Now expose the deployment as a service type NodePort:

```
$ kubectl expose deployment hello-node --type=NodePort --port=80

service/hello-node exposed
```

To see details about the created Service:

```
$ kubectl describe service
Name:                     hello-node
Namespace:                default
Labels:                   app=hello-node
Annotations:              <none>
Selector:                 app=hello-node
Type:                     NodePort
IP Families:              <none>
IP:                       10.43.86.127
IPs:                      10.43.86.127
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30592/TCP
Endpoints:                10.42.1.10:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Highlighted in yellow you can find the port number to use the service: from a browser on the host network it is possible to reach the NGINX server as http://hostipaddr:port/request

For example:

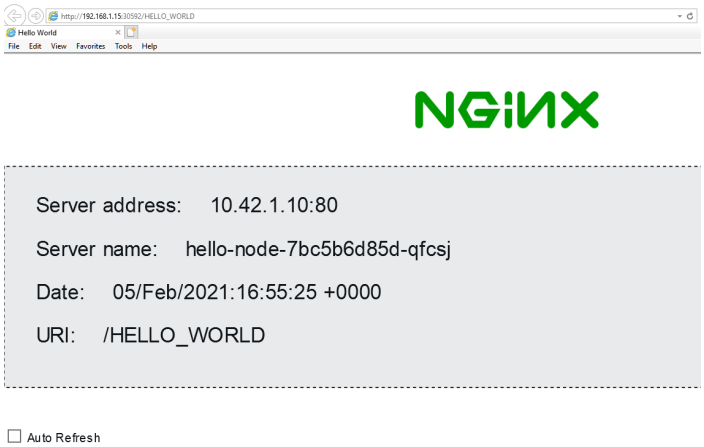

Figure 7: Testing the K3s Hello World example

2.4.2 KubeVirt Hello World

This section shows a simple example of KubeVirt flow taken from KubeVirt GitHub repository[9]. To execute this example you need a running K3s cluster with KubeVirt installed on it (installation instructions can be found in section 2.3).

```
# Create a vm
$ kubectl apply -f https://raw.githubusercontent.com/kubevirt/demo/master/manifests/vm.yaml

# Check VM description
$ kubectl describe vm testvm

# The vm is not running by now, to start a vm use follow command that create a virtualm
machine instance VMI
$ ./virtctl start testvm

# Inspect instance created
$ kubectl describe vmi testvm

# Log into the vm console
$ ./virtctl console testvm


# To shut the VM down:
$ ./virtctl stop testvm

# To delete
$ kubectl delete vm testvm
```

2.5 Licensing information

Both KubeVirt and K3s are licensed under the Apache License 2.0, a permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications and larger works may be distributed under different terms and without source code.

[9] https://github.com/kubevirt/demo

3 Monitoring Component

3.1 Component description

The goal of the monitoring component is to be able to monitor Cloud & Edge resources, it is needed to monitor nodes and their Pods in a K3s cluster to be able to know the workload. Prometheus[10] pulls metrics from the exporters (Node Exporter[11], Kube-state-metrics[12] and Prometheus Operator[13]) then Grafana[14] queries Prometheus to visualize the data on graphs, and Monitoring API queries Prometheus to return JSON responses to the rest components of ACCORDION that communicate with monitoring. Monitoring API is a component running in a Pod that pulls hardware information from the characterization agents to store them in MongoDB and creates the characterization description. Monitoring API also queries Prometheus to retrieve metrics that other ACCORDION components need. All Docker images can be found on Dockerhub except from monitoring_api and char-agent images, these images are a custom solution based on Debian (stable version) Docker image[15]and their code has to be pulled from a private Gitlab repository[16] that was created for ACCORION components and then you can build the Docker images. The monitoring component can only be installed on ARM and AMD x86-64 architectures. The monitoring component has been tested in K3s Client Version v1.20.0+k3s2 and Docker version 20.10.0 (build 7287ab3) and it has a multi Pod architecture (Figure 8):

- Node exporter for hardware and OS metrics. (Docker image: prom/node-exporter: v1.1.1[10])
- Kube-state-metrics expose critical metrics about the condition of a Kubernetes cluster (health of nodes, pods, deployments, etc.), it generates them from the Kubernetes API server. (Docker image:carlosedp/kube-state-metrics:v1.9.6 [17])
- Grafana to visualize the metrics with graphs. (Docker image: grafana/grafana:7.0.3[18])
- Prometheus monitoring system and time series database. (Docker image: prom/prometheus:v2.19.1[19])
- Prometheus Operator for Kubernetes provides easy monitoring definitions for Kubernetes services and deployment and management of Prometheus instances. (Docker image: carlosedp/prometheus-operator:v0.40.0[20])
- Prometheus Adapter is an implementation of the Kubernetes resource metrics, custom metrics, and external metrics APIs. (Docker image: directxman12/k8s-prometheus-adapter:v0.7.0[21])
- Alert Manager that handles alerts sent by client applications such as the Prometheus server. (Docker image: prom/alertmanager:v0.21.0[22])

[10] https://prometheus.io/
[11] https://hub.docker.com/r/prom/node-exporter/tags?page=1&ordering=last_updated
[12] https://github.com/kubernetes/kube-state-metrics
[13] https://github.com/carlosedp/prometheus-operator
[14] https://grafana.com/
[15] https://hub.docker.com/_/debian?tab=tags&page=1&ordering=last_updated
[16] https://gitlab.com/accordion-project
[17] https://hub.docker.com/r/carlosedp/kube-state-metrics/tags?page=1&ordering=last_updated
[18] https://hub.docker.com/r/grafana/grafana/tags?page=1&ordering=last_updated
[19] https://hub.docker.com/r/prom/prometheus/tags?page=1&ordering=last_updated
[20] https://hub.docker.com/r/carlosedp/prometheus-operator/tags?page=1&ordering=last_updated
[21] https://hub.docker.com/r/directxman12/k8s-prometheus-adapter/tags?page=1&ordering=last_updated
[22] https://hub.docker.com/r/prom/alertmanager/tags?page=1&ordering=last_updated

- Characterization-agent exposes hardware information for a device/node. (Docker image: char-agent)
- Monitoring API which queries Prometheus to retrieve information and represent it in JSON in the endpoints. (Docker image: monitoring_api)
- MongoDB which stores the information of the characterization agents. (Docker image: mongo:4.0.21[23])

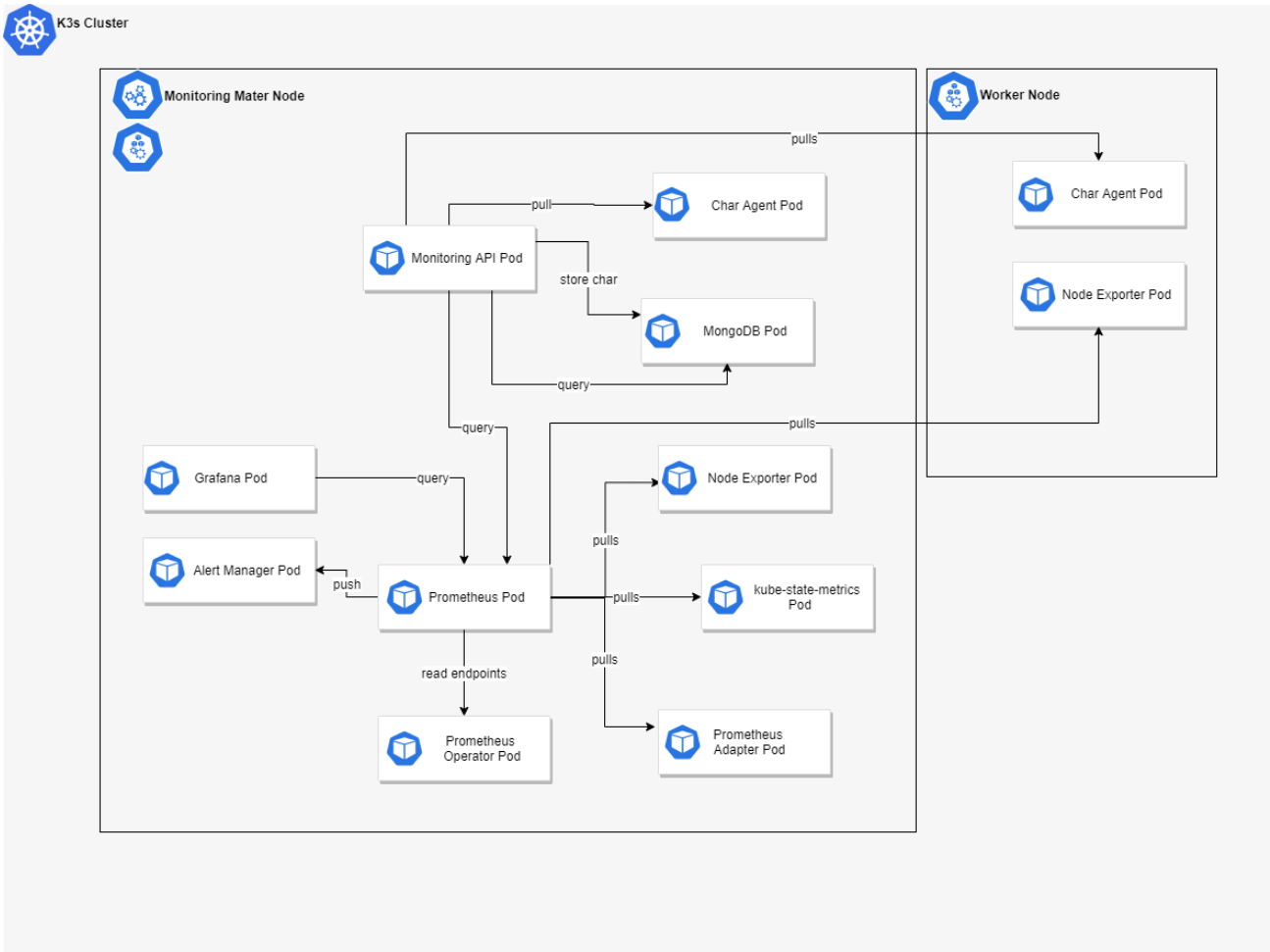

Figure 8: Monitoring Pod Architecture

The monitoring Pods which have to be deployed on the K3s master node are Prometheus, Grafana, Alert Manager, MongoDB, Kube-state-metrics, Prometheus Adapter and Monitoring API pod. Prometheus Adapter and kube-state-metrics have to be deployed on the master to be able to communicate with the Kubernetes API server. In the experiments of deployment that we conducted when Prometheus, Grafana and Alert manager pods were deployed on worker nodes they failed to run. The procedure of pulling the code and building the Docker images is described in the following Section 3.2 Package Information. Then there is Section 3.3 Installation instructions which describes the installation procedure. Please note that before moving to Section 3.3 and perform the installation procedures

[23] https://hub.docker.com/_/mongo?tab=tags&page=3&ordering=last_updated

you should disable the firewall on every machine (master and workers) of your K3s cluster, the command[24] to perform this action is:

```
$ sudo ufw disable
```

3.2 Package information

3.2.3 Monitoring API

Before installing the monitoring component, the first step would be to clone the Monitoring API project from the repository to the master node of K3s and then create the Docker image of the Monitoring API. For this procedure credentials are needed, which could be available upon request. The following command is the one that clones the project to your K3s master node:

```
$ git clone -b v1.0.0 https://gitlab.com/accordion-project/wp3/monitoringapi.git
```

When cloning is done the structure of the project should look like this:

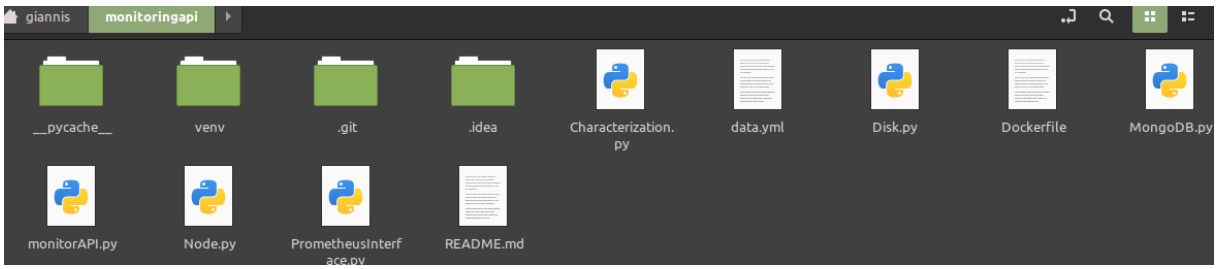

Figure 9: Monitoring API file structure

File named monitoringAPI.py is basically the API which is based on Flask[25]. For the monitoring endpoint `monitoringAPI.py` calls methods from `PrometheusInterface.py` which have the PromQL[26] queries predefined to perform them on the Prometheus pod. For the characterization endpoint `monitoringAPI.py` calls methods from `Characterization.py`. The file `Characterization.py` is the one that pulls hardware information from the characterization agents, then MongoDB.py is being used to store this information in the MongoDB pod and creates the `data.yml` file which is an extended TOSCA description that can describe the nodes of a K3s cluster. `Node.py` and `Disk.py` are basically classes with getters and setters that are being used by `Characterization.py` to parse the JSON responses of characterization agents. The procedure of building the Docker image could take a couple of minutes. To build the Docker image the command is:

```
$ docker build -f Dockerfile -t monitoring_api .
```

[24] https://help.ubuntu.com/community/UFW
[25] https://flask.palletsprojects.com/en/1.1.x/
[26] https://prometheus.io/docs/prometheus/latest/querying/basics/

If the procedure was successful in the result the last two lines should like this:

```
Successfully built 383da1f4e60c
Successfully tagged monitoring_api:latest
```

Figure 10: Successful Docker image

More information for the usage of the Monitoring API can be found in the Section 3.4 User manual.

3.2.4 Characterization Agent

Another project that should be cloned from a repository is the characterization agent. This time, characterization should be cloned to every node of K3s and then the image should be built in each of them. The cloning command is the following, as it is obvious from the previous cloned project, credentials are needed to clone the project:

```
$ git clone -b v1.0.0 https://gitlab.com/accordion-project/wp3/char-agent.git
```

The structure of the project should look like this:

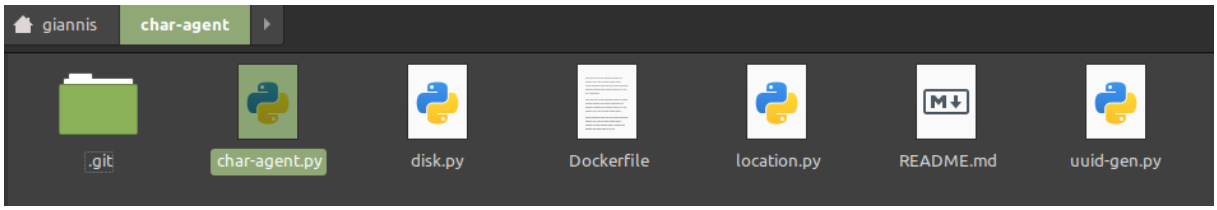

Figure 11: Characterization Agent file structure

`Char-agent.py` file is actually an API that calls `disk.py` and `location.py` to retrieve the required information about the hard disk and location of the node. The `uuid-gen.py` is also called by `Char-agent.py` to generate a UUID for the device. `Char-agent.py` by its own retrieves CPU, RAM, OS, GPU information and if the device has a battery or not.

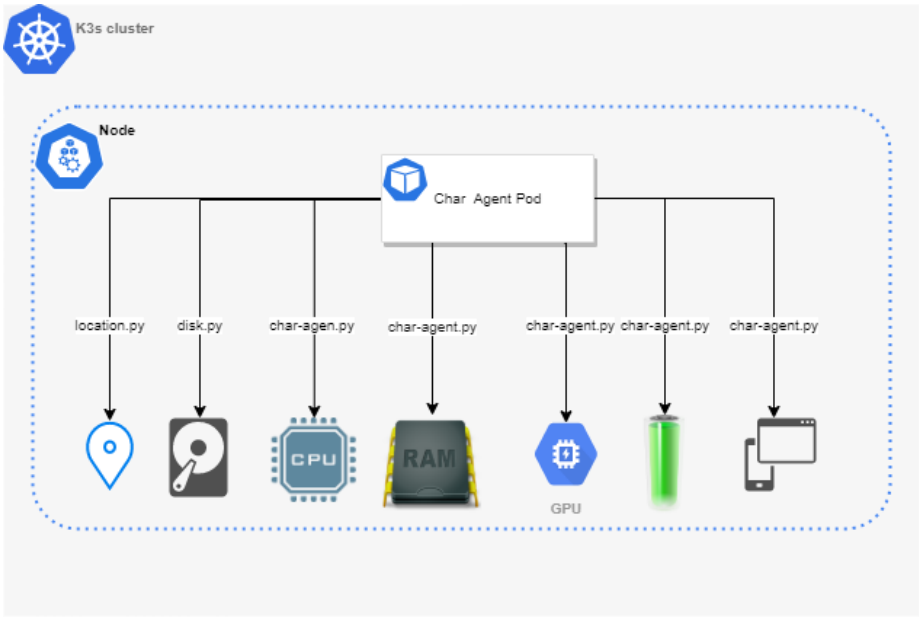

Figure 12: Characterization agent

The Docker image build command has the same syntax as the previous one. If it is successfully built, then the result would have to show the same message as before:

```
$ docker build -f Dockerfile -t char-agent  .
```

Afterwards it is required to run the following commands in the worker nodes (bare metals or VMs) of the K3s cluster:

```
$ xhost local:root
$ export DISPLAY=':0.0'
```

Without these commands, characterization-agent containers won't be able to find the GPU model and characteristics of the workers. The GPU information is retrieved from the mesa-utils[27] and to be more specific from the glxinfo which also requires connection to the Internet. There was also a trick for the export command in Raspberry Pi 4B to avoid the errors you have to configure a static screen resolution or else export will give you an error in the next boot if the Raspberry is not connected to a monitor.

3.3 Installation instructions

After cloning and building the components, as indicated in the previous subsection, the next operation would be to clone the installation scripts from the repository to the K3s master node to start the installation procedure:

```
$ git clone -b v1.0.0 https://gitlab.com/accordion-project/wp3/monitoring-installation.git
```

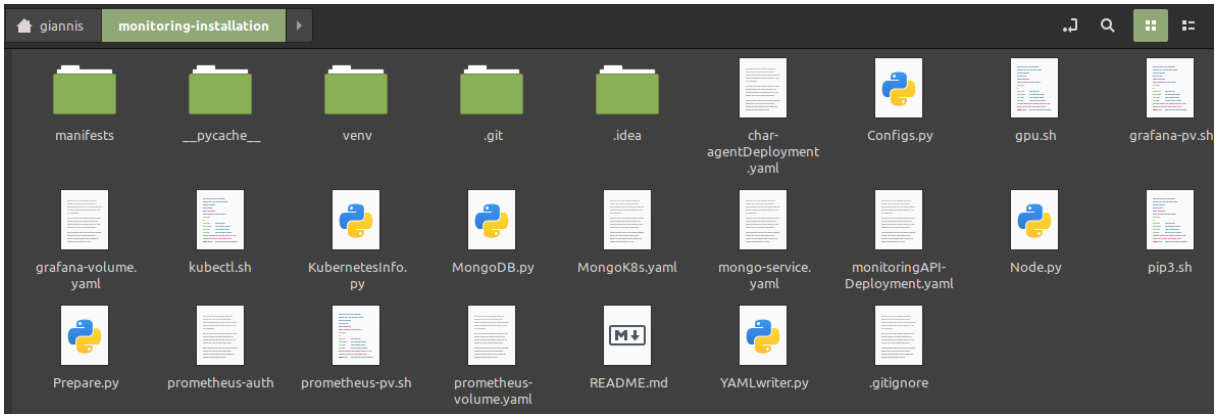

Figure 13: monitoring-installation file structure

The first step would be to install pip3[28]. If it not already installed in your machine run the following command:

```
$ sudo apt-get -y install python3-pip
```

Then you could run the `Prepare.py` to prepare your machine for the installation:

```
$ sudo python3 Prepare.py
```

[27] https://wiki.debian.org/Mesa
[28] https://help.dreamhost.com/hc/en-us/articles/115000699011-Using-pip3-to-install-Python3-modules

By running the Prepare script, that is located in the monitoring-installation project, the appropriate folders that will store the data of Prometheus and Grafana of the monitoring stack are created by calling `prometheus-pv.sh` and `grafana-pv.sh`. In addition, required python libraries are going to be installed by the `pip.sh` script called by `Prepare.py`. It also makes the GPU of the master available for the characterization-agent by executing the same commands that were previously executed on the workers. If the operation was successful the output would be as shown in Figure 14.

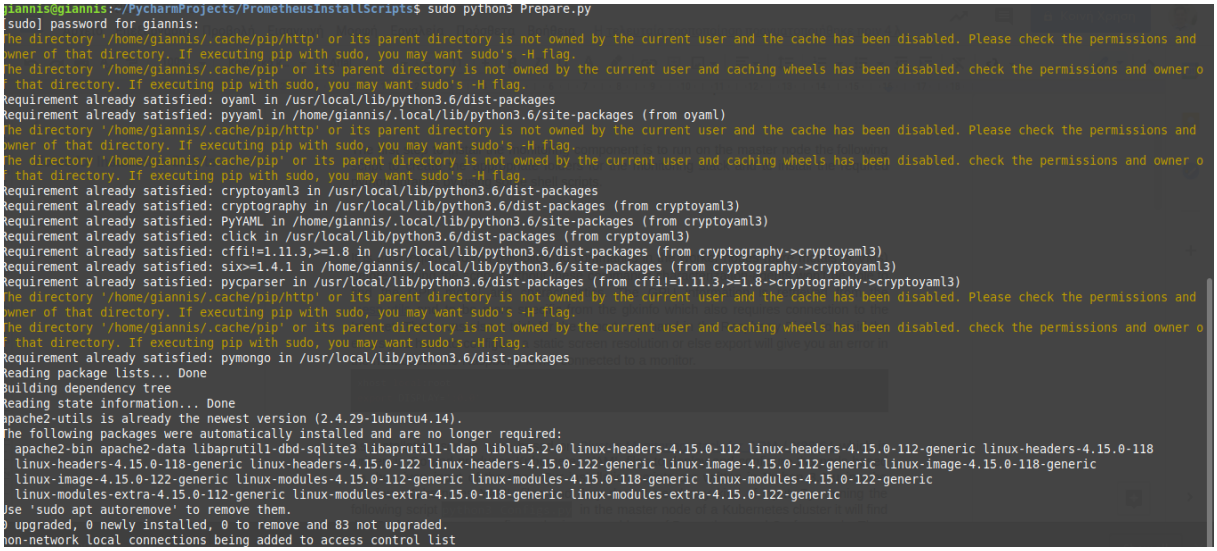

Figure 14: Result of Prepare.py

The final step would be to run the config python script to install the monitoring stack. By running the following command in the master node the script the Monitoring component will be installed in the K3s cluster nodes:

```
$ python3 Configs.py
```

The script will find the IP of the master to configure the Ingress[29] address of Alert manager, Prometheus and Grafana pods. The first two YAML files are needed for Prometheus Operator to configure the endpoints of Prometheus. `YAMLwriter.py` uses the master node IP to create:

- prometheus-kubeControllerManagerPrometheusDiscoveryEndpoints.yaml
- prometheus-kubeControllerSchedulerPrometheusDiscoveryEndpoints.yaml
- ingress-prometheus.yaml
- ingress-grafana.yaml
- ingress-alertmanager.yaml

`Configs.py` secures that Prometheus, Grafana, kube-state-metrics, Monitoring API and Prometheus Operator Pods are going to be deployed on the master node, a label is applied to the master node and in the associated files, in the nodeSelector[30] segment, the label is also added. Then a password is randomly generated to be used in the Prometheus Ingress to add basic-auth to it and is stored with htpasswd[31] then from this file a Kubernetes secret is

[29] https://kubernetes.io/docs/concepts/services-networking/ingress/
[30] https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/
[31] https://httpd.apache.org/docs/2.4/programs/htpasswd.html

created that is known to the Monitoring API. The password and the key are stored in encrypted YAML files at the home path of the local machine, the encryption algorithm is Fernet symmetric. `Configs.py` calls `KubernetesInfo.py` to parse the result of `kubectl get nodes -o wide` finds the rest nodes of the cluster, this is done to store the IPs of the nodes to the MongoDB pod with the help of `MongoDB.py` for characterization purposes, when Monitoring API is deployed adds the rest of information to the MongoDB. The manifests folder which contains several K3s configuration files is based on the Github project Cluster Monitoring stack for ARM / X86-64,[32] these files are applied along with those that are outside the folder to K3s with the kubectl.sh script that is being called by Configs.py.

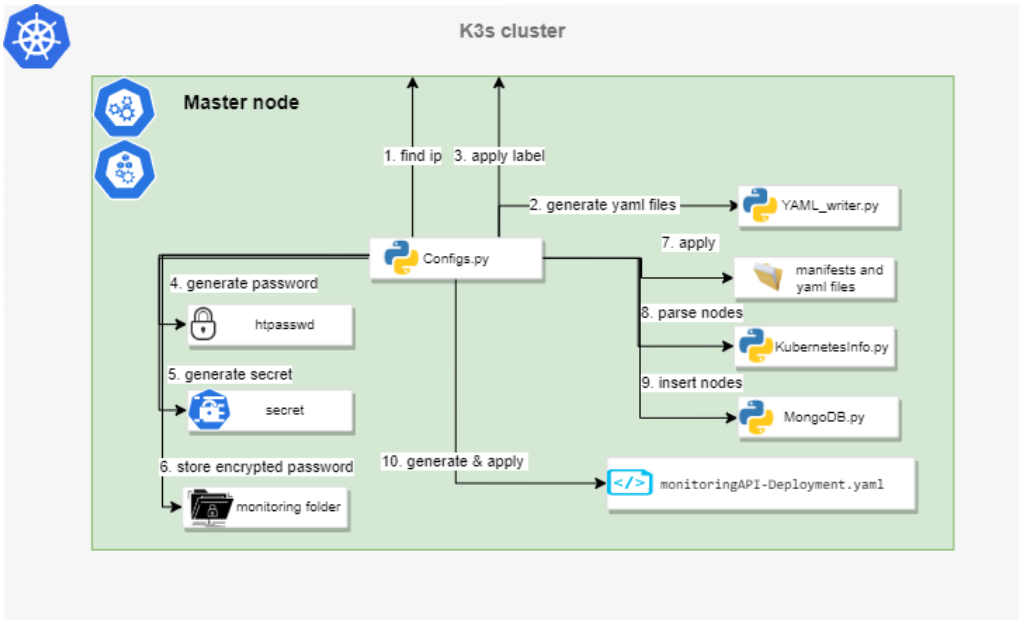

Figure 15: Configs.py

The `Configs.py` script should take a few seconds to deploy the pods, its response of messages is too long but you can check if the installation was successful (Figure 16) by running the following command which retrieves all the pods of the monitoring and their current state:

```
$ kubectl get -n monitoring pods -o wide
```

```
giannis@giannis:~$ kubectl get -n monitoring pods -o wide
NAME                                  READY   STATUS    RESTARTS   AGE   IP              NODE          NOMINATED NODE   READINESS GATES
prometheus-operator-67755f959-2m6ql   2/2     Running   0          74s   10.42.2.13      raspberrypi1  <none>           <none>
char-agent-57gwx                      1/1     Running   0          70s   192.168.1.203   raspberrypi1  <none>           <none>
char-agent-fc9v9                      1/1     Running   0          70s   192.168.1.205   raspberrypi   <none>           <none>
node-exporter-dz6xs                   2/2     Running   0          72s   192.168.1.205   raspberrypi   <none>           <none>
node-exporter-n2cwk                   2/2     Running   0          72s   192.168.1.203   raspberrypi1  <none>           <none>
char-agent-dk7kd                      1/1     Running   0          70s   192.168.1.2     giannis       <none>           <none>
prometheus-adapter-8d9968f86-47l29    1/1     Running   0          72s   10.42.0.156     giannis       <none>           <none>
mongo-remote-6fd6dd5897-zbvht         1/1     Running   0          72s   192.168.1.2     giannis       <none>           <none>
kube-state-metrics-b857584fd-5ljpw    3/3     Running   0          72s   10.42.0.158     giannis       <none>           <none>
node-exporter-8j826                   2/2     Running   0          72s   192.168.1.2     giannis       <none>           <none>
alertmanager-main-0                   2/2     Running   0          71s   10.42.0.159     giannis       <none>           <none>
prometheus-k8s-0                      3/3     Running   1          71s   10.42.0.160     giannis       <none>           <none>
monitoring-api-65df65ddd-fnh6c        1/1     Running   0          38s   192.168.1.2     giannis       <none>           <none>
grafana-6bbdd996bc-nwgtg              1/1     Running   0          72s   10.42.0.157     giannis       <none>           <none>
```

Figure 16: Succesful Installation

[32] https://github.com/carlosedp/cluster-monitoring

3.4 User manual

After the successful installation of the monitoring stack, the user should be able to access the Monitoring API and the graphical user interface of Grafana. The Monitoring API is configured with the IP of the host, which is the IP of the K3s master node. The monitoring endpoint of the Monitoring API is using two HTTP parameters: metric and namespace. The metric parameter indicates which queries will run on the Prometheus to return the appropriate results, for the queries that are related to Pods, namespace parameter has also to be defined to return pods of the same namespace. The characterization endpoint of the Monitoring API has only one HTTP parameter which is the format of the response, there are two options

a. JSON which will return a response in JSON
b. TOSCA which will download an extended TOSCA YAML file.

In the following examples for the case of the monitoring endpoint you will see a namespace named application, this was an experiment to monitor WordPress (Docker image 4.8-apache[33]) and MySQL (Docker image 5.6[34]) Pods under this namespace. If you do not have another namespace except monitoring namespace in your K3s cluster please replace application namespace with monitoring namespace. Here are some examples:

1. http://0.0.0.0:3000/monitoring?metric=pod_info&namespace=application

   An endpoint that provides information about Pods of the same namespace (pod-name, namespace, pod-IP, pod-name, k3s kind, replica node-name, node-IP). As the second parameter is the namespace of the application it will be easier that each K3s namespace of the use case will have the name of its owner or the name of the application.


```
{"timestamp": 1613918373.292831, "Pod Info Results": [{"pod": "wordpress-
86885f548-44m8z", "pod_ip": null, "namespace": "application", "created_by_kind":
"ReplicaSet", "replica": "1", "node": "giannis", "node_ip": "192.168.1.2"},
{"pod": "mysql-bfc5c9f44-477rv", "pod_ip": "192.168.1.2", "namespace":
"application", "created_by_kind": "ReplicaSet", "replica": "1", "node": "giannis",
"node_ip": "192.168.1.2"}]}
```


2. http://0.0.0.0:3000/monitoring?metric=virtual_metrics&namespace=application

   An endpoint that provides metrics (CPU usage, RAM usage) of Pods that belong on the same namespace, this is done to monitor pods of a specific application. As the second parameter is the namespace of the application it will be easier that each K3s namespace of the use case will have the name of its owner or the name of the application.


```
{"Results": [{"timestamp": 1613918536.169836, "Pod Info Results": [{"pod": "mysql-
bfc5c9f44-477rv", "pod_ip": "192.168.1.2", "namespace": "application",
```


[33] https://hub.docker.com/_/wordpress?tab=tags&page=1&ordering=last_updated
[34] https://hub.docker.com/_/mysql?tab=tags&page=1&ordering=last_updated

```
"created_by_kind": "ReplicaSet", "replica": "1", "node": "giannis", "node_ip":
"192.168.1.2"}, {"pod": "wordpress-86885f548-44m8z", "pod_ip": "10.42.0.46",
"namespace": "application", "created_by_kind": "ReplicaSet", "replica": "1",
"node": "giannis", "node_ip": "192.168.1.2"}]}, {"timestamp": 1613918536.238763,
"Kube Pod Status Phase Results": [{"kube_pod_status_phase": "Running", "pod":
"mysql-bfc5c9f44-477rv", "instance": "10.42.0.42:8443", "namespace":
"application"}, {"kube_pod_status_phase": "Running", "pod": "wordpress-86885f548-
44m8z", "instance": "10.42.0.42:8443", "namespace": "application"}]},
{"timestamp": 1613918536.300128, "Pod CPU Usage Results": [{"node": "giannis",
"pod": "mysql-bfc5c9f44-477rv", "pod_cpu_usage(seconds)": "0.018315272811091667"},
{"node": "giannis", "pod": "wordpress-86885f548-44m8z", "pod_cpu_usage(seconds)":
"0.004569118139741808"}]}, {"timestamp": 1613918536.368414, "Pod Memory Usage
Results": [{"node": "giannis", "pod": "mysql-bfc5c9f44-477rv",
"pod_memory_usage(bytes)": "20627632.01304764"}, {"node": "giannis", "pod":
"wordpress-86885f548-44m8z", "pod_memory_usage(bytes)": "2882807.349989445"}]}]}
```


3. http://0.0.0.0:3000/monitoring?metric=physical_metrics

An endpoint that provides the monitoring metrics for bare metal and VMs of the cluster (CPU usage, RAM usage, disk write and read latencies, filesystem usage, disk size, disk free space disk IO).


```
{"Results": [{"timestamp": 1613918256.884719, "Cpu Usage Results": [{"node":
"giannis", "cpu_usage(percentage)": " 11.39"}, {"node": "raspberrypi",
"cpu_usage(percentage)": " 6.08"}, {"node": "raspberrypi1", "cpu_usage(percentage)": "
6.80"}]}, {"timestamp": 1613918256.948491, "Memory Usage Results": [{"node":
"giannis", "mem_usage(percentage)": " 43.58"}, {"node": "raspberrypi",
"mem_usage(percentage)": " 87.23"}, {"node": "raspberrypi1", "mem_usage(percentage)":
" 75.87"}]}, {"timestamp": 1613918257.013779, "Disk Write Latency Results": [{"node":
"giannis", "device": "sdc", "disk_write_latency(percentage)": " 0.80"}, {"node":
"giannis", "device": "sdd", "disk_write_latency(percentage)": " 1.22"}, {"node":
"raspberrypi", "device": "mmcblk0", "disk_write_latency(percentage)": " 1.20"},
{"node": "raspberrypi", "device": "mmcblk0p2", "disk_write_latency(percentage)": "
1.20"}, {"node": "raspberrypi1", "device": "mmcblk0",
"disk_write_latency(percentage)": " 18.77"}, {"node": "raspberrypi1", "device":
"mmcblk0p2", "disk_write_latency(percentage)": " 18.77"}]}, {"timestamp":
1613918257.075608, "Disk Read Latency Results": [{"node": "giannis", "device": "sdc",
"disk_read_latency(percentage)": " 0.02"}]}, {"timestamp": 1613918257.137085,
"Filesystem Usage Results": [{"node": "giannis", "mountpoint": null, "fstype": null,
"filesystem_usage(percentage)": " 19.49"}, {"node": "raspberrypi", "mountpoint": null,
"fstype": null, "filesystem_usage(percentage)": " 74.97"}, {"node": "raspberrypi1",
"mountpoint": null, "fstype": null, "filesystem_usage(percentage)": " 74.55"}]},
{"timestamp": 1613918257.198618, "Disk Size Results": [{"node": "giannis",
"disk_total_size(bytes)": "2204823101440"}, {"node": "raspberrypi",
"disk_total_size(bytes)": "17658145792"}, {"node": "raspberrypi1",
"disk_total_size(bytes)": "17658145792"}]}, {"timestamp": 1613918257.258936, "Disk
Free Space Results": [{"node": "giannis", "disk_free_space(bytes)": "2170876522496"},
{"node": "raspberrypi", "disk_free_space(bytes)": "6878654464"}, {"node":
"raspberrypi1", "disk_free_space(bytes)": "6956704256"}]}, {"timestamp":
1613918257.318943, "Disk IO Usage Results": [{"node": "giannis",
"disk_io_time_spent(seconds)": "0.007369490285714287"}, {"node": "raspberrypi",
"disk_io_time_spent(seconds)": "0.00036869494949545245"}, {"node": "raspberrypi1",
"disk_io_time_spent(seconds)": "0.08465387434343274"}]}]}
```

4. http://0.0.0.0:3000/characterization?format=json/tosca

This endpoint provides characterization information of every node (bare metal or VM) of the cluster. There are two parameters, the first one is JSON which returns the results in json format and the second one which provides a TOSCA YAML file with the same description.


```
[{"device": {"_id": {"$oid": "60326f8f528bf960d6b36ce3"}, "device_name":
"raspberrypi", "ip": "192.168.1.205", "UUID": "e7cd9caa-7451-11eb-85aa-dca632298c4f",
"RAM(bytes)": 4095737856, "Battery": "None", "CPU": {"Arch": "armv7l", "bits": "32",
"cores": 4}, "GPU": {"GPU_name": "llvmpipe (LLVM 7.0, 128 bits) (0xffffffff)",
"GPU_type": "Intergated graphics processing", "GPU_video_memory(bytes)": 4095737856,
"unified_memory": "no"}, "OS": {"OS_name": "Linux", "OS_version": "4.19.118-v7l+"},
"DISK": [{"device": "/dev/root", "fstype": "ext4", "mountpoint": "/dev/termination-
log"}, {"device": "/dev/root", "fstype": "ext4", "mountpoint": "/etc/resolv.conf"},
{"device": "/dev/root", "fstype": "ext4", "mountpoint": "/etc/hostname"}, {"device":
"/dev/root", "fstype": "ext4", "mountpoint": "/etc/hosts"}], "K3s": {"node_role": ""},
"Region": {"continent": "Europe", "country": "Greece", "city": "Athens"}}}, {"device":
{"_id": {"$oid": "60326fa4528bf960d6b36ce4"}, "device_name": "raspberrypi1", "ip":
"192.168.1.203", "UUID": "e7344550-7451-11eb-9473-dca632299078", "RAM(bytes)":
4095737856, "Battery": "None", "CPU": {"Arch": "armv7l", "bits": "32", "cores": 4},
"GPU": {"GPU_name": "llvmpipe (LLVM 7.0, 128 bits) (0xffffffff)", "GPU_type":
"Intergated graphics processing", "GPU_video_memory(bytes)": 4095737856,
"unified_memory": "no"}, "OS": {"OS_name": "Linux", "OS_version": "4.19.118-v7l+"},
"DISK": [{"device": "/dev/root", "fstype": "ext4", "mountpoint": "/dev/termination-
log"}, {"device": "/dev/root", "fstype": "ext4", "mountpoint": "/etc/resolv.conf"},
{"device": "/dev/root", "fstype": "ext4", "mountpoint": "/etc/hostname"}, {"device":
"/dev/root", "fstype": "ext4", "mountpoint": "/etc/hosts"}, {"device": "/dev/root",
"fstype": "ext4", "mountpoint": "/dev/termination-log"}, {"device": "/dev/root",
"fstype": "ext4", "mountpoint": "/etc/resolv.conf"}, {"device": "/dev/root", "fstype":
"ext4", "mountpoint": "/etc/hostname"}, {"device": "/dev/root", "fstype":
"ext4", "mountpoint": "/etc/hosts"}], "K3s": {"node_role": ""}, "Region": {"continent":
"Europe", "country": "Greece", "city": "Athens"}}}, {"device": {"_id": {"$oid":
"60326fa4528bf960d6b36ce5"}, "device_name": "giannis", "ip": "192.168.1.2", "UUID":
"ea831f38-7451-11eb-8bc7-fcaa149d94de", "RAM(bytes)": 8396820480, "Battery": "None",
"CPU": {"Arch": "x86_64", "bits": "64", "cores": 6}, "GPU": {"GPU_name": "AMD BONAIRE
(DRM 2.50.0, 4.15.0-135-generic, LLVM 7.0.1) (0x665f)", "GPU_type": "Dedicated
graphics processing", "GPU_video_memory(bytes)": 2147483648,
"GPU_total_available_memory(bytes)": 4289724416, "unified_memory": "no"}, "OS":
{"OS_name": "Linux", "OS_version": "4.15.0-135-generic"}, "DISK": [{"device":
"/dev/root", "fstype": "ext4", "mountpoint": "/dev/termination-log"}, {"device":
"/dev/root", "fstype": "ext4", "mountpoint": "/etc/resolv.conf"}, {"device":
"/dev/root", "fstype": "ext4", "mountpoint": "/etc/hostname"}, {"device": "/dev/root",
"fstype": "ext4", "mountpoint": "/etc/hosts"}, {"device": "/dev/root", "fstype":
"ext4", "mountpoint": "/dev/termination-log"}, {"device": "/dev/root", "fstype":
"ext4", "mountpoint": "/etc/resolv.conf"}, {"device": "/dev/root", "fstype": "ext4",
"mountpoint": "/etc/hostname"}, {"device": "/dev/root", "fstype": "ext4",
"mountpoint": "/etc/hosts"}, {"device": "/dev/sdc1", "fstype": "ext4", "mountpoint":
"/dev/termination-log"}, {"device": "/dev/sdc1", "fstype": "ext4", "mountpoint":
"/etc/resolv.conf"}, {"device": "/dev/sdc1", "fstype": "ext4", "mountpoint":
"/etc/hostname"}, {"device": "/dev/sdc1", "fstype": "ext4", "mountpoint":
"/etc/hosts"}], "K3s": {"node_role": "control-plane,master"}, "Region": {"continent":
"Europe", "country": "Greece", "city": "Athens"}}}]
```

5. http://0.0.0.0:3000/monitoring?namespace=all

This endpoint provides all the namespace of the clusters along with their CPU and memory usage. Each endpoint could be an application, so this is a way to monitor an application as a whole.


```
{"Results": [{"timestamp": 1613918683.353328, "Namespace CPU Results":
[{"namespace": "monitoring", "cpu_usage(seconds)": "0.1492824091256434"},
{"namespace": "kube-system", "cpu_usage(seconds)": "0.012014744984338889"},
{"namespace": "application", "cpu_usage(seconds)": "0.0007409287284330483"}]},
{"timestamp": 1613918683.416821, "Namespace Memory Results": [{"namespace": "kube-
system", "memory_usage(bytes)": "232685568"}, {"namespace": "monitoring",
"memory_usage(bytes)": "1953517568"}, {"namespace": "application",
"memory_usage(bytes)": "1104633856"}]}]}
```


For the case of Grafana endpoints the IP configured and once more is the IP of the K3s master node. If the master node of your K3s cluster has 192.168.1.2 as an IP, Ingress of Grafana configures a prefix and a suffix so the whole address would be https://grafana.192.168.1.2.nip.io. By accessing this address, you will be redirected to a login page which will ask for the credentials. Right now only the admin credentials are available which are username *admin* and password *admin*. In the next steps we definitely have to create a stronger password for the administrator account, but also we have to create a group of users with different permissions on Grafana dashboards.

All the dashboards are in the https://grafana.192.168.1.2.nip.io/dashboards inside the default folder (Figure 17).

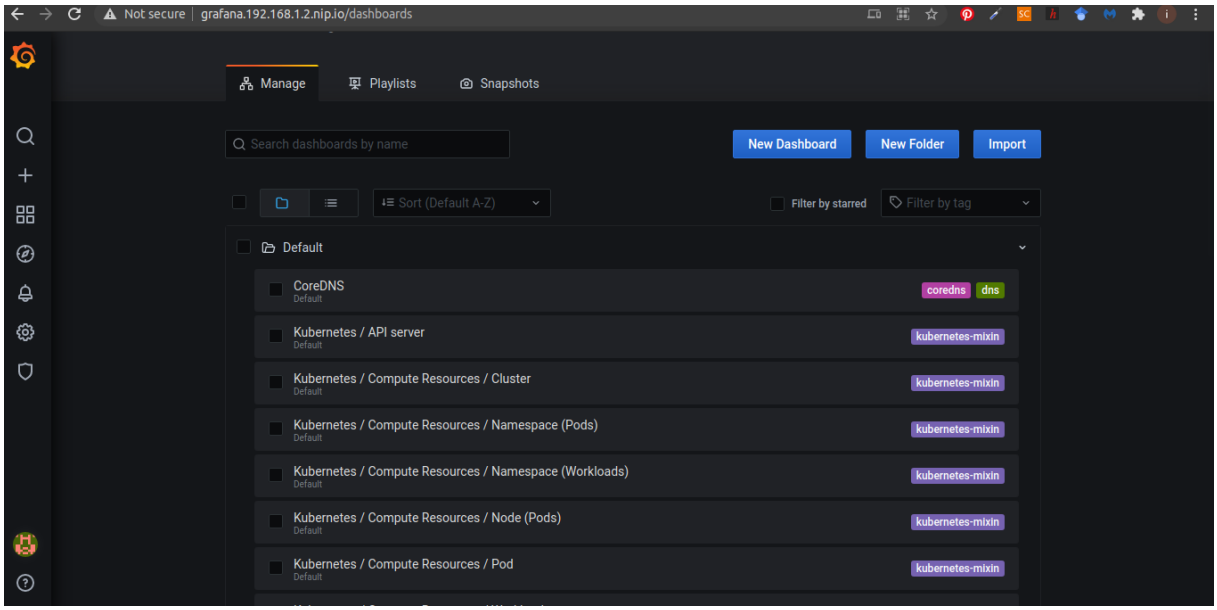

Figure 17: All Dashboards

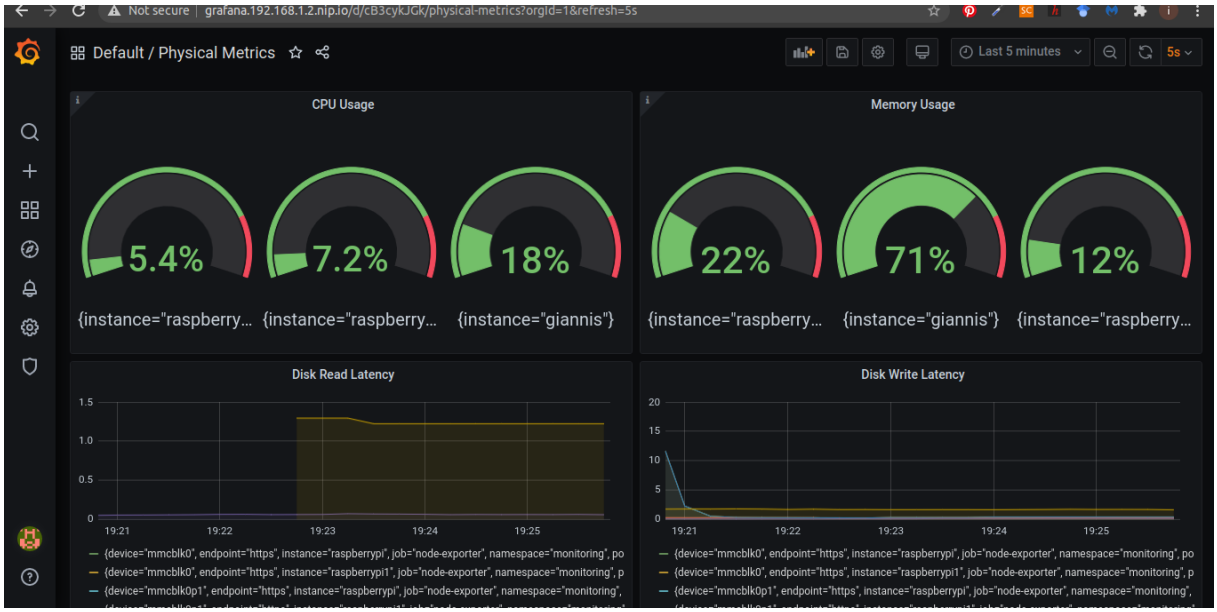

Figure 18: Physical Metrics Dashboard Part 1

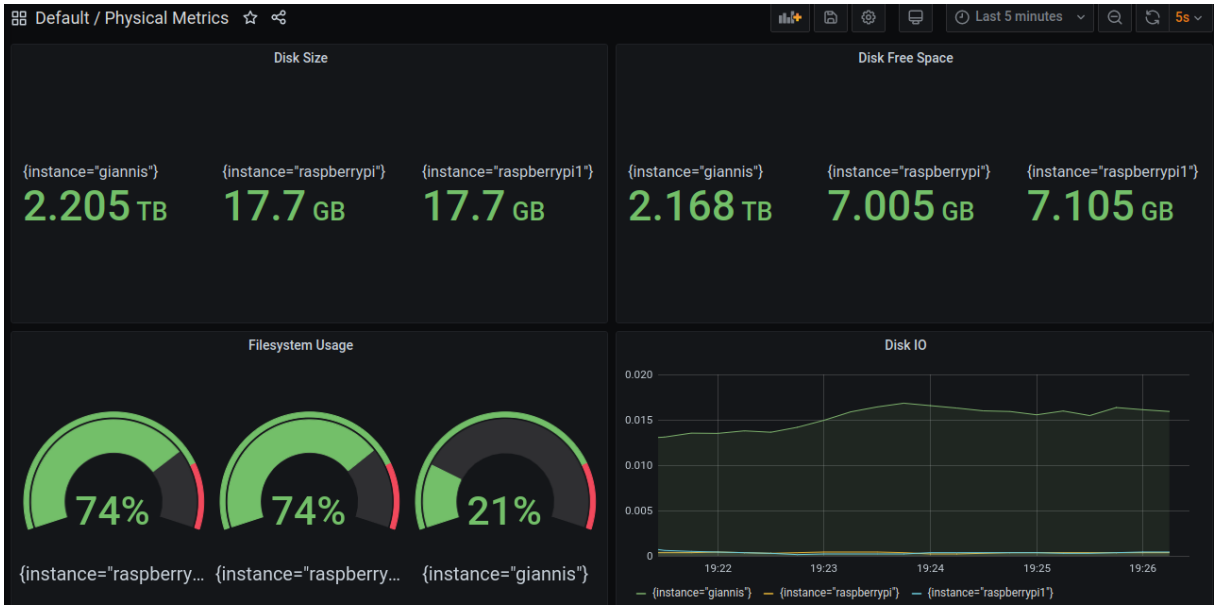

Figure 19: Physical Metrics Dashboard Part 2

The Physical Metrics dashboard represent the same values that we can find in the respective endpoint of the Monitoring API (Figure 18, Figure 19).

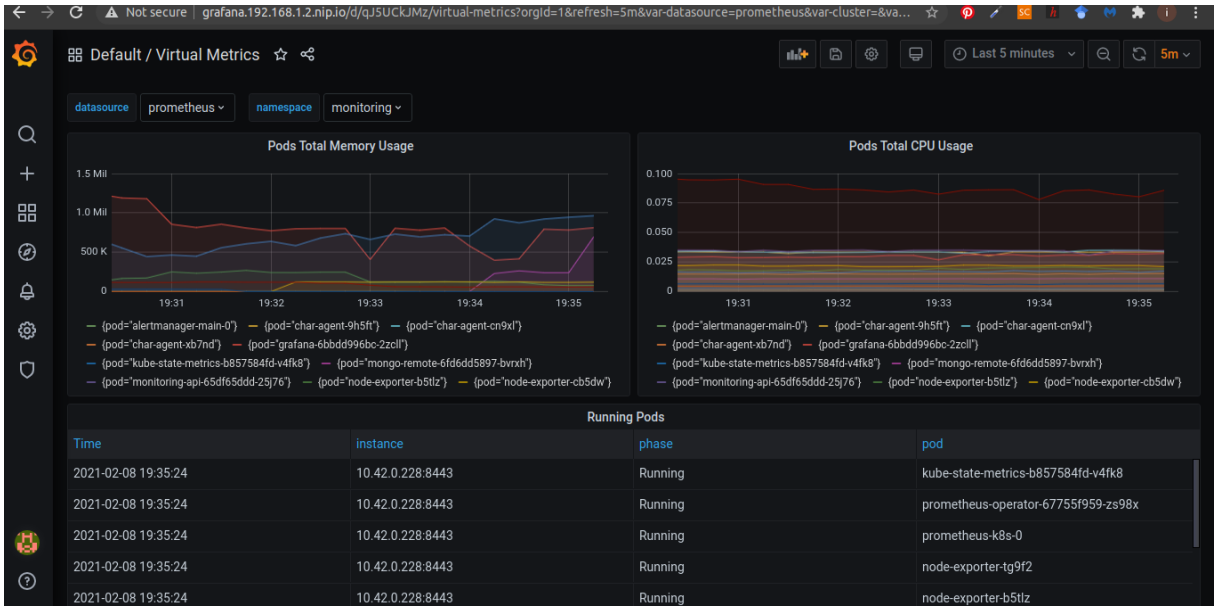

Figure 20: Virtual Metrics Dashboard Part 1

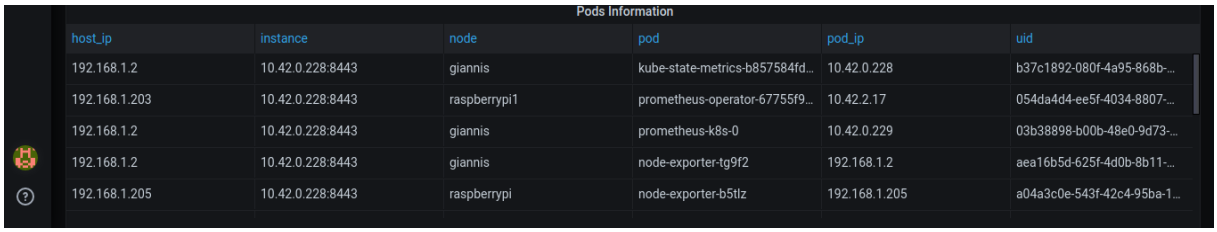

Figure 21: Virtual Metrics Dashboard Part 2

The Virtual metrics dashboard (Figure 20, Figure 21) presents the same values of the virtual metrics in the Monitoring API. There is a parameter on the dashboard named namespace, which basically gives you all the namespaces of the cluster, by selecting one of the namespaces all the graphs and tables will return information about pods that belong to the same namespace. If there is no additional namespace in your K3s cluster, please select monitoring.

3.5 Licensing information

Most of the YAML configuration files for the monitoring component were produced from Cluster Monitoring stack for ARM / X86-64 open-source component which is under MIT license. This license permits commercial use, modification, distribution and private use. Then these files were modified to add some more configurations to them.

The rest of the open-source components that are used as Pods (PrometheusARM, Node exporter, Alert Manager, Prometheus Operator, Prometheus Adapter, kube-state-metrics, Grafana) are under Apache 2.0 license. In addition, the open-source components that are used to configure the monitoring component (configmap-reload and kube-rbac-proxy). This license permits commercial use, modification, distribution, patent use and private use.

As characterization agent and Monitoring API are custom solutions that were created for ACCORDION purposes we think that they will use Apache 2.0 license. The libraries that have been used for developing characterization agent and Monitoring API are shown in the following table along with their licenses.

Table 2: Libraries & Licenses for Monitoring components

License	Library
MIT[35]	oyaml[36], py-cpuinfo[37], mesa-utils[38]
BSD 2-Clause License[39]	cryptoyaml3[40]
Apache 2.0[41]	pymongo[42]
BSD License (BSD-3-Clause)[43]	flask[44], flask_restful[45], psutil[46]
Public Domain (UNLICENSE)[47]	ipify[48]

[35] https://opensource.org/licenses/MIT
[36] https://pypi.org/project/oyaml/
[37] https://pypi.org/project/py-cpuinfo/
[38] https://wiki.debian.org/Mesa
[39] https://opensource.org/licenses/BSD-2-Clause
[40] https://pypi.org/project/cryptoyaml3/
[41] https://www.apache.org/licenses/LICENSE-2.0
[42] https://pypi.org/project/pymongo/
[43] https://opensource.org/licenses/BSD-3-Clause
[44] https://pypi.org/project/Flask/
[45] https://pypi.org/project/Flask-RESTful/
[46] https://pypi.org/project/psutil/
[47] https://unlicense.org/
[48] https://pypi.org/project/ipify/

4 Resource Indexing and Discovery

4.1 Component description

Computational resources in ACCORDION are defined by their static and dynamic characteristics. The dynamic characteristics depend on the workload imposed on the ACCORDION nodes by the applications and are subject to continuous changing. In order to provide an effective resource orchestration and allocation, the ACCORDION system has to rely on the up-to-date state of the characteristics of resources. The role of the Resource Indexing & Discovery (RID) component is then twofold. First, the RID component has to keep updated on the state of the available computational resources distributed among the nodes in the system. Second, the RID component provides the functionality for effective information retrieval based on the predefined characteristics. These characteristics can be static (amount of CPU/RAM/DISK available) or dynamic (current load of the resource). The main source of information about the state of the nodes load is the Resource monitoring & characterization component. The RID component periodically pulls updates on resource characteristics from the monitoring component. For now, the main component that is aimed to use RID is intelligent Orchestrator. Nevertheless, any component in the system can query the required resources via RID component. A more detailed view of the RID components was presented in Deliverable 3.1.

The current implementation of the RID component consists of a Python application, with an instance of the RID running in each minicloud. The RID component contains a client that periodically pulls the data from the monitoring component of the same minicloud, and (in the current implementation) stores this data in its local storage. The long-term plan is that the local storage does not necessarily keep only local data, or that a minicloud information is not necessarily stored in the local storage. In fact, advanced distributed indexing techniques are one of the ultimate objectives of the RID, such as to distribute data among all their instances, in order to scale retrieval of resource information. The RID accepts queries from other ACCORDION components by exposing a REST server. Currently, multi-attribute range queries are supported. The RID is containerized into a Docker image, which has to be deployed inside each of the ACCORDION miniclouds. For testing purposes, the RID exposes also a custom web interface, which makes it possible to submit queries and observe the results.

4.2 Package information

The current version of the RID can be download as a GIT project with the following command:

```
$ git clone -b v1.0.0 https://gitlab.com/accordion-project/wp3/resource-indexing-discovery.git
```

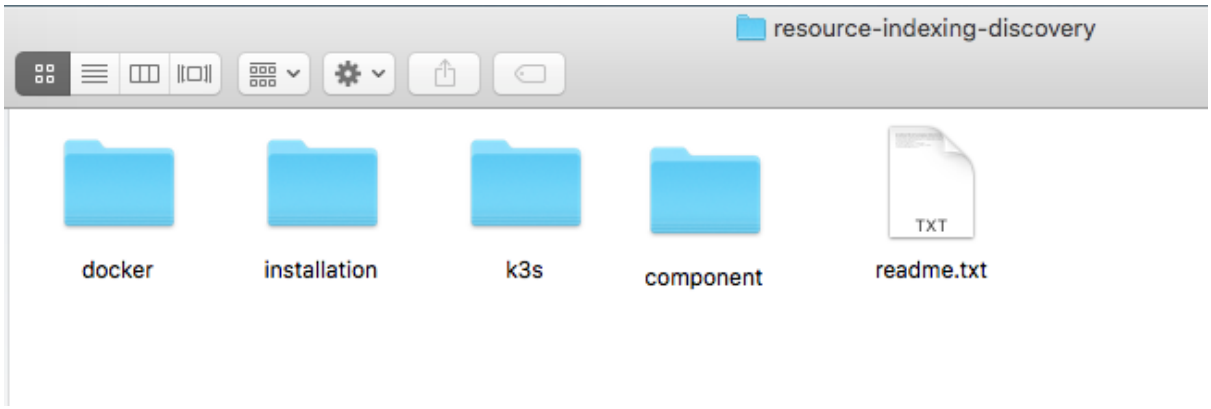

Figure 22: Resource Indexing and Discovery file structure

The project contains four folders, as indicated in Figure 22:

- `docker` - This folder contains the Dockerfile to build the docker image. The docker file copies all necessary code from the src folder to the Docker image.
- `Installation` - This folder contains shell scripts to install the component. There are two installation scripts, one that builds a docker image using a Dockerfile and runs it by using docker (`build-run-docker.sh`), and another one that builds the same docker image and runs it on K3s (`build-docker-run-k3s.sh`).
- `k3s` - This folder contains the YAML file (`rid-deployment.yaml`) that describes the K3s deployment, starting from the built docker image.
- `component` - This folder has two subfolders.
  - The `src` subfolder contains the source code of the component, which is two Python files. The file `rid_server.py` contains the implementation of the REST and web servers. The file `rid_data.py` contains the implementation of local storage and query management. The `src` subfolder also has several data files that serve to populate the local storage in case the monitoring service is down or not reachable. These files are meant to be used only in the test phase. In the following versions of the components, the data will be retrieved automatically from the monitoring component.
  - The `web` subfolder contains an HTML file (`index.html`) that is the implementation of a simple testing web interface.

Finally, the file `readme.txt` describes the purpose and the installation procedure of the component.

4.3 Installation instructions

The installation procedure assumes that the content of the git project (see above) is currently available on the target machine. The installation folder (as described above) contains two shell scripts that build and run a docker image in the target machine. The scripts assume a UNIX system to be running. Currently, it has been tested on a Ubuntu 18.04 Linux distribution and a macOS High Sierra 10.13.6 (only the Docker installation).

To install and run the RID using only Docker, it is necessary only to execute:

```
$ ./installation/build-run-docker.sh
```

Once finished the REST server is up and running, as shown in Figure 23.

```
Successfully built b8ad1d1d2a88
Successfully tagged rid:v1
 * Serving Flask app "rid_server" (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: on
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 207-558-976
```

Figure 23: Output of the build-run-docker.sh script

To install and deploy using K3s, it is necessary to execute:

```
$ ./installation/build-docker-run-k3s.sh
```

4.4 User manual

The RID exposes a REST interface on port 5000 that accepts a single multi-attribute range query.

```
POST http://0.0.0.0:5000/query
```

The query must be in the JSON format with the given schema:

```
{
  "type": array,
  "items": -{
       "type": object,
       "required": -[],
       "properties": -{
              "name": -{
                     "required": true,
                     "type": string
              },
              "value_min": -{
                     "required": true,
                     "type": number
              },
              "value_max": -{
                     "required": true,
                     "type": number
              }
       }
  }
}
```

Essentially, the query is an array of objects. Each object has a name and a minimum and maximum numerical value. For example, a correct query would be:

```
[{"name":"device.RAM(bytes)", "value_min":8396832768, "value_max":8396832768}]
```

The response will provide information on the resources that satisfy the criteria expressed in the query. The JSON schema of the answer looks like the following:

```
{
  "type": "object",
  "required": [],
  "properties": {
        "device._id.$oid": {
              "type": "string"
        },
        "device.device_name": {
              "type": "string"
        },
        "device.ip": {
              "type": "string"
        },
.....
        "device.Region.country": {
              "type": "string"
        },
        "device.Region.city": {
              "type": "string"
        }
  }
}
```

For example, here is shown the result of the example query (visualized with the custom web interface of the RID).

4.5 Licensing information

Currently, the RID component does not use any third-party software. Therefore, licensing is the one adopted by the ACCORDION project.

5 ACCORDION Edge Storage Component (ACES)

5.1 Component description

The ACCORDION Edge Storage Component (ACES for short) is responsible for providing optimized edge storage services to the ACCORDION framework and its hosted applications. These services include data storage, retrieval and migration tasks, security and privacy protection capabilities, QoS and QoE violation prevention and mitigation as well as other data-related services that serve the runtime requirements of ACCORDION and the hosted applications.

The component is based on the MinIO[49], Prometheus[50] and Kubernetes (K3s[51]) technologies, combining and optimizing them in order to better serve the needs of ACCORDION. The first version of the component is designed to run on nodes with public IPs, allowing the direct communication of nodes with each other, using the K3s framework as an orchestrator, allowing us to easily deploy and configure the edge storage services.

Prometheus is used to gather monitoring data about the real-time performance of the nodes and the component as a whole in order for us to be able to analyze the behaviour of different applications and optimize the cluster architecture, the options and the data distribution. Finally, MinIO is a highly scalable and decentralized platform, allowing us to deploy it freely on usable nodes, which we have labelled as storage workers. Adding Goofys[52] on top of MinIO we have the capability of using the edge storage component as a file system folder which is useful for applications that we cannot or do not want to integrate with the Restful API of MinIO. A figure depicting the full architecture and the interactions between the sub-components can be found in deliverable D3.1.

5.2 Package information

ACES is a package including Kubernetes deployment files in YAML format, installation scripts in bash script format and a configuration file in JSON format that contains all options needed to configure the component.

All files of the package will be available on the official ACCORDION Gitlab page[53] and can be obtained with the following command:

```
$ git clone -b v1.0.0 https://gitlab.com/accordion-project/wp3/edge-storage-component.git
```

In detail, we have one YAML file called `acesServerDeployment.yaml` which is the Kubernetes deployment file for the storage server. This file will install all necessary services, authentication keys, roles and images on the Kubernetes cluster, reading information from the configuration file. It will use the Kubernetes architecture, deploying most services on the Kubernetes master. Of course, the actual MinIO instances that store the data will be deployed on the nodes labelled as “storage-worker”. The second yaml file is called `acesClientDeployment.yaml`

[49] https://min.io/
[50] https://prometheus.io/
[51] https://k3s.io/
[52] https://github.com/kahing/goofys
[53] https://gitlab.com/accordion-project/wp3/edge-storage-component

and it will allow nodes labelled as “storage-client” to use ACES as a file system folder by deploying the Goofys framework on them and mounting the MinIO storage server.

The bash scripts are again two, `acesServerDeploy.sh` that configures and deploys the `acesServerDeployment.yaml` on the Kubernetes master and `acesClientDeploy.sh` that configures and deploys the `acesClientDeployment.yaml` on the client nodes. These scripts are just applying the options selected in the configuration file to the YAML files and then run the necessary commands to deploy the YAML files on the Kubernetes cluster.

Table 3: List of package files for Edge storage component

Filename	Description
acesServerDeployment.yaml	Kubernetes deployment file for ACES master
acesClientDeployment.yaml	Kubernetes deployment file for ACES clients
acesServerDeploy.sh	Bash script for deploying the ACES servers
acesClientDeploy.sh	Bash script for deploying the ACES clients
acesConfig.conf	JSON file containing the configuration options for ACES

5.3 Installation instructions

The installation of the component is separated into three steps, the Kubernetes configuration, the ACES configuration and the ACES installation.

Kubernetes configuration contains all the required steps that need to be taken on the Kubernetes platform as a preparation for the installation of ACES. For the first version of ACES this step contains two actions:

- Label at least one node as “storage-worker” and one as “storage-master”.
- Create a folder on the storage-worker nodes and make sure that the Kubernetes user has full permissions on it and that it has enough free space to act as a data storage device.
- If we plan to deploy the `acesClientDeployment.yaml` file, we also need to create a folder with the necessary permissions on the client node and label the node as “storage-client”.

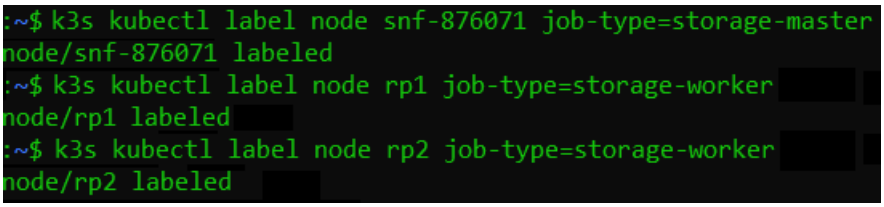

Figure 24: Labeling process for a cluster of two workers and one master.

ACES configuration is the process of setting the required and optional settings present in the configuration file (`acesConfig.conf`) in JSON format. The options are listed in the file for reference while comments are available to clearly define what each option does and how it should be filled. The administrator of the edge storage platform needs to fill them in order for the system to be installed in an optimized fashion on the cluster.

ACES installation is just running the bash scripts with elevated privileges in order to apply the configurations to the YAML files and then deploy them on the local Kubernetes cluster. It is important to remember that the scripts need to be executed on the Kubernetes master with the YAML and configuration files in the same folder.

```
                    :~$ ./acesServerDeploy.sh
namespace/minio created
secret/minio-keys 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
```

Figure 25: Output from acesServerDeploy.sh: all commands were successful and the K8S items were created

```
                    :~$ k3s kubectl get pod --namespace minio
NAME                        READY   STATUS    RESTARTS   AGE
svclb-minio-service-p2mcn   1/1     Running   0          71s
svclb-minio-service-7j47l   1/1     Running   0          71s
svclb-minio-service-zrxbv   1/1     Running   0          71s
minio-0                     1/1     Running   0          72s
minio-1                     1/1     Running   0          30s
```

Figure 26: An example cluster running on two storage worker nodes (Raspberry PIs) having one service for each worker and one for the master (access point)

5.4 User manual

We have three ways to use ACES, the first way is through the MinIO Web GUI which is clearly described in detail on the official MinIO documentation[54]. A sample MinIO storage deployment can be seen in Figure 27.

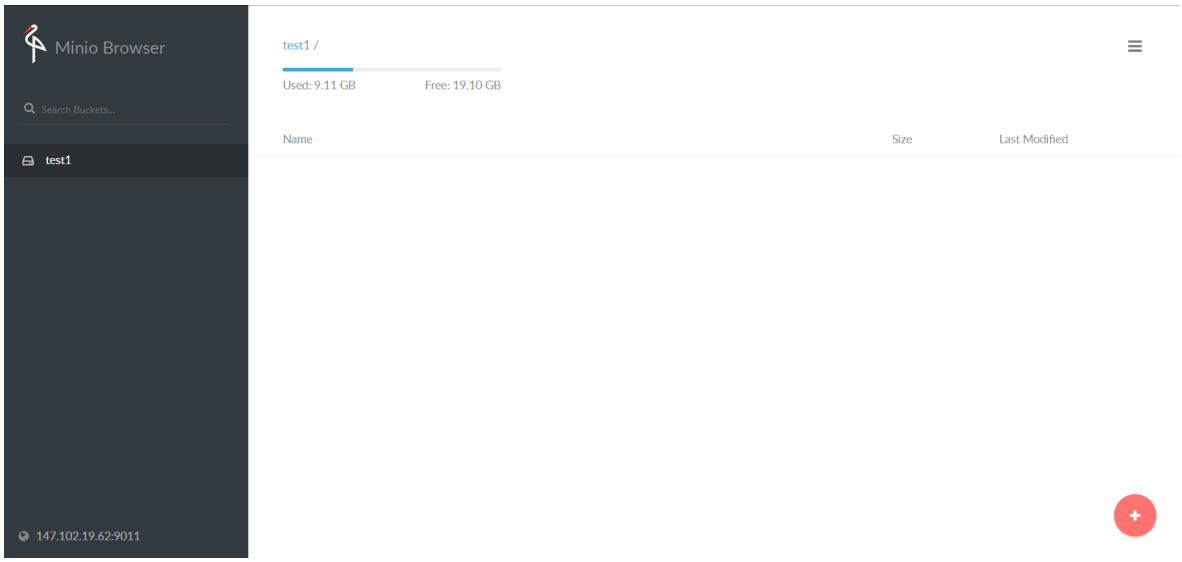

Figure 27: The MinIO web-based interface.

The second way is through the MinIO client which is a command line tool that is also documented in detail on the official MinIO website[55]. A connection to a remote host can be seen as an example in Figure 28.

[54] https://docs.min.io/docs/minio-quickstart-guide.html
[55] https://docs.min.io/docs/minio-client-complete-guide.html

```
/ # mc alias set ACES_1 "http://147.102.19.62:9011" "755bcc4b06f681863718cd5d2d548a64" "404c32ad042b310db3184303cf2bcc69"
mc: Configuration written to `/root/.mc/config.json`. Please update your access credentials.
mc: Successfully created `/root/.mc/share`.
mc: Initialized share uploads `/root/.mc/share/uploads.json` file.
mc: Initialized share downloads `/root/.mc/share/downloads.json` file.
Added `ACES_1` successfully.
/ # mc ls ACES_1
[2021-02-16 13:19:14 UTC]     0B test1/
```

Figure 28: An example connection with the command line MinIO client.

The third way is through the Goofys framework which mounts the remote storage bucket as a local file system folder, creating a direct connection. This way enables users to perform standard file operations like create, move, delete, rename and update on their local filesystem while Goofys performs the related operations to the remote storage in the background. An example file creation can be seen in Figure 29.

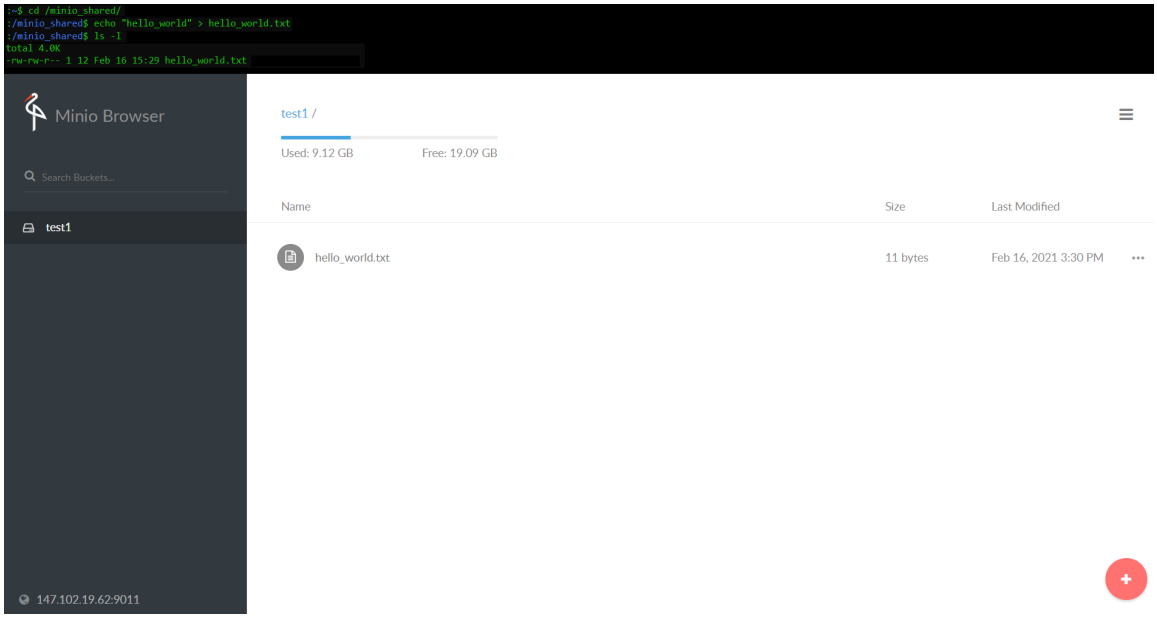

Figure 29: An example file creation with Goofys, we create the file in the shared folder and it appears on the web interface

5.5 Licensing information

This component, including all originally created source files, scripts and other resources will be published as free software under the terms of the GNU General Public License version 3 or later, as published by the Free Software Foundation.

MinIO is provided under GNU Affero General Public License version 3 licensing which enables us to use it as an open-source component providing that we also use a GNU public License.

Prometheus and K3s are protected under Apache License which gives us full usability of their open-source components.

6 Unikraft

6.1 Component Description

Unikraft[56] is a comprehensive toolchain and library operating system which builds highly specialized unikernels, software bundles that consist of a target application along with just the operating system primitives and libraries features it needs to run. Unikraft breaks the status quo of building unikernels manually, providing an automated toolchain that builds tailored unikernels that meet your (and your application's) needs. As such, it fits perfectly well, and plays a key role, in ACCORDION's lightweight virtualization target, where we're targeting and investigating how to enhance the project's use cases and application with light-weight, Unikraft-built unikernel images.

To make it much more user-friendly, Unikraft comes with a companion command-line tool called kraft which makes it easier to build and run Unikraft images. As future work, we are working towards Kubernetes integration, such that users of the popular framework can use much more efficient (Unikraft-built) images *transparently*, without having to change any Kubernetes configuration settings. We will report on the progress of this activity in future deliverables and will concentrate on kraft in the rest of this section of this deliverable.

6.2 Package Information

The kraft application is distributed as a public, open-source repo at https://github.com/unikraft/kraft. We are also in the process of providing Debian packages and Docker images with kraft already in them; we are also integrating this with Unikraft's CI/CD system (based on Concourse) so that it will regularly, and automatically, publish new versions of the Debian and Docker distribution mechanisms.

6.3 Installation Instructions

The present instructions are adapted from https://github.com/unikraft/kraft.

The kraft tool and Unikraft build system have a number of package requirements; please run the following command (on apt-get-based systems) to install the requirements:

```
$ apt-get install -y --no-install-recommends build-essential libncurses-dev libyaml-dev flex \
        git wget socat bison unzip uuid-runtime;
```

To install kraft simply run:

```
$ pip3 install git+https://github.com/unikraft/kraft.git
```

You can then type `kraft` to see its help menu (see Figure 30).

[56] http://www.unikraft.org/

6.3.5 Building an Application

The simplest way to get the sources for, build and run an application is by running the following commands:

```
$ kraft list
$ kraft up -p PLATFORM -m ARCHITECTURE APP
```

At present, Unikraft and kraft support the following applications:

- C "hello world" (helloworld[57]);
- C "http reply" (httpreply[58]);
- C++ "hello world" (helloworld-cpp[59]);
- Golang (helloworld-go[60]);
- Python 3 (python3[61]);
- Micropython (micropython[62]);
- Ruby (ruby[63]);
- Lua (lua[64]);
- Click Modular Router (click[65]);
- JavaScript (Duktape) (duktape[66]);
- Web Assembly Micro Runtime (WAMR) (wamr[67]);
- Redis (redis[68]);
- Nginx (nginx[69]);
- SQLite (sqlite[70]);

6.4 User Manual

The present user manual is adapted from http://unikraft.neclab.eu/kraft.html.

Once kraft it installed you can begin by initializing a new unikernel repository using the command

[57] https://github.com/unikraft/app-helloworld
[58] https://github.com/unikraft/app-httpreply
[59] https://github.com/unikraft/app-helloworld-cpp
[60] https://github.com/unikraft/app-helloworld-go
[61] https://github.com/unikraft/app-python3
[62] https://github.com/unikraft/app-micropython
[63] https://github.com/unikraft/app-ruby
[64] https://github.com/unikraft/app-lua
[65] https://github.com/unikraft/app-click
[66] https://github.com/unikraft/app-duktape
[67] https://github.com/unikraft/app-wamr
[68] https://github.com/unikraft/app-redis
[69] https://github.com/unikraft/app-nginx
[70] https://github.com/unikraft/app-sqlite

```
$ kraft init
```

Then, as an example, you can build a Python 3 unikernel application by running the following:

```
$ kraft list
$ mkdir ~/my-first-unikernel && cd ~/my-first-unikernel
$ kraft up -a helloworld -m x86_64 -p kvm
```

Note that, if this is the first time you are running kraft, you will be prompted to run an update that will download Unikraft core and additional library pool sources. These sources are saved to the directory indicated by the environment variable UK_WORKDIR, which defaults to `~/.unikraft`.

With a newly initialized unikernel application, the `./my-first-unikernel` directory will be populated with a `deps.json` file that contains references to the relevant library dependencies which are required to build a unikernel with support for Python 3. This file is used by kraft to configure and build against certain Unikraft library versions. In addition to this file, a new `.config` file will also be placed into the directory. This file is used by Unikraft's build system to switch on or off features depending on your application's use case.

The unikernel application must now be configured against the Unikraft build system so that it can resolve any additional requirements:

```
$ kraft configure ./my-first-unikernel
```

Note that this step can be made more interactive by launching into Unikraft's Kconfig configuration system. Launch an ncurses[71] window in your terminal with the command

```
$ kraft configure --menuconfig
```

The configuration step used in kraft will perform the necessary checks pertaining to compatibility and availability of source code and will populate your application directory with new files and folders, including:

- `kraft.yaml` – This file holds information about which version of the Unikraft core and additional libraries to use, which architectures and platforms to target and which network bridges and volumes to mount during runtime,
- `Makefile.uk` – A Kconfig target file you can use to create compile-time toggles for your application,
- `build/` – All build artefacts are placed in this directory including intermediate object files and unikernel images,
- `.config` – The selection of options for architecture, platform, libraries and your application (specified in `Makefile.uk`) to use with Unikraft.

When your unikernel has been configured to your needs, you can build the unikernel to all relevant architectures and platforms using:

```
$ kraft build ./my-first-unikernel
```

This step will begin the build process. All artefacts created during this step will be located under `./my-first-unikernel/build`.

[71] https://en.wikipedia.org/wiki/Ncurses

6.4.6 Overview of commands

The overview of the commands can be simply obtained by the tool's help menu:

```
Usage: kraft [OPTIONS] COMMAND [ARGS]...

Options:
  --version                       Show the version and exit.
  -C, --ignore-git-checkout-errors
                                  Ignore checkout errors.
  -X, --dont-checkout             Do not checkout repositories.
  -v, --verbose                   Enables verbose mode.
  -h, --help                      Show this message and exit.

Commands:
  build      Build the application.
  clean      Clean the application.
  configure  Configure the application.
  init       Initialize a new unikraft application.
  list       List architectures, platforms, libraries or applications.
  run        Run the application.
  up         Configure, build and run an application.

Influential Environmental Variables:
  UK_WORKDIR The working directory for all Unikraft
             source code [default: ~/.unikraft]
  UK_ROOT    The directory for Unikraft's core source
             code [default: $UK_WORKDIR/unikraft]
  UK_LIBS    The directory of all the external Unikraft
             libraries [default: $UK_WORKDIR/libs]
  UK_APPS    The directory of all the template applications
             [default: $UK_WORKDIR/apps]
  KRAFTCONF  The location of kraft's preferences file
             [default: ~/.kraftrc]

Help:
  For help using this tool, please open an issue on Github:
  https://github.com/unikraft/kraft
```

Figure 30: Kraft help menu

The easiest way to configure, build and run a Unikraft image is via the `kraft up` command. The syntax of this command is shown in the following Figure 31.

```
Usage: kraft up [OPTIONS] NAME

  Configures, builds and runs an application for a selected architecture and
  platform.

Options:
  -p, --plat [linuxu|kvm|xen]    Target platform.
  -m, --arch [x86_64|arm|arm64]  Target architecture.
  -i, --initrd TEXT              Provide an init ramdisk.
  -X, --background               Run in background.
  -P, --paused                   Run the application in paused state.
  -g, --gdb INTEGER              Run a GDB server for the guest on specified
                                 port.
  -d, --dbg                      Use unstriped unikernel.
  -n, --virtio-nic TEXT          Attach a NAT-ed virtio-NIC to the guest.
  -b, --bridge TEXT              Attach a NAT-ed virtio-NIC an existing
                                 bridge.
  -V, --interface TEXT           Assign host device interface directly as
                                 virtio-NIC to the guest.
  -D, --dry-run                  Perform a dry run.
  -M, --memory INTEGER           Assign MB memory to the guest.
  -s, --cpu-sockets INTEGER      Number of guest CPU sockets.
  -c, --cpu-cores INTEGER        Number of guest cores per socket.
  -F, --force                    Overwrite any existing files in current
                                 working directory.
  -j, --fast                     Use all CPU cores to build the application.
  --with-dnsmasq                 Start a Dnsmasq server.
  --ip-range TEXT                Set the IP range for Dnsmasq.
  --ip-netmask TEXT              Set the netmask for Dnsmasq.
  --ip-lease-time TEXT           Set the IP lease time for Dnsmasq.
  -h, --help                     Show this message and exit.
```

Figure 31: Kraft up help menu

For more advanced command and additional information please refer to the kraft user manual at http://docs.unikraft.org/kraft.html.

6.5 Licensing Information

The kraft and Unikraft software systems are released under a commercially-friendly BSD license. The full license can be accessed online, for both kraft[72] and Unikraft[73].

[72] https://github.com/unikraft/kraft/blob/staging/COPYING.md
[73] https://github.com/unikraft/unikraft/blob/staging/COPYING.md

7 Conclusions

This document is the accompanying report documenting the software that is released as part of ACCORDION Deliverable D3.2 and explains how to install and use it. The document provides a technical description and licensing information for each delivered software component, along with installation and user guides. Most sections include screenshots showing how the related component appears during its installation and use.

The software components described make up the first version of an “Edge minicloud”. The implemented minicloud model can include only resources located in a single site and typically owned by a single provider. There should not be any firewall among the minicloud resources, even if the main firewall is expected to exist, shielding the provider’s network from the internet. The upcoming integration phase will analyze the traffic that will be expected to cross the provider’s firewall and will define which ports should be left open in its configuration. Research in the next year will be done to explore the possibility of implementing a minicloud across multiple sites.

After the integration phase, the ACCORDION minicloud will be tested by running real applications on it, typically those from the WP6 use cases. This validation phase will evaluate if the provided functionalities and performance are suitable to satisfy the requirements and will indicate which improvements are needed. Based on the feedbacks, the needed improvements will be done to the next minicloud version and more minicloud components may be added to more completely match the intended architecture and satisfy the users’ expectations.