Tailored IoT & BigData Sandboxes and Testbeds for Smart, Autonomous and Personalized Services in the European Finance and Insurance Services Ecosystem

Infinitech

D6.11 – Sandboxes for FinTech and InsuranceTech Innovators - II

Revision Number	1.0
Task Reference	T6.5
Lead Beneficiary	ENG
Responsible	Domenico Messina - Susanna Bonura
Partners	Participating partners in Task according to DOA
Deliverable Type	Report (R)
Dissemination Level	Public (PU)
Due Date	2021-09-30
Delivered Date	2021-10-01
Internal Reviewers	BOUN, CTAG
Quality Assurance	CCA
Acceptance	WP Leader Accepted and Coordinator Accepted
EC Project Officer	Pierre-Paul Sondag
Programme	HORIZON 2020 - ICT-11-2018
	This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no 856632

Contributing Partners

Partner Acronym	Role[1]	Author(s)[2]
ENG	Lead Beneficiary	Domenico Messina – Susanna Bonura
HPE	Contributor	
NOVA	Contributor	
JRC	Contributor	
ATOS	Contributor	
SILO	Contributor	
WEA	Contributor	
GEN	Contributor	
GFT	Contributor	
BOUN	Internal Reviewer	
CTAG	Internal Reviewer	
CCA	Quality Assurance	

[1] Lead Beneficiary, Contributor, Internal Reviewer, Quality Assurance

[2] Can be left void

Revision History

Version	Date	Partner(s)	Description
0.1	2021-07-20	ENG and contrib. partners	ToC Version
0.2	2021-08-30	ENG, WEA, LXS	Pilot #13 contribution merged
0.3	2021-09-02	ENG, SILO, UBI, ISPRINT	Pilot #12 contribution merged
0.4	2021-09-09	ENG, JRC, INNOV	Pilot #2 contribution merged
0.5	2021-09-15	ENG, ATOS, GEN, AGRO	Pilot #11 and Pilot#14 contribution merged
0.6	2021-09-16	HPE, ENG	Contribution on design and tools (HPE) merged
0.7	2021-09-27	ENG	First Version ready for Internal Peer Review
0.8	2021-09-29	BOUN, CTAG, CCA	Version after Internal Peer Review and QA
0.9	2021-09-30	HPE, GFT	Overall review
1.0	2021-10-01	ENG	Version for Submission

Executive Summary

Within the INFINITECH Work Package 6 – Tailored Sandboxes and Testbeds for Experimentation and Validation, this document describes the second of the three expected results of the task T6.5 – FinTech/InsuranceTech Testbed Establishment and Customization.

This document describes the creation of the NOVA testbed and the construction of the required architectural stack needed to manage the sandboxes. Then, a way was defined to map a Pilot's components upward (in relation to the INFINITECH Reference Architecture) and downward (in relation to the Kubernetes ecosystem, which was chosen to orchestrate the execution environment). The downward mapping was a placeholder for the imminent migration of the NOVA infrastructure, and it clarified what the contributors expected to get. Moreover, this document shows how that placeholder evolves into an actual set of application-and-orchestration-specific artifacts and how they are consumed in order to get a fully working sandbox for each Pilot. It underlines what the main characteristics of each sandbox are, an explains if any critical points were met and, in this case, how they are handled.

The description of the used tools is provided, to gather both standard and domain-specific/application-specific metrics to make a user be able to perform benchmarking activities and optimizations.

Once all the metrics are identified and all the tools are described, the probes will be accessed over time to reach the third and last objective of having a big picture of the whole workload in the various system conditions that may occur; metrics-related idle information will be an essential input for the deliverable D6.12 – Sandboxes for FinTech and InsuranceTech Innovators - III.

Table of Contents

List of Figures

List of Tables

Abbreviations/Acronyms

Abbreviation	Definition
AgI	Agricultural Insurance
AMI	Amazon Machine Image
API	Application Programming Interface
AWS	Amazon Web Services
AWS EBS	Amazon Web Services Elastic Block Store
AWS EKS	Amazon Web Services Elastic Kubernetes Service
AWS ELB	Amazon Web Services Elastic Load Balancer
AWS KMS	Key Management Service
BP	Blueprint
CICD	Continuous Integration Continuous Development
CLI	Command Line Interface
CNCF	Cloud Native Computing Foundation
CNI	Container Network Interface
DNS	Dynamic Name Resolution
ENI	Elastic network interfaces
EKS	Elastic Kubernetes Service
EO	Earth Observation
GKS	Google Kubernetes Engine
HA	High Availability
HCL	Hashi Corp Configuration Language
HTAP	Hybrid Transactional and Analytical Processing
IAM	Identity and Access Management
IP	Internet Protocol
K3S	Lightweight Kubernetes
K8S	Kubernetes
ML/DL	Machine Learning Deep Learning
PoC	Proof of Concept
PV	Persistent Volume
PVC	Persistent Volume Claim
RBAC	Role-Based Access Control

RKE	Rancher Kubernetes Engine
SHARP	Smart, Holistic, Autonomous, Regulatory-compliant, Personalized
SSH	Secure Socket Shell
YAML	YAML Ain't Markup Language
VaR	Value-at-Risk
VM	Virtual Machine
VPC	Virtual Private Cloud

1 Introduction

The activities performed within the task T6.5 after planning and executing all the technical operations described in the deliverable D6.10 – Sandboxes for FinTech and InsuranceTech Innovators - I, had a positive impact on the involved partners: Pilots and tech-proxies had a first experience with "The INFINITECH way" software development process, targeting the NOVA testbed as the execution environment. This experience is maturing and evolving day-by-day, building up remarkable teamwork that allowed us to overcome all the constraints held by any on-premises infrastructures (in comparison with a cloud solution) to reach, at the end of the day, a smooth development experience that allows users to "just focus on the code and forget about the rest", according to the canonical DevOps best practices.

D6.11 – Sandboxes for FinTech and InsuranceTech Innovators - II digs deeper into the topic of sandbox tailoring, explaining all the artifacts produced by the contributors and the approach taken to reach the objectives that will be briefly discussed in the next paragraph.

1.1 Objective of the Deliverable

Taking a step back, just to provide an eagle's eye view of the whole T6.5 roadmap, the focus on the first period was directed towards the creation of the testbed (NOVA) and to the construction of the required architectural stack to provide and manage the sandboxes. At the same time, the team provided a way to map Pilot's components upward (in relation with the INFINITECH Reference Architecture) and downward (in relation with Kubernetes ecosystem, which was chosen to orchestrate the execution environment). The downward mapping was a placeholder for the imminent migration of the NOVA infrastructure, and it clarified what the contributors expected to get.

This document has, as a primary objective, to describe how that placeholder evolves into an actual set of application-and-orchestration-specific artifacts and how they are consumed in order to get a fully working sandbox for each Pilot. It underlines what the main characteristics of each sandbox are, it explains if any critical points were met and, in this case, how they are handled.

Another and no less important objective converges the analysis to the software development lifecycle and to the software execution lifecycle through metrics collections and related analytics. Some pages will be dedicated to the description of the tools used in order to gather both standard and domain-specific/application-specific metrics, so that the reader will have a reference guide to get started with the active monitoring process for maintenance and operational actions.

Once all the metrics are identified and all the tools are described, the probes will be accessed over time to reach the third and last objective of having a big picture of the whole workload in the various system conditions that may occur; metrics-related idle information will be an essential input for the deliverable D6.12 – Sandboxes for FinTech and InsuranceTech Innovators - III.

1.2 Insights from other Tasks and Deliverables

WP6 has been contributing to different WPs and deliverables and in turn relies on inputs coming from other WPs, as shown in the following figure.

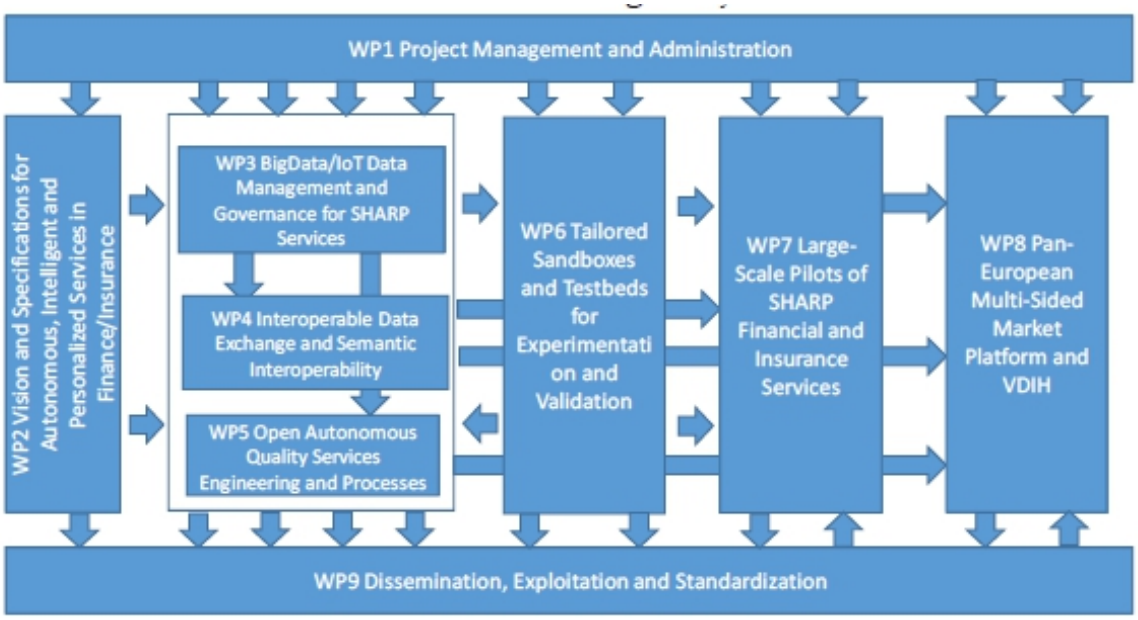


Figure 1 – INFINITECH Work Breakdown Structure

Within T6.5, the deliverables submitted until now from WPs 3-4-5 have been taken into account.

In addition, the following deliverables within the WP6 are key inputs to this document:

D6.2 – Testbeds Status and Upgrades - II,

D6.5 – Tools and Techniques for Tailored Sandboxes and Management of Datasets – II.

It is worth noticing that there is a relation between this deliverable and the deliverable on the Sandboxes in Incumbent Testbeds (D6. 7-8-9), since they share the same goals but from different perspectives.

Finally, the progress of task T6.5 (together with the other tasks in WP6) is a key driver of the INFINITECH WP7, which is focused on the Large Pilots Operations and Stakeholders Evaluation of the proposed Financial and Insurance Services.

1.3 Structure

The content of this document is divided into four chapters. Besides the introductory section which highlights the objectives of the deliverable and provides the context in which this series of activities take place, we propose a technical chapter “INFINITECH System Design for Shared Testbed” that describes, from one side, the overall NOVA architectural stack and the cluster blueprint that generated the five Pilots’ clusters, and from the other side, a detailed description of the system monitoring tools adopted to gather metrics in the different levels of the stack, as well as a description of those meaningful standard metrics aimed to perform a fine-tuning of the running NOVA infrastructure and the virtual environments that are provided in the form of Kubernetes clusters.

The third chapter is fully dedicated to the Pilots. It goes in-depth with the sandbox tailoring and concretizes all the expected results and the configuration objects discussed in the corresponding section of the deliverable D6.10. The chapter also introduces the relevant domain-specific metrics, explaining how they are gathered, and elaborates a preliminary consideration according to the probes activated in the idle condition which represent a starting point for the deductions that will be taken into account later on, after the heavy-workload conditions will be monitored.

The final chapter contains the final thoughts and the anticipations for the next version of the deliverable.

2 INFINITECH System Design for shared testbed

2.1 General system design of the shared testbed

As reported in deliverable D6.10, the Nova shared testbed is a set of hardware resources organized to host several Pilots in a way that Pilots will not interfere each other. Each Pilot is organized as a set of namespaces, each one implementing a use-case.

The Nova infrastructure resources has been partitioned using the same high level design used for the cloud-hosted shared testbed, which is based on a set of Kubernetes [1] (aka K8S) clusters, one for each Pilot, each one containing several namespaces representing use-cases, as depicted in Figure 2.

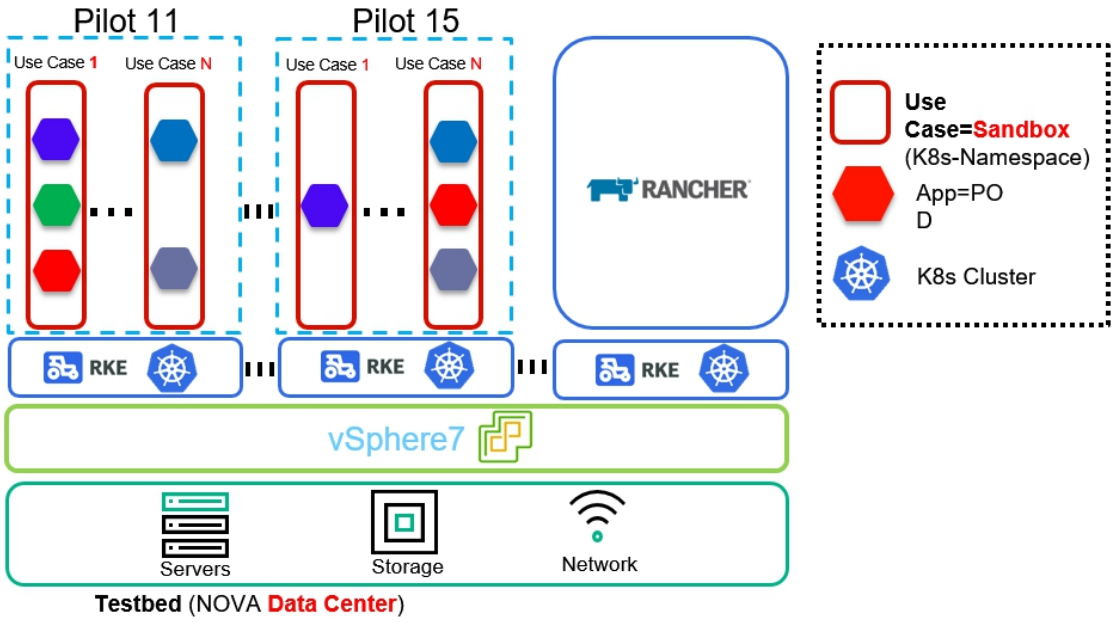


Figure 2 - NOVA Blueprint replication

Since each cluster is realized by several VMs (Virtual Machines), to speed up VMs management, we adopted Packer [2]: an opensource tool to create VM templates, potentially for multiple platforms, with a declarative language. With Packer, it is possible to define a static and versionable virtual machine definition, where it is possible to specify operating system and installed software, and from which the VM image is built without the burden of all classic manual steps necessary to create a VM, like install and configure OS. Packer allows us to configure a single VM template in 3 files, the first is the general definition as follow:

```
{
 "builders": [
  {
   "CPUs": 2,
   "RAM": 2048,
   "RAM_reserve_all": true,
   "boot_command": [
    "<tab> text ks=http://{{ .HTTPIP }}:{{ .HTTPPort }}/ks.cfg<enter><wait>"
   ],
   "boot_order": "disk,cdrom",
   "cluster": "{{user `cluster`}}",
   "convert_to_template": "true",
```

```
    "datastore": "{{user `datastore`}}",
    "disk_controller_type": "pvscsi",
    "folder": "{{user `folder`}}",
    "guest_os_type": "centos7_64Guest",
    "host": "{{user `host`}}",
    "http_directory": "./http",
    "insecure_connection": "true",
    "iso_checksum": "SHA256:b79079ad71cc3c5ceb3561fff348a1b67ee37f71f4cddfec09480d4589c191d6",
    "iso_urls": "https://mirror.bytemark.co.uk/centos/7.9.2009/isos/x86_64/CentOS-7-x86_64-NetInstall-2009.iso",
    "network_adapters": [
     {
       "network": "{{user `network`}}",
       "network_card": "vmxnet3"
     }
    ],
    "password": "{{user `password`}}",
    "ssh_password": "{{user `ssh_password`}}",
    "ssh_username": "{{user `ssh_username`}}",
    "storage": [
     {
       "disk_size": 8192,
       "disk_thin_provisioned": true
     }
    ],
    "type": "vsphere-iso",
    "username": "{{user `username`}}",
    "vcenter_server": "{{user `vcenter_server`}}",
    "vm_name": "template_centos7"
  }
 ],
 "provisioners": [
  {
    "execute_command": "echo '{{user `ssh_password`}}' | sudo -S -E bash '{{.Path}}'",
    "scripts": [
      "script.sh"
    ],
    "type": "shell"
  }
 ]
}
```

That file is parametrized CPU, RAM and disk memory, seems fixed but can be changed during VM instantiation, while all other parameters specified as "{{param `param`}}" ex. "{{user `username`}}", are specified in a second file named“variable.json”:

```
{
  "vcenter_server":"10.172.70.50",
  "username":"******@vsphere.local",
  "password":"*******",
  "datastore":"SVT_DATASTORE_01-VMS",
  "folder": "/Packer/template",
  "host":"infinitech-no1.vps.uninova.pt",
  "cluster": "Infinitech-Uninova",
  "network": "R-UNINOVA-INFINITECH-PROD",
  "ssh_username": "****",
  "ssh_password": "*******"
}
```

This file refers to NOVA vCenter, which is where the infrastructure is hosted and all other base system configuration are located, like:

Operating System

ISO image for operating system installation

Boot order

CPU

RAM

Network configuration

Credential for SSH

Script for further image customization

One additional file is needed, and it is a bash script named “script.sh”, here are placed instructions to install all tools needed for this configuration, in our case: Docker [2] and its prerequisites, open-vm-tools, kubectl and bash-completion. We also replaced service chronyd with ntpd, fill hosts file with necessary entry and reset cloud-init.

```
#remove old docker version
yum remove docker docker-client docker-client-latest docker-common docker-latest docker-latest-logrotate docker-logrotate docker-engine

#install docker
yum install -y yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
#pre-requisite docker
yum install -y device-mapper-persistent-data lvm2 perl
#docker
yum install -y docker-ce docker-ce-cli containerd.io
systemctl enable docker

#install vm-tools
yum install -y open-vm-tools

#Download the kubectl cli
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
mv kubectl /usr/local/bin
chmod +x /usr/local/bin/kubectl

#Tab completition
yum install -y bash-completion

#Disabled chronoyd
systemctl disable chronyd
systemctl enable ntpd

#Customize hosts
echo "10.172.73.13 rancher.vps.uninova.pt">>/etc/hosts

#Reset the machine-id value. This has known to cause issues with DHCP
echo -n > /etc/machine-id

# Reset any existing cloud-init state
#
cloud-init clean -s -l
```

This infrastructure counts several Kubernetes clusters plus one for Rancher [10] application runtime. This cluster is the first created and we used the VM template described above.

This cluster as for Rancher recommendation [30] is composed by three nodes: each node hosts all three main K8S functions: control-plane, etcd and worker. To create this cluster, as for all other clusters, we’ve selected a Kubernetes distribution called RKE [4] (Rancher Kubernetes Engine), a CNCF [5] (Cloud Native Computing Foundation) certified project that simplifies Kubernetes operation, which allows the creation of

clusters without operating system constraints by relying on compatible versions of Docker. It creates a cluster with one command and its declarative configuration simplifies Kubernetes maintenance, enabling versioning, simply and atomic update of infrastructure, modifying cluster definition file. In the following example, it is depicted as a cluster configuration file extract, where the first two cluster nodes are defined.

```
nodes:
- address: 10.172.73.12
  port: "22"
  internal_address: ""
  role:
  - controlplane
  - worker
  - etcd
  hostname_override: rancher01
  user: automation
  docker_socket: /var/run/docker.sock
  ssh_key: ""
  ssh_key_path: ~/.ssh/id_rsa
  ssh_cert: ""
  ssh_cert_path: ""
  labels: {}
  taints: []
- address: 10.172.73.13
  port: "22"
  internal_address: ""
  role:
  - controlplane
  - worker
  - etcd
 … continue
```

On top of that K8s cluster, we installed Rancher platform via Helm [6] package manager.

Rancher (with its GUI) enables easy k8s cluster creation and maintenance. Within it we can manage all NOVA Pilots clusters, and also Rancher K8S cluster itself, named “rancher” as you can see in Figure 3.

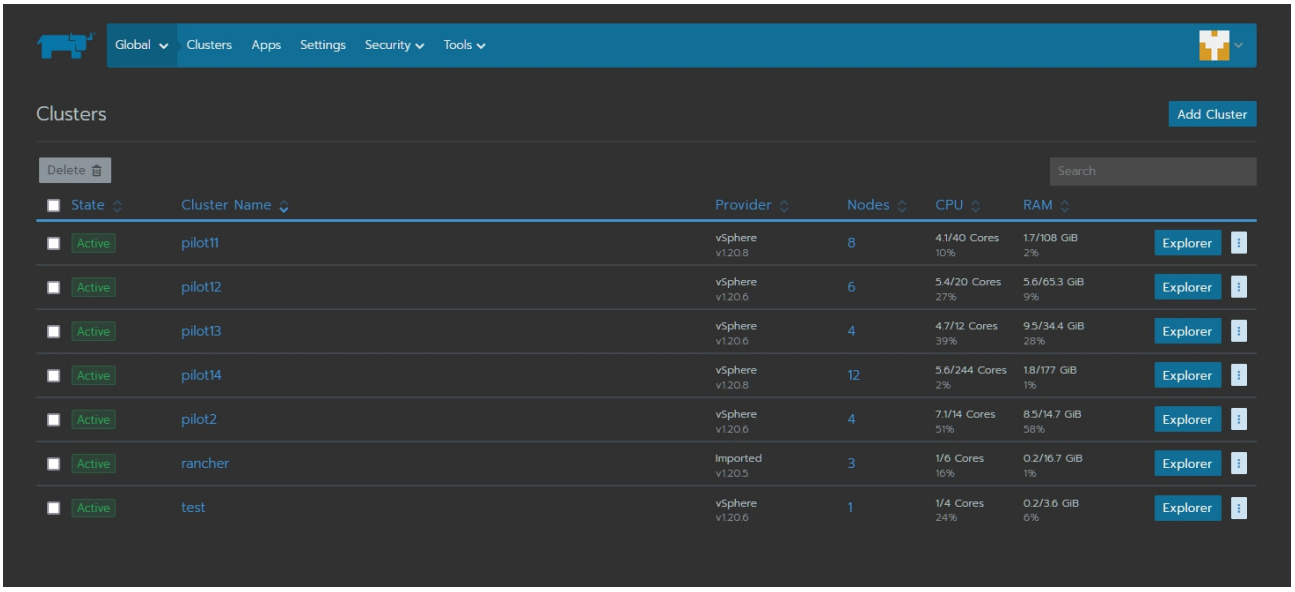


Figure 3 – Overview of the clusters managed by Rancher

Rancher works also as proxy for users that want to access their own cluster via kubectl, thanks to its features of authentication and authorization. We have integrated the Rancher with the Blueprint

OpenLDAP service, in order to allow INFINITECH users to use the same credentials used to access other INFINITECH platforms.

Moreover, to access the Rancher UI (User Interface) via HTTPS, a DNS name has been registered: rancher.vps.uninova.pt to point to the UNINOVA firewall. The firewall converts the public IP into a private IP. The private IP is managed by a VM based on NGINX [7] that acts as reverse proxy to forward the HTTPS requests in round-robin manner to one of the three Rancher's nodes: 10.172.73.12, 10.172.73.13, 10.172.73.14 where the web application is running on port 443, see Figure 4.

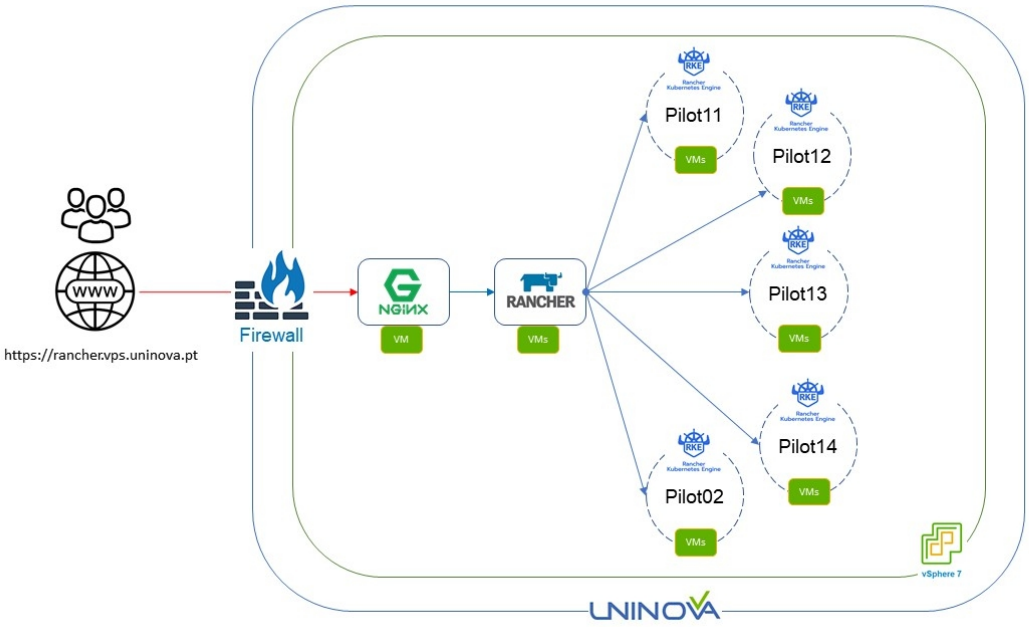


Figure 4 – NOVA Network

All Pilot clusters are easily created via Rancher user interface: An administrator indicates a few necessary parameters such as number of nodes, resources of those nodes and kind of K8s function it implements and the frameworks do the job, including managing VM hosting the nodes, see Figure 5.

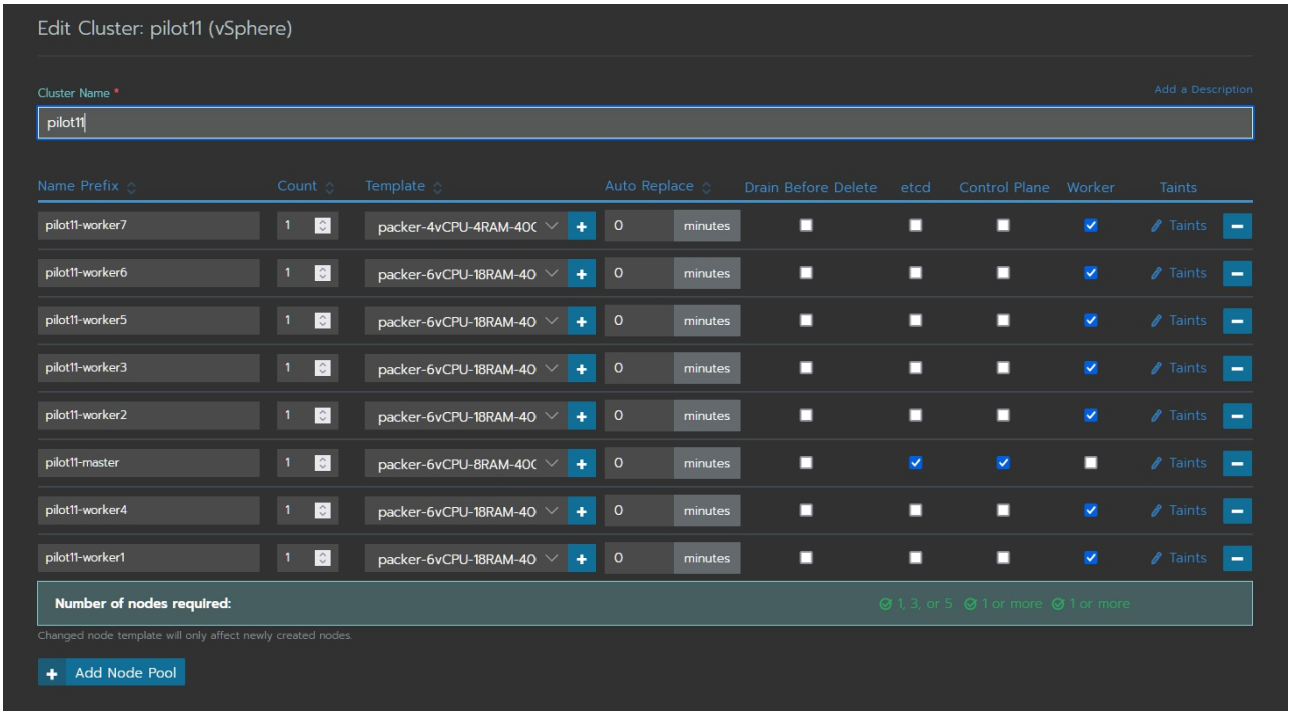


Figure 5 – Edit Cluster on Rancher

But before to start creating the needed clusters, Rancher needs one more configuration: “Node Template”. A node template is a Rancher object that contains the configuration to use when provisioning nodes in a specific provider, cloud or on-premise; in our case the provider is vSphere, as this infrastructure is hosted on a VMware platform.

Starting from the Pilot hardware requirements (see Table 1), we made a bunch of templates that can simplify the operation of sizing the cluster with the appropriate resources, all templates created use the packer VM image described at the beginning of the installation, each one configured and with different resources.

Table 1 – Pilot Requirements

Pilot\HW Requirements	Cores	Memory (GB)	Storage (TB)
Pilot 2	4	16	0.5
Pilot 11	34	104	15.2
Pilot 12	18	64	15
Pilot 13	8	32	0.6
Pilot 14	240	180	0.5

Due to the lack of an automatic virtual machine distribution on hosts, based on resource availability, we make copies of same node template on the different hosts: for that reason in Table 2, the complete host name is not indicated (that should end with the number of host server that runs this image), but only resources configured are given.

Table 2 - Node Template

Name	Cpus	Memory (GB)	Disk (GB)
packer-24vCPU-18RAM-40GB-host(1-3)	24	18	40
packer-6vCPU-18RAM-40GB-host(1-3)	6	18	40
packer-6vCPU-8RAM-40GB-host(1-3)	6	8	40
packer-4vCPU-16RAM-40GB-host(1-3)	4	16	40
packer-4vCPU-4RAM-40GB-host(1-3)	4	4	40

2.2 Testbed monitoring tools

The standard tools to monitor a Kubernetes cluster are Prometheus [8] and Grafana [9]: the first one is the real monitoring tool that collects and stores the metric values, while the second one is the graphic visualization tool that allows users to represent the data in an intuitive and easy to use dashboard.

The following picture depicts the general flow in a hypothetical Kubernetes cluster:

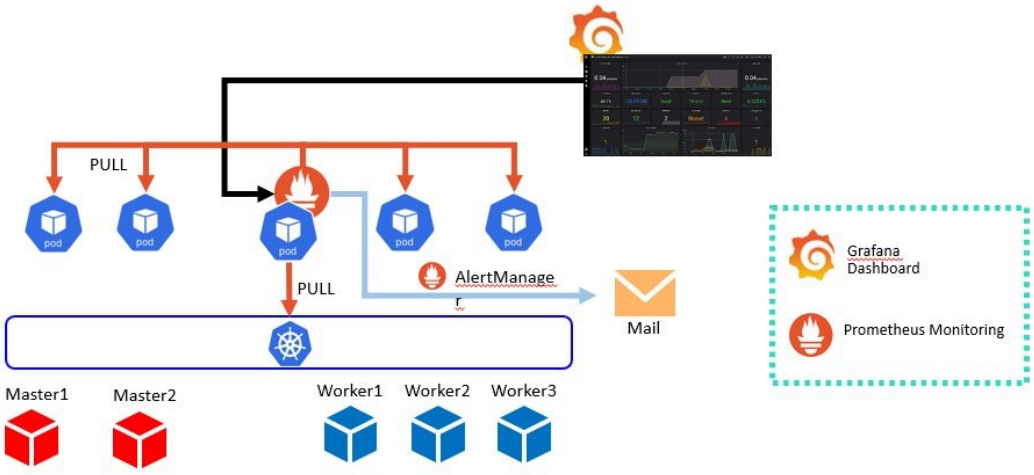


Figure 6 – Prometheus and Grafana in a k8s cluster

As shown in the picture, the monitoring system can be summarized in three components:

Prometheus server: allows users to retrieve metrics from the Kubernetes cluster (like PODs resources request, namespace utilization) and store them in its own key-value database.

Grafana Dashboard: allows users to query the metrics retrieved from Prometheus and render them into beautiful graphs and visualizations

Alert Manager: a Prometheus plugin that handles alerts sent by Prometheus server towards external systems via email, slack notification etc.

In the NOVA infrastructure, the Kubernetes clusters are managed by Rancher, which is natively integrated with Prometheus and Grafana, so in this case, it is not necessary to install and integrate the two tools manually, but it is possible to enable them at installation time or after the cluster is created directly in the Rancher UI.

After selecting a specific cluster, it is possible to check immediately if this feature is enabled from the presence of the Metrics in the bottom (Cluster Metrics, etcd Metric, etc) and for the Grafana logo on the bottom right.

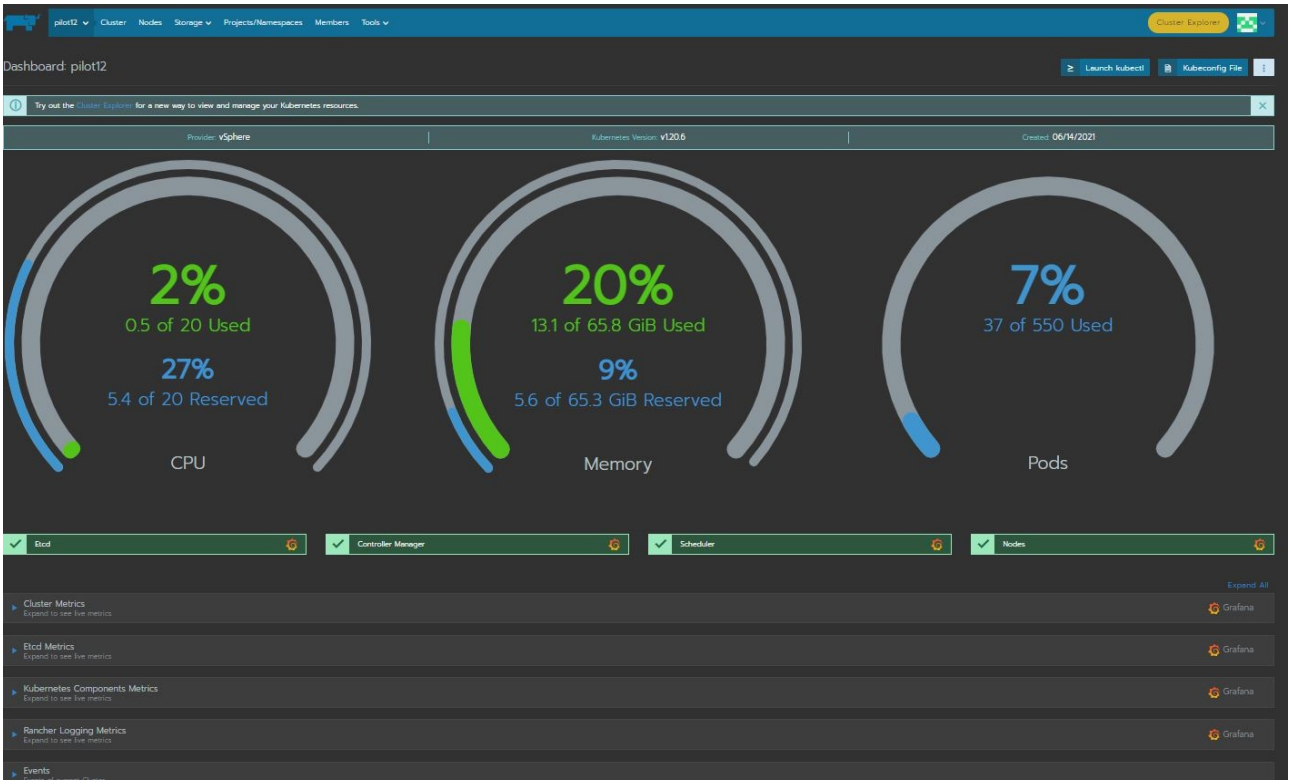


Figure 7 – Cluster-specific main page

To enable or disable this feature or to modify the monitor configuration, it is possible to access the menu Tools and then select Monitoring to reach the monitor configuration page:

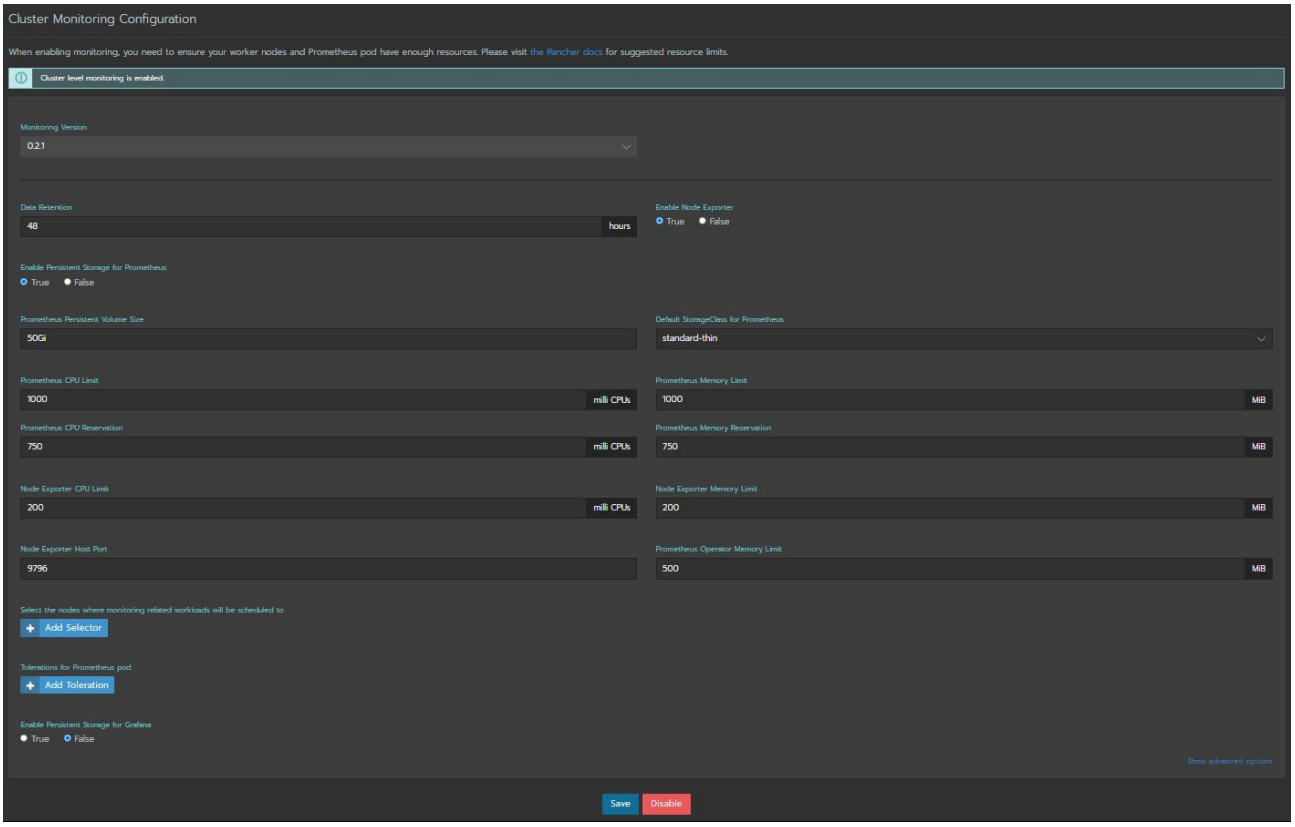


Figure 8 – Cluster-specific Monitoring configuration page

Here, it is possible to choose the Data Retention time, the size of the persistent volume where Prometheus will store historical data and resource limit of the Promethus-Grafana PODs in terms of CPU and Memory.

Referring to Figure 7, by expanding one item (for example Cluster Metrics), it is possible to see the summary of the related metrics on the same page, with Rancher look and feel. Hovering the mouse on one graph, the appropriate values are shown as depicted in the CPU utilization section in Figure 9:

Figure 9 -– Monitoring Metrics Summary

The data visualization shown inside Rancher can be also accessed using the Grafana tool, clicking on the Grafana logos on the right of the Figure 7:

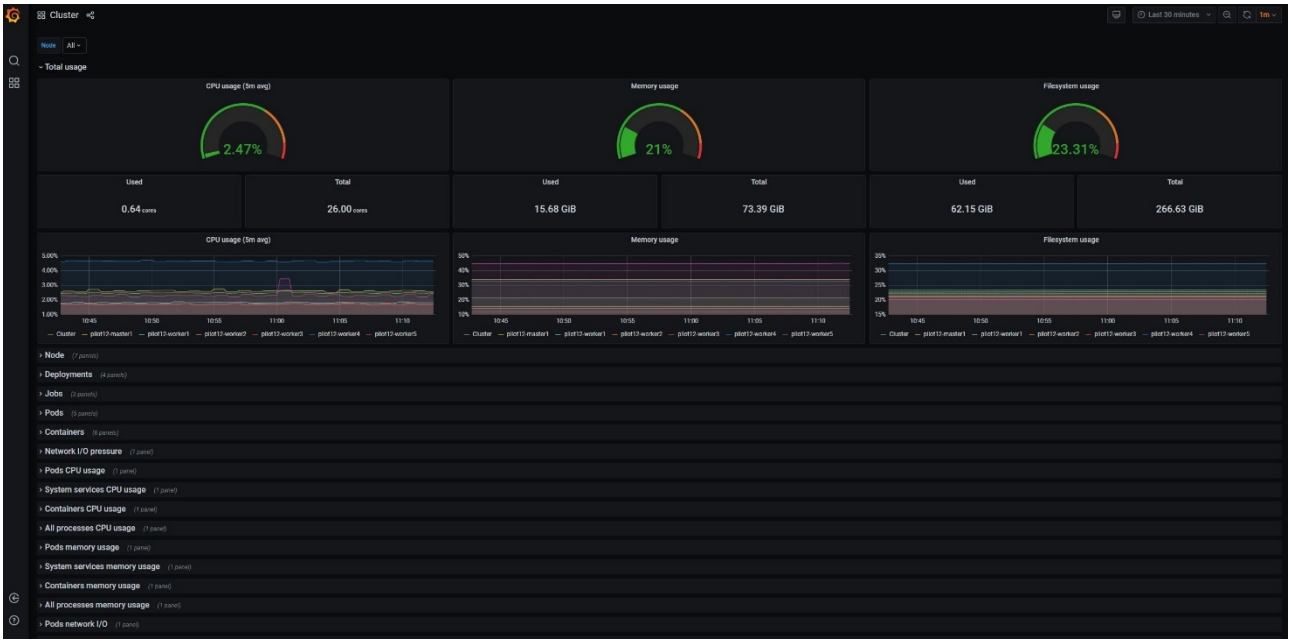


Figure 10 – Grafana Data Representation

In this way, it is possible to interact with Grafana that is a standard de facto monitoring and data visualization software, shown in Figure 10. In the Rancher visualization as well as in Grafana, it is possible to explode the metric categories and hover the mouse on the graphs to access the data values (see Figure 11).

Figure 11 – Grafana-specific Values

2.3Testbed metrics collection

Numerous metrics collected by Prometheus are provided by the Kubernetes metric-server. In the following table, the most important metrics accessible by Rancher are shown:

Table 3 – Metrics collection

Category	Metric
Cluster Metrics	CPU Utilization
	Load Average
	Memory Utilization
	Disk Utilization
	Disk i/O
	Network Packets
	Network I/O
Etcd Metrics	GRPC Client Traffic
	DB Size
	Active Streams
	Raft Proposals
	RPC Rate
	Disk Sync Duration
Kubernetes Components Metrics	API Server Request Rate
	Controller Manager Queue Depth
	Scheduling Failed Pods
	Ingress Controller Connections

For further information about Rancher metrics, it is possible to refer to [4].

3 Sandboxes for FinTech and InsuranceTech innovators

3.1 Pilot#2 – Real-time risk assessment in Investment Banking

The Pilot#2 implements a real-time risk assessment and monitoring procedure for two standard risk metrics: VaR (Value-at-Risk) and ES (Expected Shortfall). The main outcome is the measurement of market risks of Forex portfolios. In addition, the Pilot will evaluate what-if scenarios allowing pre-trade analysis, i.e., estimating changes in risk measures before a new trading position is entered. Moreover, the Pilot will implement sentiment analysis in financial and economic news providing complementary risk information to traders and portfolio managers. While VaR and ES are quantitative risk measures based on numerical price data, the market sentiment will be derived from text data applying natural language processing. An overview of Pilot #2 is given in Figure 12.

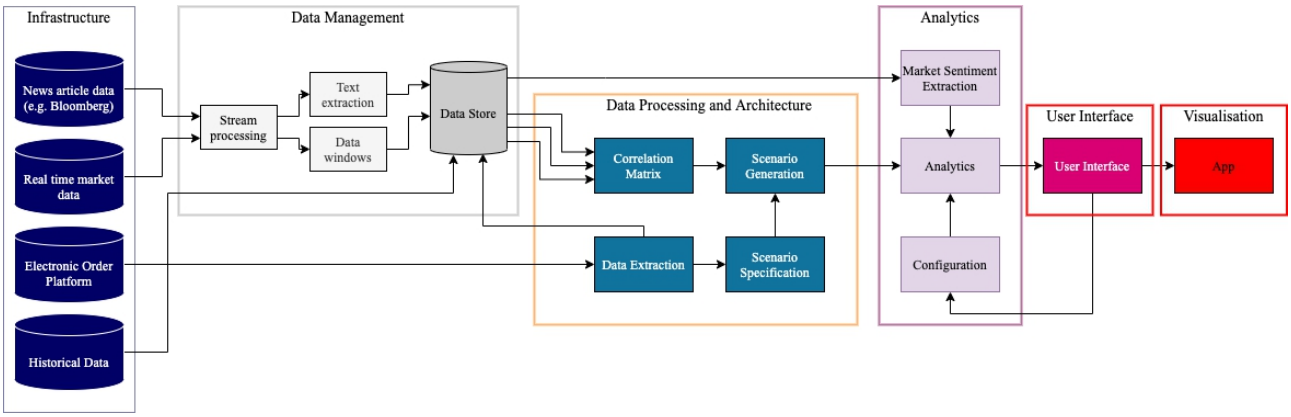


Figure 12 – Pilot #2 Overview

In the overall architecture, there can be identified three layers of building blocks that consists of the overall solution:

Data Management layer: Data from the real-time market database and the news feed databases is injected into the Data Management layer through a stream processing component which is capable of handling large volumes of data that feature very high ingestion. The real-time data is concatenated with the historical data on the fly and at the same time is appropriately transformed in data windows, creating segments of time series.

Data Processing and Architecture layer: In order to update risk measurement, instantly when a new trading position takes place, data from the Electronic Order Platform is injected directly to Data Processing and Architecture layer. Consequently, the updated portfolio positions along with the latest market values are taken into account towards calculating the correlation matrix. The latter is required in order to calculate VaR and ES. Furthermore, this layer is responsible to generate the various scenarios through Monte Carlo simulations which are also required for the risk estimation.

Analytics layer: This layer receives inputs from the Data Management and Processing layers to estimate VaR and ES, leveraging both statistical and deep learning methods. Moreover, Analytics

layer receives input from the Pilot's user interface in order to provide the what-if-analysis feature which is also using the developed risk assessment models.

3.1.1 NOVA sandbox description

The current version of Pilot #2 consists of the following four distinct components:

AI for VaR Prediction implements building blocks from both Data Processing and Analytics layers.

UI based on VaR implements the what-if-analysis feature while also serves as the Pilot's user interface.

infinistore is the main component of the data management layer serving as a central data store, providing the online aggregates feature.

Kafka is also part of the data management layer, which enables the injection of high-volume data streams (i.e., tick data) to the Infinistore.

The following Kubernetes elements define the Pilot's deployment in the dedicated sandbox:

statefullset.apps/infinistore: this is the stateful set that contains the INFINISTORE data management system. As the INFINISTORE is a stateful component, a stateful set is considered the correct choice instead of a deployment config. The stateful sets preserve their internal IPs when a pod is being restarted. At this phase of the project and for the given data load, the requirements for computational power are 4 vCPUs with 8 GBs of memory.

service/infinistore-service: this service element allows the connectivity of the INFINISTORE stateful set with other elements inside the sandbox.

persistent.volume.claim/infinistore-datasets-pvc: this is the persistent volume claim that is needed to persistently store the ingested data of the datastore so that it can be available every time the datastore restarts. At this phase of the project, the requirement is for 50 GBs of storage to validate that the integrated solution is working with a medium data load.

statefulset.apps/lx-kafka: this is the stateful set that contains the LX-Kafka queue. Similar to INFINISTORE, a stateful set is used to preserve the IPs of Kafka's internal components. To serve the current version of Pilot #2, 2vCPUs with 4 GBs of memory are required.

service/lx-kafka-np: this is a node port that allows connectivity with components that are external to the sandbox via the internet. Thus, this service enables a secure connection, which is required to ingest data from outside of the sandbox to Kafka. As a result, this node port exposes the defined port of LX-Kafka to the internet.

deployment.apps/ai-model-for-var-prediction: this is deployment config that contains the ai-model-for-var-prediction component. The latter, being a REST API, does not need to persist state. Thus, a deployment controller has opted, which is lighter than a stateful set. Currently, the requirements for computational power are 2 vCPUs with 2 GBs of memory.

service/ai-model-for-var-prediction-np: this is a node port service that allows external (outside of the sandbox) connectivity to the ai-model-for-var-prediction deployment. This connectivity is required in order to post new trading positions from the bank-side to the ai-model-for-var-prediction deployment.

deployment.apps/ui-risk-assessment-based-on-var: Similar to deployment.apps/ai-model-for-var-prediction, this is also a deployment config containing the ui-risk-assessment-based-on-var component. The requirements for computational power of this component are 2 vCPUs with 2 GBs of memory.

service/ui-risk-assessment-based-on-var-np: this is also a node port service that allows external (outside of the sandbox) connectivity to the ui-risk-assessment-based-on-var deployment. This connectivity is required to enable access to the Pilot's UI, which is deployed in the sandbox via the internet.

3.1.2 Second-stage components deployment on NOVA

Each of the aforementioned Pilot's components is deployed in a dedicated node. As a result, the deployed sandbox consists of the following K8s pods:

pod/infinistore, which instantiates the stateful set of INFINSTORE

pod/lx-kafka, which instantiates the stateful set of KAFKA

pod/ai-model-for-var-prediction, which instantiates the deployment set of ai-model-for-var-prediction

pod/ui-risk-assessment-based-on-var, which instantiates the deployment set of ui-risk-assessment-based-on-var

As mentioned in the previous subsection, for the components of Pilot #2 to be able to reach each other, there has been defined services that expose the corresponding ports. For instance, the following code snippet of the ai-model-for-var-prediction - deployment depicts the port that needs to be reachable:

```
spec:
    containers:
     - env:
        - name: DATASTORE_HOST
          value: infinistore-service
       image: harbor.infinitech-h2020.eu/analytics/ai-model-for-var-prediction: latest
       name: ai-model-for-var-prediction
       ports:
        - containerPort: 5000
```

Moreover, the respective node port has been defined that exposes the specific port (the port where the API listens to) to external components, as depicted in the following code snippet:

```
spec:
 type: NodePort
 selector:
   app: ai-model-for-var-prediction
 ports:
  - name: "5000"
    protocol: TCP
    port: 5000
    targetPort: 5000
    nodePort: 30204
```

Here we can see that the port where the ai-model-for-var-prediction API listens to, the 5000, is being mapped to a target port 30204. That means that external components need to make a HTTPS call to the external IP of the Kubernetes cluster on port 30204. Then the Kubernetes cluster will forward this connection to the specific pod, the ai-model-for-var-prediction as defined by the appropriate selector in the code snippet, mapped to its 5000 port that the pod has exposed internally.

The same approach is utilized for the configuration of the ui-risk-assessment-based-on-var component.

3.1.3 Metrics in idle condition

The following figures (Figure 13 and Figure 14) show the current resources utilization by Pilot #2 in idle conditions.

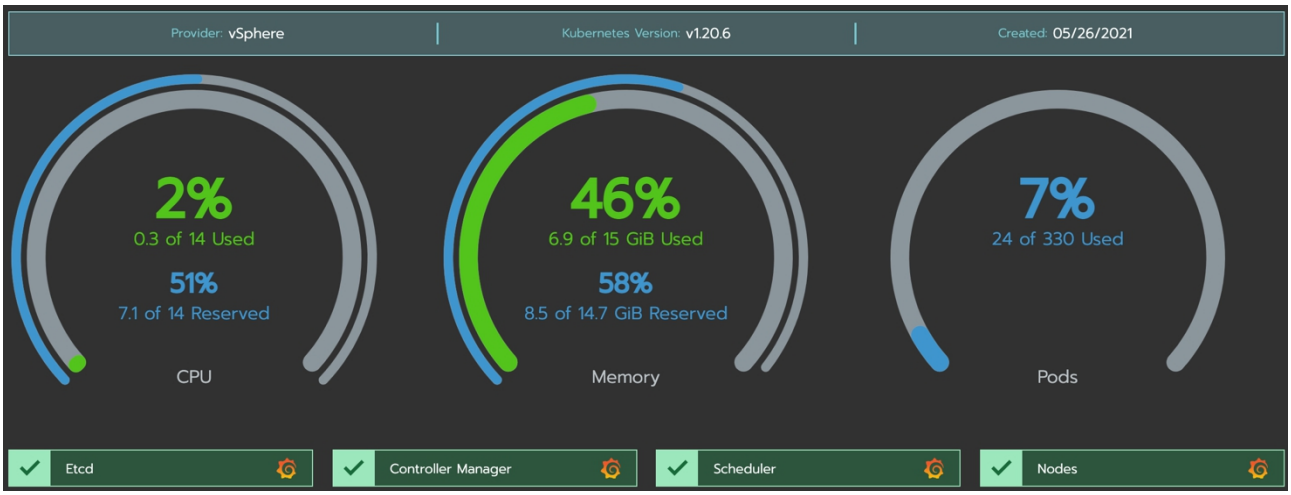


Figure 13 – Resources utilization by Pilot#2

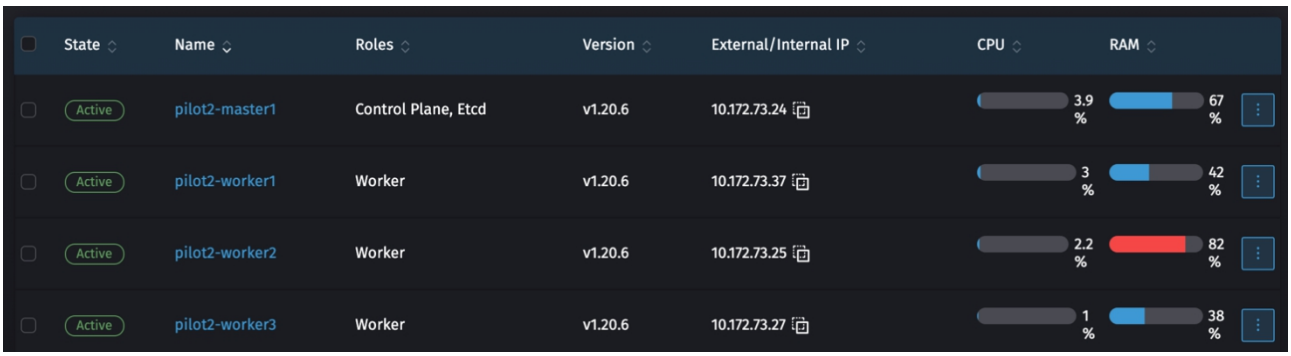


Figure 14 – Pilot#2 nodes list

3.2 Pilot#11 – Personalized insurance products based on IoT connected vehicles

Pilot#11 belongs to the Cluster 4, intended to exploit IoT infrastructures and real-world data to enhance risk profiling methodologies by applying AI technologies to offer customised insurance products and avoid fraud. In particular, as presented in D7.1 and D7.12 and expanded in D6.10, Pilot#11 relies on an IoT connected vehicles infrastructure and an urban traffic simulator to capture, analyse and synthesise

technical datasets from connected cars. This will be merged with other relevant data sources (e.g. traffic alerts and weather information) to develop two AI powered services oriented to car insurance business: Pay as you Drive, to customise prices by classifying the driver by the way he/she drives; and Fraud Detection to identify the actual driver and driving conditions within a traffic incident.

From the overall approach of the Pilot presented in D7.12, the second stage of its sandbox deployment corresponds to the data gathering, standardisation, homogenisation, and presentation components, as shown in Figure 15, leaving the AI models and AI framework for the final stage.

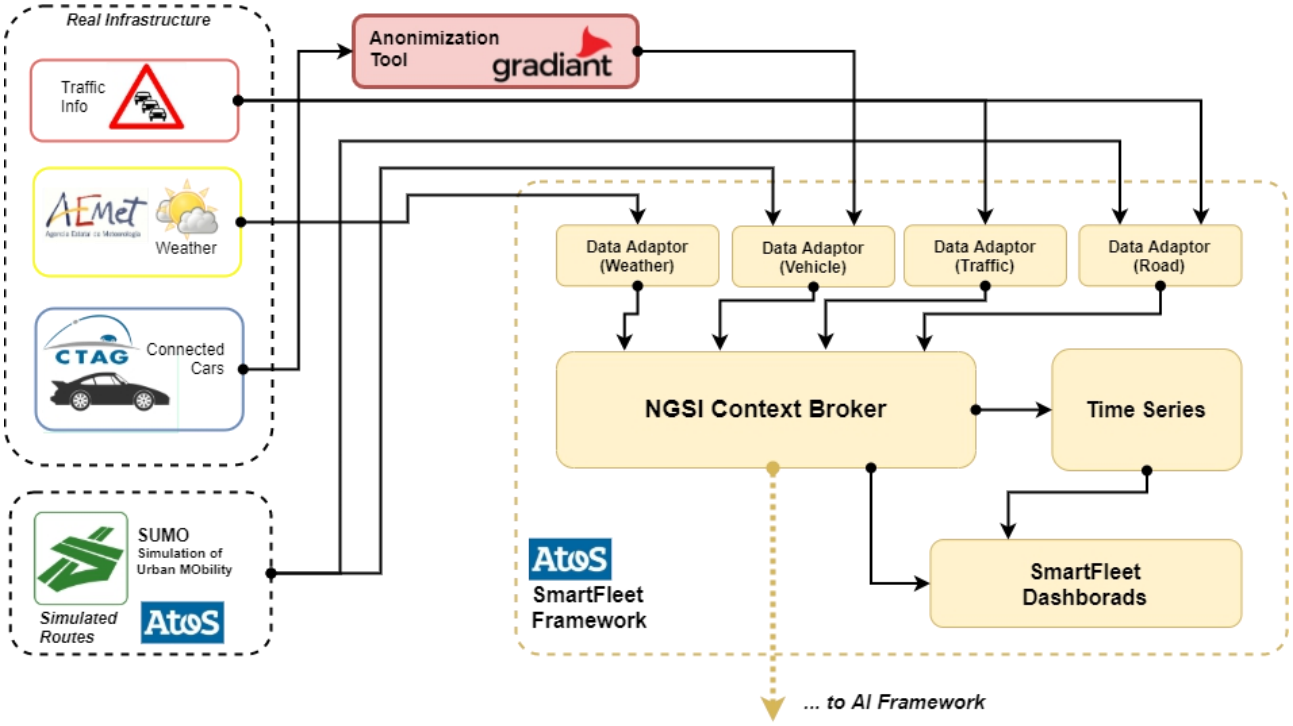


Figure 15 – Pilot #11 data gathering & management

This infrastructure is covered by the Smart Fleet platform and the Urban Traffic Simulator that will provide the required datasets for the exploitation components.

3.2.1 NOVA sandbox description

Figure 16 presents the specific components that build the Smart Fleet framework which is the core of the data gathering and management layer of the Pilot #11. These components are deployed in the NOVA testbed according to the INFINITECH Kubernetes-based approach.

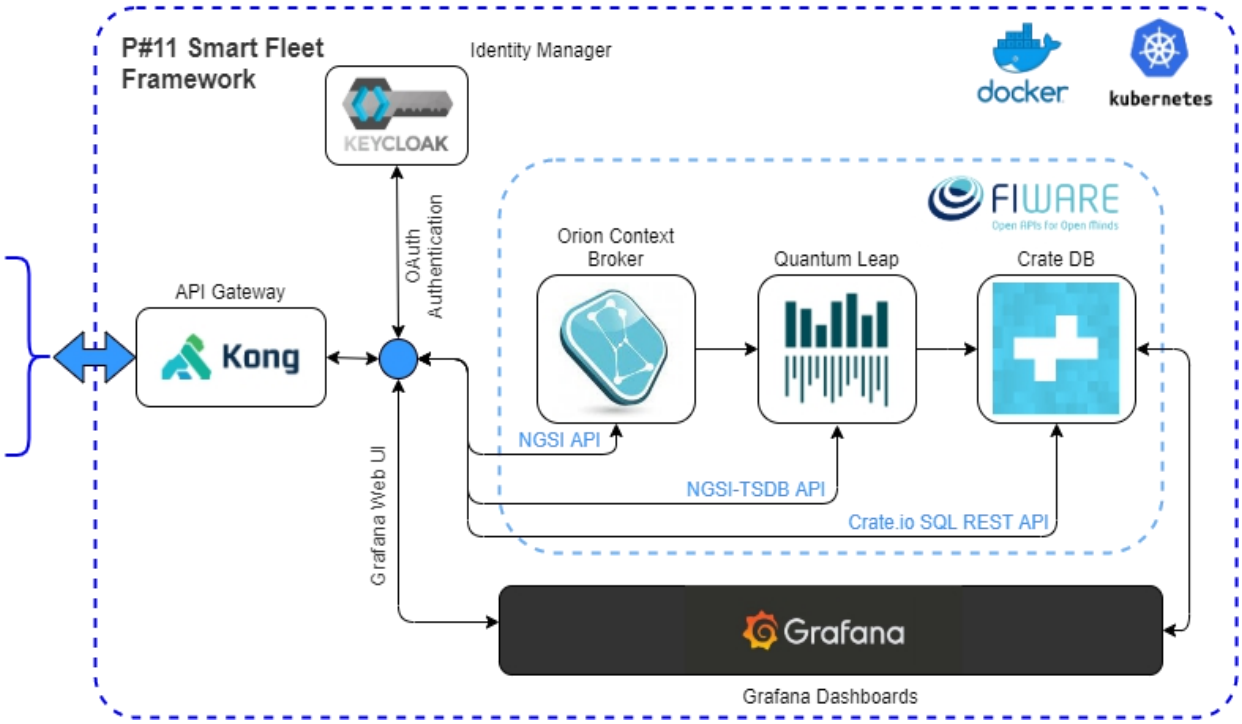


Figure 16 – Atos' Smart Fleet core components

The Atos Smart Fleet framework is based on FIWARE architecture and its backbone is composed by:

the NGSI Context broker, which manages all the information coming from the integrated data sources according to the selected FIWARE Data models [11]. The version used for this deployment customises the Orion Context broker V2.2 [12] which supports and provides a NGSIv2 [13] interface to query/retrieve context information.

An instance of a CrateDB open-source SQL database to support all historical context information and to implement an SQL RESTful interface that provides with classified and homogenised datasets to the AI framework, whilst supports other data analytics and presentation tools.

The Quantum Leap [14] service that, on one hand connects the Context Broker with the Database for gathered data persistence and, on the other, it implements an NGSI-Time Series Data Base (NGSI-TSBD) [15] RESTful API to provide time-related historical information

Connected to this FIWARE backbone, and complementing this baseline, there are also deployed:

an instance of Grafana, a dashboard monitoring tool that allows the end user to create their own visualizations of all the gathered data and provides a web-based user interface to visualise and share all of them

an API Gateway (Kong [16]) to concentrate the access for all the APIs that compose the data access and data uploads for the ATOS Smart Fleet framework

The Pilot #11 sandbox deployment includes a customised instance of the SUMO (Simulator of Urban Mobility) [17] with several embedded city scenarios that provides, through NGSI API and according FIWARE Vehicles data model, on demand simulated data from connected vehicles.

The proper performance of Pilot #11 also requires from specific NGSI adaptors (also known as injectors) to capture information from real-time sources (vehicles, weather, and alerts). The NOVA sandbox will include the weather injector as an open-source code that deals with open data sets from the Spanish Meteorological Agency (AEMET) [18]. Injectors for connected vehicles and traffic alerts are specific developments for CTAG framework and will remain on CTAG premises, but the reposted data will be included in sandbox databases.

This sandbox will also interact with the Anonymization tool provided by Gradiant and be deployed as part of the INFINITECH technologies.

With these deployed components, Pilot#11's NOVA sandbox (second stage) will provide end users with connected vehicles, weather information and traffic alerts generated within the Pilot context, as well as with a tool to simulate new traffic information and collect synthetic vehicles data.

3.2.2 Second-stage components deployment on NOVA

According to the description presented in the previous section, the resources deployed in NOVA Kubernetes infrastructure are summarised in Figure 17.

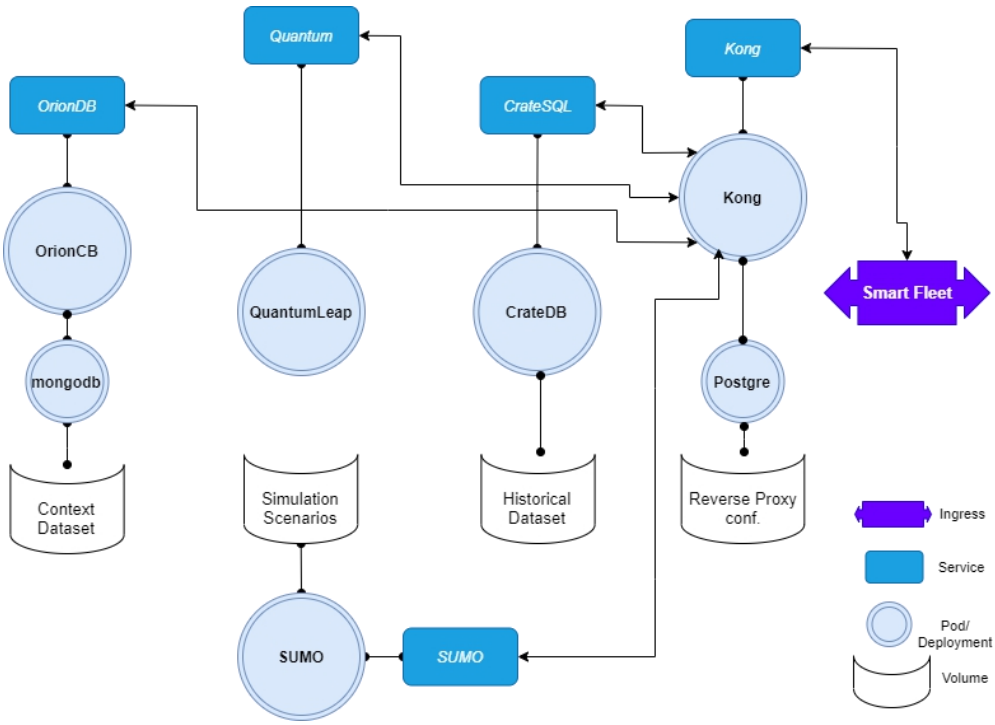


Figure 17 – Pilot #11 application topology on Kubernetes

Grouped as the aforementioned components, the deployed Kubernetes assets supporting each of them are:

Context Broker:

- fiware/orion/orion-cb-deployment.yaml: deploys the FIWARE Orion Context Broker (fiware/orion:2.5.2) and configures the container port to support the NGSIv2 API
- fiware/orion/orion-cb-service: sets and exposes (internally) the NGSIv2 API
- fiware/mongodb/mongodb-cb-pvc.yaml: PersistentVolumeClaim for the Context Information storage
- fiware/mongodb/mongodb-cb-deployment.yaml: deploys the MongoDB database to support Orion Context Broker
- fiware/mongodb/mongodb-cb-service.yaml: service to expose the MongoDB port and get connected with the Orion CB.

Quantum Leap:

- fiware/quantum/quantumleap-deployment.yaml: deploys the FIWARE Quantum Leap component (orchestracities/quantumleap:0.8.1) and configures the container port to support the NGSI-TSDB API
- fiware/quantum/quantumleap-service: sets and exposes (internally) the NGSI-TSDB API
- fiware/quantum/quantumleap-configmap.yaml: configures Quantum Leap to connect to the CrateDB instance

Crate DB:

- fiware/cratedb/crate-db-deployment.yaml: deploys the CrateDB instance (crate:4.1.2) and configures the container ports to support CRUD operations
- fiware/cratedb/crate-db-service: set and expose (internally) the CrateDB ports to configure database accesses
- fiware/cratedb/crate-db-pvc.yaml: PersistentVolumeClaim for the historical dataset storage

API Gateway

- access/kong/kong-deployment.yaml: deploys the modified Kong instance (atossmartfleet/kong_oicd:1.0) and configures the container ports to support management and access interfaces
- access/kong/kong-service: set and expose (internally) the kong's ports for its APIs
- access/kong/kong-manager-ingress: sets the external access for the kong management API
- access/kong/kong-smartfleet-ingress: sets the external access for all the endpoints configured in kong to access the services (APIs) supported by the SmartFleet platform (NGSIv2, NGSI-TSBD, SQL and SUMO API)
- access/postgreSQL/postgres-db-pvc.yaml: PersistentVolumeClaim for the API Gateway configuration persistence
- access/postgreSQL/postgres-deployment.yaml: deploys the PostgreSQL database for Kong persistance
- access/postgreSQL/postgres-service.yaml: creates and exposes the PostgreSQL database service
- Simulator for Urban Mobility
- simulator-urban-mobility/sumoserver-configmap.yaml: configures the SUMO service to work as an NGSI data injector
- simulator-urban-mobility/sumoserver-deployment.yaml: deploys the SUMO server customised to integrate with the Smart Fleet framework
- simulator-urban-mobility/sumoserver-service.yaml: exposes the API to manage the SUMO simulations

At the time of writing these lines, the injector to import AEMET weather datasets according NGSI format and its corresponding K8s files are under development.

In addition, this Pilot#11 deployment also feeds the Anonymisation component[3] developed in T3.5 by GRAD, which in turn captures raw data from routes in NGSI format to anonymise these datasets. This will be later used by the AI models to evaluate different data anonymisation levels and its impact on the AI assisted drivers' classification.

[3] D3.13 Data Governance Framework and Tools – II

3.2.3 Metrics in idle condition

The following figures show a snapshot of the standard metrics in idle conditions of Pilot #11 Kubernetes cluster.

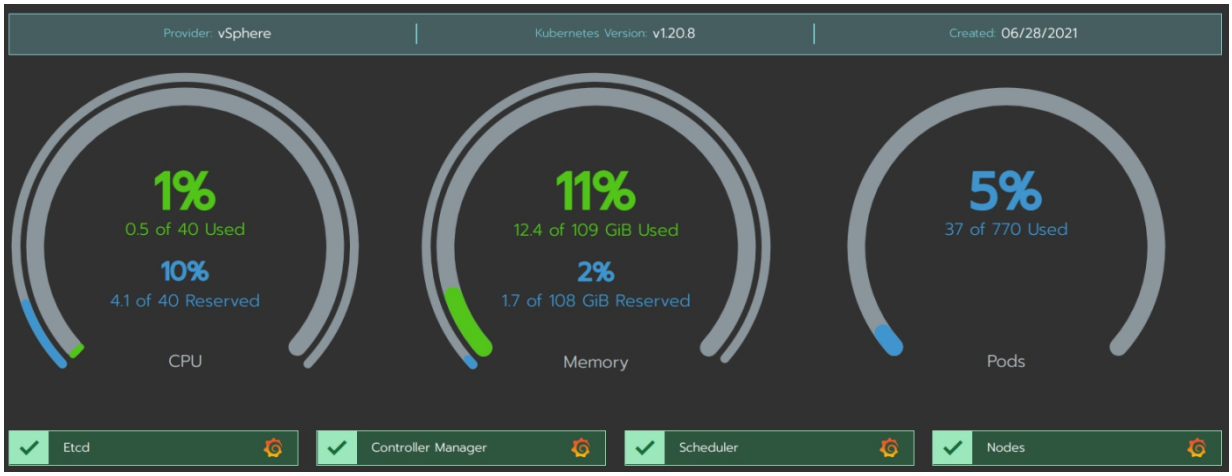


Figure 18 – Resources utilization by Pilot #11

State	Name	Roles	Version	External/Internal IP	CPU	RAM
Active	pilot11-master1	Control Plane, Etcd	v1.20.8	10.172.73.63	2.2 %	49 %
Active	pilot11-worker1	Worker	v1.20.8	10.172.73.64	0.84 %	6.8 %
Active	pilot11-worker2	Worker	v1.20.8	10.172.73.69	1 %	7.1 %
Active	pilot11-worker3	Worker	v1.20.8	10.172.73.62	0.87 %	6.9 %
Active	pilot11-worker4	Worker	v1.20.8	10.172.73.65	0.87 %	6.9 %
Active	pilot11-worker5	Worker	v1.20.8	10.172.73.68	0.86 %	6.8 %
Active	pilot11-worker6	Worker	v1.20.8	10.172.73.67	1.3 %	12 %
Active	pilot11-worker7	Worker	v1.20.8	10.172.73.61	1.4 %	34 %

Figure 19 – Pilot #11 nodes list

3.3 Pilot#12 – Real World Data for Novel Health-Insurance products

Personalization of health insurance products needs to be based on continuous risk assessment of the individual, since lifestyle and behaviour cannot be assessed at one instance in time; they involve people's habits and their continuous change. Health insurance products which employing continuous assessment of customers' lifestyle and behaviour are dynamically personalized.

Behavioural assessments, much like their clinical counterparts, rely on data. For behaviour, the data collection needs to be continuous, facilitated by software tools for the collection of information capturing the important aspects of lifestyle and behaviour. In the Pilot #12 of INFINITECH, insurance experts define the data to be collected, and the Healthentia e-Clinical platform facilitates the collection. Continuous risk assessment services are provided to health insurance professionals by training machine learning (ML) prediction models for the required health parameters.

Pilot #12 of INFINITECH focuses on health insurance and risk analysis by developing two AI-powered services, risk assessment and fraud detection: The risk assessment service allows the insurance company to adapt prices by classifying individuals according to their lifestyle. The fraud detection service is based on outlier analysis for data, but mainly on the use of a virtual coach to advise individuals in their lifestyle choices, aiming at improving their health but also in persuading them to use the system correctly. These two services rely on a model of health outlook trained on the collected data and used in the provision of the services.

An overview of Pilot #12 is given in Figure 20. It comprises two systems: The Pilot #12 testbed, built within the INFINITECH project and deployed on the NOVA sandbox. The Healthentia e-Clinical platform, provided by Innovation Sprint. The data is collected by Healthentia. The Pilot #12 testbed facilitates secure and privacy-preserving offline model training. The trained model is utilized for the risk assessment and the lifestyle coach online ML services of Healthentia, and the results are visualized by the dashboards of the Healthentia portal web app.

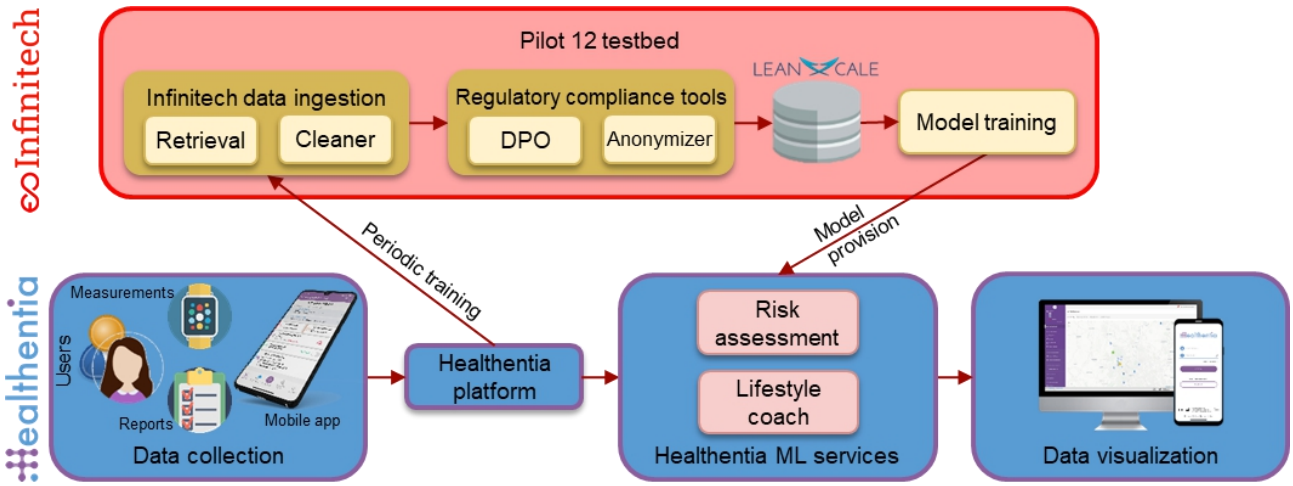


Figure 20 – INFINITECH Pilot #12 system comprising Healthentia and the Pilot #12 testbed

3.3.1 NOVA sandbox description

Pilot #12 is split into two parts, the online that is handled within the Healthentia platform, and the offline model training that is deployed on the NOVA sandbox. The components deployed on the sandbox are:

Data Collection Tool (UBITECH): this tool continuously queries the Healthentia platform for new data and stores it in the miniIO internal storage.

MinIO (UBITECH): This is a single MinIO instance used as an internal only storage (not exposed to the outer world in any way) that will maintain the raw data from the Healthentia platform, as it was received by the Data Collection Tool.

Regulatory Compliance Tools (ATOS): The Data Protection Orchestrator is invoked to start a new model training process. It contacts the Data Collection Tool to retrieve the list of files in miniIO that

should be included in the training process, and then forwards this information to the Anonymization Tool.

Anonymization Tool (GRAD): The anonymizer loads the designated data from the MinIO and performs different levels of anonymization. It then stores the results in LeanXcale.

Infinistore (LeanXcale): This is the data repository for model training. Data anonymized at different levels is stored here.

Model Trainer (Innovation Sprint): This is a collection of Python scripts and classes handling: (a) training, validation and testing dataset creation for different input attributes' and output outcomes' scenarios, (b) model training, (c) model evaluation and (d) model exporting to Healthentia.

The components and their interactions are depicted in Figure 21.

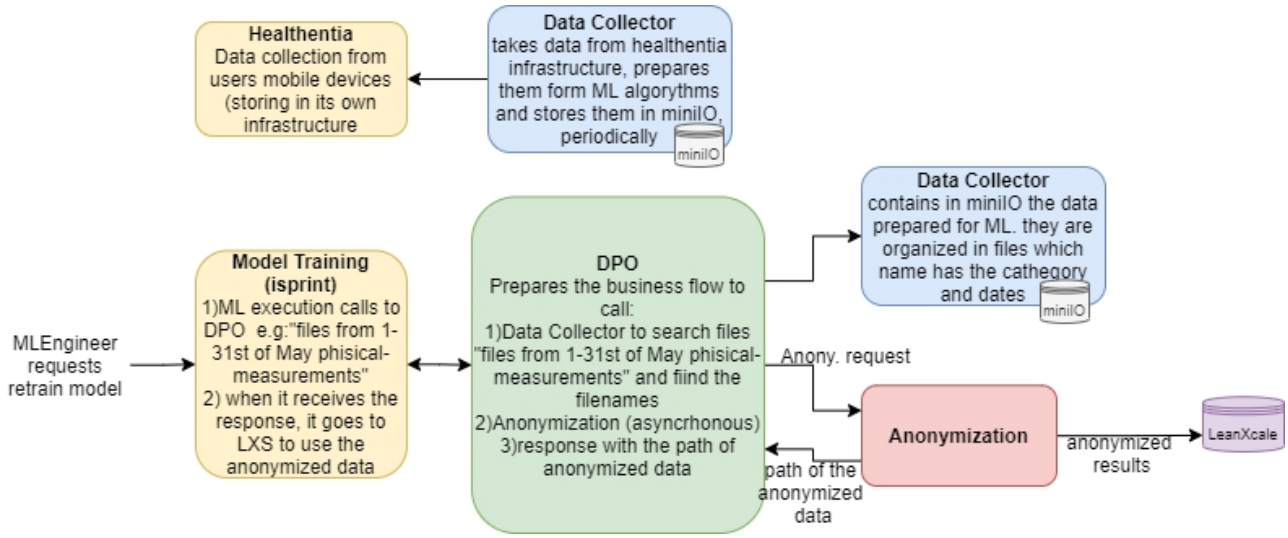


Figure 21 – INFINITECH Pilot #12 testbed components deployed on NOVA sandbox

3.3.2 Second stage components deployment on NOVA

The Data Collection Tool and infinistore (LeanXcale) are already deployed on the NOVA sandbox. For details about how to deploy infinistore instances, please refer to the section 3.4.2 of this document.

MinIO that is used as an internal storage, as described in section 3.33.3.1, is deployed as a stateful set in a standalone mode (not distributed/clustered mode). It utilizes a persistent volume claim of the size of 1 GB and exposes internally a port, so that other services of the deployment can communicate with it.

The definition of the persistent volume is included in the following snippet:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: infinitech-data-checkin-minio-data1-1
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: standard-thin
  resources:
    requests:
      storage: 1Gi
```

Furthermore, the service definition responsible for internally exposing the required port is the following:

```
apiVersion: v1
kind: Service
metadata:
  name: minio-1
  labels:
    app: minio-1
spec:
  ports:
    - name: "9000"
      port: 9000
      targetPort: 9000
  selector:
    app: minio-1
```

Finally, the stateful set definition that creates the single MinIO pod is depicted below:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: minio-1
  labels:
    app: minio-1
spec:
  serviceName: minio-1
  replicas: 1
  selector:
    matchLabels:
      app: minio-1
  updateStrategy:
    type: RollingUpdate
  podManagementPolicy: OrderedReady
  template:
    metadata:
      labels:
        app: minio-1
    spec:
      containers:
        - args:
            - server
            - /data
          env:
            - name: MINIO_ROOT_USER
              valueFrom:
                configMapKeyRef:
                  key: MINIO_ROOT_USER
                  name: env-minio-env
            - name: MINIO_ROOT_PASSWORD
              valueFrom:
                configMapKeyRef:
                  key: MINIO_ROOT_PASSWORD
                  name: env-minio-env
          image: minio/minio:RELEASE.2021-06-17T00-10-46Z
          livenessProbe:
            exec:
              command:
                - curl
                - -f
                - http://localhost:9000/minio/health/live
            failureThreshold: 3
            periodSeconds: 30
            timeoutSeconds: 20
          name: infinitech-data-checkin-minio-1
```

```
        ports:
          - containerPort: 9000
        resources: {}
        volumeMounts:
          - mountPath: /data
            name: infinitech-data-checkin-minio-data1-1
    restartPolicy: Always
    volumes:
      - name: infinitech-data-checkin-minio-data1-1
        persistentVolumeClaim:
          claimName: infinitech-data-checkin-minio-data1-1
```

For the installation of the Data Collection Tool microservice, the Deployment option of Kubernetes was chosen. Unlike stateful sets which are more oriented for stateful objects like databases, deployments are oriented for stateless objects, such as microservices. A persistent volume of 100 MB for storing the tool's logs is mounted to the created pod and additionally a configuration for exposing internally the port of the pod took place, similarly to the one of MinIO.

The snippet of the Data Collection Tool's persistent volume is the following:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: infinitech-data-checkin-handler-logs
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: standard-thin
  resources:
    requests:
      storage: 100Mi
```

The related service definition that was used for exposing the necessary port is:

```
apiVersion: v1
kind: Service
metadata:
  name: data-handler
  labels:
    app: data-handler
spec:
  ports:
    - name: "8080"
      port: 8080
  selector:
    app: data-handler
```

Finally, the deployment definition which starts the necessary pod for the Data Collection Tool is depicted below:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: data-handler
spec:
  replicas: 1
  selector:
    matchLabels:
      app: data-handler
  strategy:
```

```
  type: Recreate
template:
  metadata:
    labels:
      app: data-handler
  spec:
    containers:
      - args:
          - bash
          - -c
          - sleep 40 && java -Xms256M -Xmx2048M -jar data-handler.jar
        env:
          - name: CONSUL_DISCOVERY_HOSTNAME
            valueFrom:
              configMapKeyRef:
                key: CONSUL_DISCOVERY_HOSTNAME
                name: env-handler-env
          - name: CONSUL_HOST
            valueFrom:
              configMapKeyRef:
                key: CONSUL_HOST
                name: env-handler-env
          - name: CONSUL_PORT
            valueFrom:
              configMapKeyRef:
                key: CONSUL_PORT
                name: env-handler-env
          - name: CRON_HEALTHENTIA
            valueFrom:
              configMapKeyRef:
                key: CRON_HEALTHENTIA
                name: env-handler-env
          - name: KAFKA_BOOTSTRAP_SERVERS
            valueFrom:
              configMapKeyRef:
                key: KAFKA_BOOTSTRAP_SERVERS
                name: env-handler-env
          - name: KAFKA_CONSUMER_CONCURRENCY
            valueFrom:
              configMapKeyRef:
                key: KAFKA_CONSUMER_CONCURRENCY
                name: env-handler-env
          - name: KAFKA_CONSUMER_GROUP_ID
            valueFrom:
              configMapKeyRef:
                key: KAFKA_CONSUMER_GROUP_ID
                name: env-handler-env
          - name: KAFKA_LISTENER_AUTO_STARTUP
            valueFrom:
              configMapKeyRef:
                key: KAFKA_LISTENER_AUTO_STARTUP
                name: env-handler-env
          - name: KAFKA_MINIO_TOPIC
            valueFrom:
              configMapKeyRef:
                key: KAFKA_MINIO_TOPIC
                name: env-handler-env
          - name: MINIO_ACCESS_KEY
            valueFrom:
              configMapKeyRef:
                key: MINIO_ACCESS_KEY
                name: env-handler-env
          - name: MINIO_ENDPOINT
            valueFrom:
```

```
          configMapKeyRef:
            key: MINIO_ENDPOINT
            name: env-handler-env
      - name: MINIO_SECRET_KEY
        valueFrom:
          configMapKeyRef:
            key: MINIO_SECRET_KEY
            name: env-handler-env
      - name: MINIO_WORKFLOW_BUCKET
        valueFrom:
          configMapKeyRef:
            key: MINIO_WORKFLOW_BUCKET
            name: env-handler-env
      - name: MONGODB_AUTHENTICATION_DATABASE
        valueFrom:
          configMapKeyRef:
            key: MONGODB_AUTHENTICATION_DATABASE
            name: env-handler-env
      - name: MONGODB_DATABASE
        valueFrom:
          configMapKeyRef:
            key: MONGODB_DATABASE
            name: env-handler-env
      - name: MONGODB_HOST
        valueFrom:
          configMapKeyRef:
            key: MONGODB_HOST
            name: env-handler-env
      - name: MONGODB_PASSWORD
        valueFrom:
          configMapKeyRef:
            key: MONGODB_PASSWORD
            name: env-handler-env
      - name: MONGODB_PORT
        valueFrom:
          configMapKeyRef:
            key: MONGODB_PORT
            name: env-handler-env
      - name: MONGODB_USERNAME
        valueFrom:
          configMapKeyRef:
            key: MONGODB_USERNAME
            name: env-handler-env
      - name: SPRING_PROFILES_ACTIVE
        valueFrom:
          configMapKeyRef:
            key: SPRING_PROFILES_ACTIVE
            name: env-handler-env
      - name: TZ
        valueFrom:
          configMapKeyRef:
            key: TZ
            name: env-handler-env
    image: harbor.infinitech-h2020.eu/data-management/data-handler:0.12.5
    name: infinitech-data-checkin-handler
    ports:
      - containerPort: 8080
  restartPolicy: Always
  volumes:
    - name: infinitech-data-checkin-handler-logs
      persistentVolumeClaim:
        claimName: infinitech-data-checkin-handler-logs
  imagePullSecrets:
    - name: registrysecret
```

As we can see, the configuration is held by the env-handler-env configmap, which is consumed by the pod definition through the env keyword.

The Regulatory Compliance Tools and the Anonymizer are very close to finalization and deployment. The Model Trainer is still being developed, and its deployment is scheduled for the end of Fall 2021.

3.3.3 Metrics in idle condition

The goal of the deployment on the NOVA sandbox is to provide an offline model training sandbox independent of the online system. This is important in real deployments, since the online system requires medium level resources all the time, while the model training system requires a lot of resources sporadically.

Since the Pilot #12 testbed on the NOVA sandbox is meant for offline operation, metrics on resource consumption are not so important. Also, metrics on model quality are irrelevant (the data quality and the model training algorithms are not controlled by the testbed). The metrics of interest have to do with the comparison of the model quality as the anonymization strength varies for the same input attributes and predicted outcomes.

3.4 Pilot #13 – Alternative/automated insurance risk selection -– product recommendation for SME

Pilot #13 will monitor risk changes, so it will be able to radically improve the risk management that companies (SMEs) face in the development of their daily activity. The indicators will be based on information from each of the companies coming from online sources that will give information about the digital presence and activity of those companies like activity, business volume, participation in social networks, number of employees, use of e-commerce, payment platform etc. The company to be analysed does not need to provide all of that information, as the developed tools are in charge of searching and gathering information related to the target company from many sources. In this way, risk profiles of each of the companies analysed will be generated, allowing to customize the product offering and to make a permanent automated risk management. But this is not the only usage of data; insurance companies will use these information sources, resulting -n better customized products. An overview of Pilot #13 is given in Figure 22.

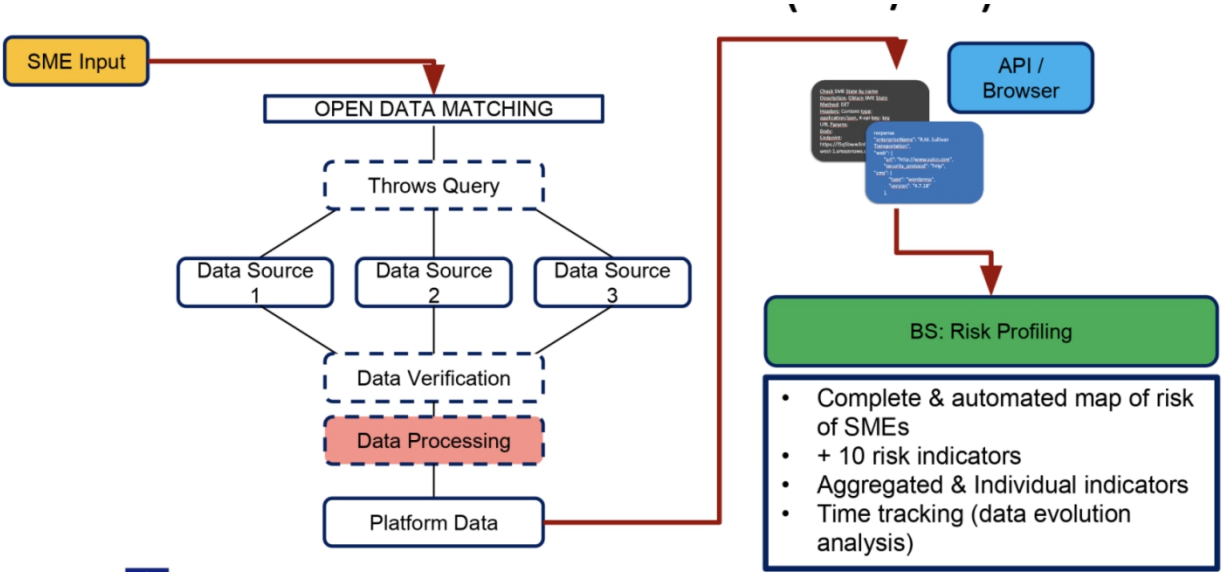


Figure 22 – Pilot#13 Overview

In the overall architecture, there can be identified three layers of building blocks that consists of the overall solution:

Data Acquisition Layer: This layer is used to obtain data from the information sources which are related with the digital presence and activities of the costumers, using automations developed by WEA. It is considered external to the INFINITECH sandbox and will feed the latter with the aggregated information after the initial pre-processing and preparation of the data.

Data Management Layer: This layer is used to store the data coming from the previous layer and allow for their efficient processing. It will make use of INFINITECH components developed in the project, and more precisely the INFINISTORE which includes the HTAP Data store along with its extensions with the polyglot engine.

Analytics Layer: This layer contains the AI algorithms and analytics that make use of the data available by the data management layer. These AI analytics will be developed by WEA and will be Pilot-specific, solving the needs of the Pilot by exploiting the technologies provided by INFINITECH for efficient data processing of data coming from a variety of heterogeneous resources. They will be hosted inside the WeAnalyze platform external to the deployed sandbox.

Visualization Layer: This layer contains the REST endpoints and visualization components that enable the end-users to see the results of the analysis via a User Interface, of by importing them to their own applications using the REST APIs. It will consume the results of the analytics layer which will be deployed in the premises of WEA, and therefore, there is no need for the components of this layer to be hosted inside the sandbox deployed in the NOVA infrastructure.

3.4.1 NOVA sandbox description

The integrated solution for Pilot #13 consists of several components that are related with the four different layers of the overall architecture, however, as it has been mentioned, only the technological building blocks that are related with the data management layer will be part of the INFINITECH sandbox, which will be deployed in the NOVA testbed. All other architectural components are considered external to the sandbox and will be hosted inside the Pilot premises or, in case of the data acquisition layer, directly to the source of the data.

As it has been described in the previous version of this deliverable (D6.10), the Pilot #13 sandbox will contain the INFINISTORE technological component. Therefore, the sandbox is defined by the following Kubernetes elements:

statefullset.apps/infinistore: This is the stateful set that contains the INFINISTORE data management system. As the INFINISTORE is a stateful component, a stateful set is considered the correct choice instead of a deployment config, due to the fact that the stateful sets preserve their internal IPs when a pod is being restarted. At this phase of the project and for the given data load, the requirement for computational power is 4 vCPUs with 16 GBs of memory, while the minimum requirement for a successful deployment is 2 vCPUs and 8 GBs of memory.

service/infinistore-service: This is a service element that allows the connectivity of the INFINISTORE stateful set with other elements inside the sandbox. Even if the sandbox of Pilot #13 contains only the INFINISTORE, this service is needed for all internal components to be able to reach each other.

service/infinistore-np: This is a node port that allows the connectivity with components that are external to the sandbox via the internet. The Pilot #13 integrated solution consists of several layers like the visualization, analytics, and data acquisition, to be able to connect to the INFINISTORE that lies inside the deployed sandbox. As a result, this node port exposes the defined ports of the INFINISTORE to the internet.

perstistent.volume.claim/infinistore-datasets-pvc: This is the persistent volume claim that is needed to persistently store the ingested data of the datastore, so that it can be available every

time the datastore restarts. At this phase of the project, the requirement is for 100 GBs of storage, to validate that the integrated solution is working with a medium data load.

3.4.2 Second-stage components deployment on NOVA

From the description of the sandbox for Pilot #13 that has been provided in the previous subsection, it is evident that after deployment and during the runtime, there will be pods that are only related to the stateful set that corresponds to INFINISTORE, and they would need to be accessed from external components. At this phase of the project, the main focus is the development and validation of the overall integrated solution of Pilot #13, without stressing the solution with intense data load. As a result, the deployment of the INFINISTORE will be in an all-in-one fashion. This means that all internal components of the INFINISTORE itself have been deployed in a single node. As a result, the deployed sandbox consists of a single pod, which instantiates the stateful set of INFINSTORE:

pod/infinistore-0: This is the pod that contains all internal components of INFINISTORE.

As mentioned in the previous subsection, for the components of INFINISTORE to be able to reach each other, there has been defined a service that exposes the corresponding ports. The following code snippet of the infinistore-service depicts the ports that need to be reachable:

```
spec:
  ports:
    - name: "2181"
      port: 2181
      targetPort: 2181
    - name: "1529"
      port: 1529
      targetPort: 1529
    - name: "9876"
      port: 9876
      targetPort: 9876
    - name: "9992"
      port: 9992
      targetPort: 9992
    - name: "14400"
      port: 14400
      targetPort: 14400
    - name: "9800"
      port: 9800
      targetPort: 9800
  selector:
    app: infinistore
```

Most of these ports are intended to be reached only by the internal components that consist of the Pilot #13 sandbox. However, another requirement is for the INFINISTORE to be reachable from external components. Therefore, a node port has been defined that exposes only specific ports (the port where the query engine listens to) to external components, as depicted in the following code snippet:

```
spec:
  type: NodePort
  selector:
    app: infinistore
  ports:
    - name: "1529"
      protocol: TCP
```

```
    port: 1529
    targetPort: 1529
    nodePort: 31301
```

Here, we can see that the port where the query engine listens to, its default is 1529, is being mapped to the target port 31301. That means that external components need to open a JDBC connection to the external IP of the Kubernetes cluster on the port 31301. Then the Kubernetes cluster will forward this connection to the specific pod, the infinistore as defined by the appropriate selector in the code snippet, mapped to its 1529 that the pod has exposed internally.

Last but not least, the internal components of INFINISTORE needs to connect themselves, therefore, they would need the corresponding hostname. As this is being given during the deployment phase and cannot be known a priori, an environment variable needs to be passed, giving to it the value of the service that exposes the corresponding ports, as it can be depicted in the following code snippet of the INFINISTORE stateful set:

```
spec:
  serviceName: infinistore-service
    spec:
      - containers:
        - image: harbor.infinitech-h2020.eu/data-management/infinistore:latest
          name: infinistore
          ports:
            - containerPort: 2181
            - containerPort: 1529
            - containerPort: 9876
            - containerPort: 9992
            - containerPort: 14400
            - containerPort: 9800
      volumes:
        - name: infinistore-datasets-storage
          persistentVolumeClaim:
            claimName: infinistore-datasets-pvc
          volumeMounts:
            - name: infinistore-datasets-storage
              mountPath: /datasets
          env:
          - name: USEIP
            value: "yes"
          - name: KVPEXTERNALIP
            value: "infinistore-service!9800"
```

In this code snippet, we can see that the environment variable KVPEXTERNALIP is given the hostname and port, where the hostname is the one that matches the serviceName that is the name of the service that allows for network connectivity inside the sandbox. It can be also depicted that there is the definition of the persistent volume claim that was described earlier, that will be mounted under the /datasets folder of the pod, so that the INFINISTORE can persistently store its data there.

3.4.3 Metrics in idle condition

Pilot #13 cluster standard metrics in idle condition result as follows:

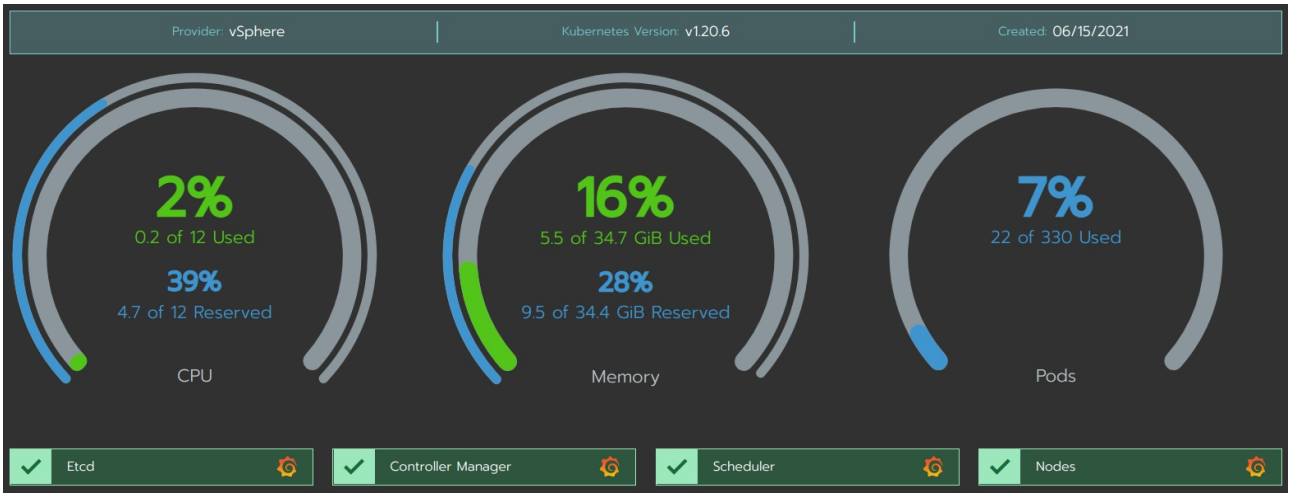


Figure 23 – Resources utilization by Pilot#13

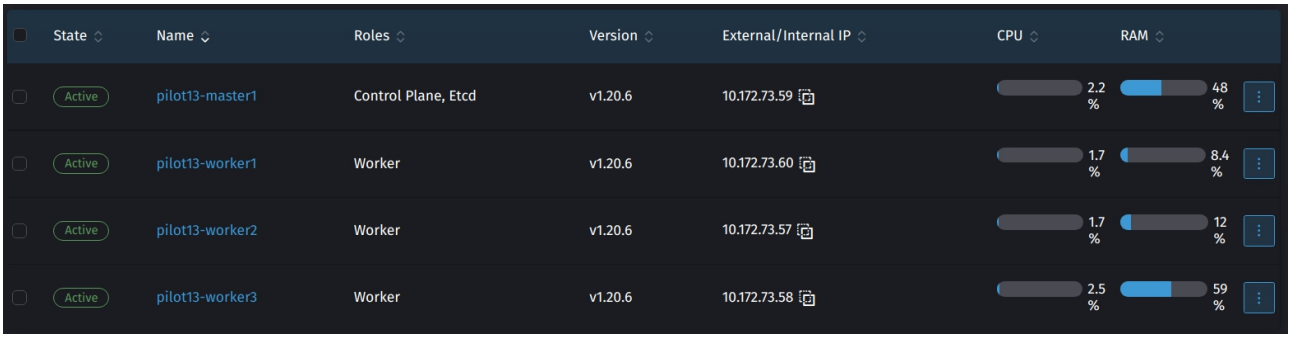


Figure 24 – Pilot#13 nodes list

3.5 Pilot#14 – Big Data and IoT for the Agricultural Insurance Industry

AgroApps is developing the entire infrastructure for the Pilot #14 data products, based on the reference architecture starting from data collection from different sources, over processing and analytics, to user interface & data visualization. The ongoing development of the service module is based on scientific research in the field of agricultural insurance, climate & weather risk modelling and the most recent evolutions, in the area of remote sensing technologies.

3.5.1 NOVA sandbox description

AgroApps' Weather Intelligence engine as described in previous deliverable (see D6.10) is the only set of software hosted on the NOVA testbed due to an extrinsic constraint. The instances of AgroApps Weather Engine are launched as Statefulsets while all the other components will only have Kubernetes Services of kind ExternalName. These services, initialized into Pilot #14's namespace, will work as network references to AgroApp's managed premises, and allow every sandbox module to deliver requests and receive responses from the remote infrastructure. For the Pilot #14 and the weather service, 12 nodes with 244 cores and 177 GB RAM are utilized on the Kubernetes Cluster.

3.5.2 Second-stage components deployment on NOVA

The Weather Intelligence service provided by AGROAPPS in the framework of INFINITECH is produced by operating the Weather Research and Forecasting Model (WRF), a meso-scale numerical weather prediction model. It consists of a dynamic Eulerian kernel (ARW), subroutines for pre-processing static geographical data and dynamic atmospheric fields, to determine the initial condition and lateral boundary conditions of the integration domains, as well as subroutines for post-processing the forecast data.

The operational workflow, which is depicted in Figure 25, employs two docker images:

1. wrf_all_user, which contain geogrid.exe, ungrib.exe, metgrid.exe, real.exe and wrf.exe
2. dtc-upp, which contains unipost.exe

The flowchart depicts the volumes/external data used by the model and mainly refers to atmospheric fields of analysis and forecasts from other numerical weather forecasting models, the system of pre-treatment of these data and their interference with the horizontal and vertical grid of the model. The ARW kernel of the forecasting models solves the primitive equations and integrates in time the atmospheric circulation. At the last level, the post-processing of the predicted atmospheric fields takes place.

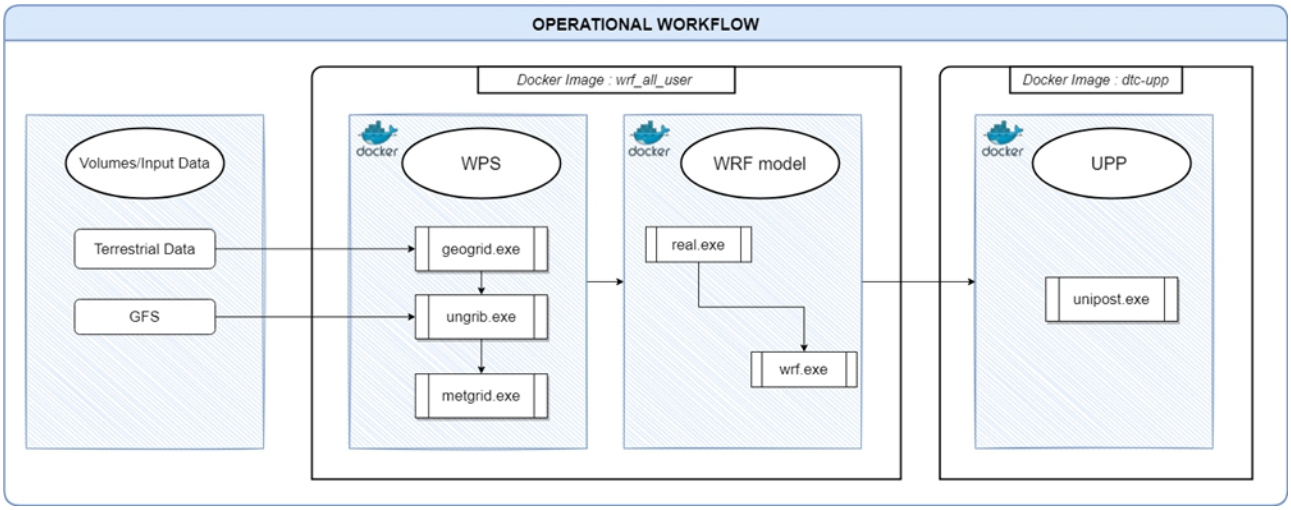


Figure 25 – Flowchart of the WRF-ARW modelling system.

AGROAPPS's deployed numerical weather prediction system consists of 3 basic elements:

1. The pre-processing part of the modelling system: This is a 3-step process that includes:
   a. geogrid.exe: Defines the integration domains and interpolates the static terrestrial data to the model grids. Terrestrial data are included with 20 categories of land use by the Moderate Resolution Imaging Spectroradiometer (MODIS) while terrain is defined by the U.S. Geological Survey (USGS) topographic maps. In areas where CORINE data are not available, the model uses data from the existing WRF land use database, USGS 1997 [19] (see Table 4). High spatial resolution SRTM [20] and SoilGrids [21] data are used to determine topography and soil mechanical properties, respectively.

Table 4 – Grouping and matching CORINE land use classes to USGS classes

CORINE Code	Category Description	USGS Code	USGS Category Description

CORINE Code	Category Description	USGS Code	USGS Category Description
11	Urban	1	Urban and Built-Up Land
12	Non-Irrigated Arable Land	2	Dryland Cropland and Pasture
13	Permanently Irrigated Land	3	Irrigated Cropland and Pasture
14	Rice Fields	3	
15	Vineyards	6	Crops/Wood mosaic
16	Fruit Trees And Berry Plantations	6	
17	Olive Growes	6	
18	Pastures	2	Dryland Cropland and Pasture
19	Annual Crops & Permanent Crops	6	
20	Complex Cultivation Patterns	6	
21	Mixed Agriculture & Natural Vegetation	6	Crops/Wood mosaic
22	Agro-Forestry Areas	6	
23	Broad-Leaved Forest	11	Deciduous Broadleaf Forest
24	Coniferous Forest	14	Evergreen Needleleaf Forest
25	Mixed Forest	15	Mixed Forest
26	Natural Grassland	7	Grassland
27	Moors & Heathland	9	Mix Shrubland/Grassland
28	Sclerophyllous Vegetation	9	
29	Transitional Woodland-Shrub	9	
30	Beaches, Dunes & Sand Plains	19	
31	Bare Rock	19	Barren or Sparsely Vegetated
32	Sparsely Vegetated Areas	19	
33	Burnt Areas	19	
34	Glaciers & Perpetual Snow	24	Snow or Ice
35-38	Inland Marshes, Peatbogs, Salines	17	Herbaceous Wetlands
39	Intertidal Flats	17	
40-43	Inland Water	16	Water Bodies
44	Sea & Ocean	16	

b. ungrib.exe: Extracts meteorological fields from the Global Forecasting System (GFS) data.

c. For the definition of the initial and lateral boundary conditions, the 00 UTC forecasts of the NCEP/GFS [22] analysis and three-hour forecasts are used. Their spatial resolution is 0.25°x0.25° (latitude - longitude) and are operationally available from NCEP [ftp://ftpprd.ncep.noaa.gov]. Additionally, high resolution data, which are operationally available from NCEP [ftp://ftpprd.ncep.noaa.gov] with spatial resolution of 1/12°x1/12° (about 0.083°x0.083°), are used to determine surface sea temperatures.

d. metgrid.exe: Includes the horizontal interpolation of gridded meteorological data from the GFS of a half a degree resolution in the simulation domain [23].

2. The model forecasting: The meteorological data are then vertically interpolated to the defined WRF's levels (with real.exe).

   The model is integrated forward in time to produce with an hourly step weather forecast (with wrf.exe). AGROAPPS operates the numerical weather prediction model WRF version 4.1.3 inside a docker container, producing high-resolution hourly forecasts for the next 8 and 3.5 days, providing atmospheric fields according to the meteorological data requirements.

   The WRF numerical weather prediction model provides a multitude of configurations of physical processes occurring in the atmosphere and on the surface of the earth in terms of solar radiation, cloud microphysics, convective cloud development, planetary boundary layer, surface layer, and soil thermodynamics and hydrology.

   To represent cloud microphysics, the model uses Thomson et al. (2008) [24] microphysics scheme which supports 5 different types of hydrometeors and is ideal for high resolution simulations, while in terms of the cumulus convection scheme, the model uses the KF-CuP [25] which is active only in the external integration domain. In the nested domains, no cumulus convection parameterization scheme is used because the spatial resolution of the grid allows the small-scale turbulence to be directly resolved from the microphysical scheme.

   To configure the radiation for both long and short-wave radiation components, the model uses the RRTMG [26] scheme, with a 10 minute run time and the slope_rad option enabled allowing the radiation scheme to be taken into account various calculations of topography and slopes of the soil.

   The planetary boundary layer is configured by the Yonsei University [27] physical parameterization scheme with the topographic correction options for surface winds and vertical mixing due to heat loss and radiation enabled. The revised Monin-Obukov MM5 [28] scheme is used for the parameterization of the atmospheric boundary layer, while the NOAH-MP [29] scheme is used for the parameterization of the hydrological and thermodynamic soil processes.

   Finally, to calculate a multitude of diagnostic variables such as the diameter of hailstones, the existence and velocity of a tornado, etc., the AFWA diagnostic tool suite is used.

3. The post-processing part of the modelling system: The output forecasted meteorological fields include two-meter temperature, relative humidity, wind speed and precipitation among the more than 100 fields available from the model. The raw output of the weather forecast model is in Network Common Data Form (netCDF) format. The raw data from the WRF model are post-processed with the Unified Post Processor (UPP) version 4.1 inside a docker container (with unipost.exe), in order to produce de-staggered grids and diagnostic variables. UPP performs the following 2-steps process for the prognostic data:

   a. Generate GRIB format files per forecast time for each model grid, where the primary results are first converted from Arakawa staggered C-grid to a de-staggered A-grid, and then inserted into a regular grid with constant step per geographic width and length (regular latitude-longitude grid). Normal grids are sub-regions of the four model integration domains and have a spatial resolution of 18km x 18km, 6km x 6km and 2km x 2km for the external domain and the nested ones, respectively.

b. Calculation of diagnostic atmospheric fields which are inserted along with the atmospheric prognostic fields in GRIB files. The operationally generated atmospheric are the following:

   i. Air Temperature at 2m Height
   ii. Relative Humidity at 2m Height
   iii. Wind Speed and Direction at 10m Height
   iv. Total Precipitation and Precipitation Rate
   v. Reference Evapotranspiration
   vi. Precipitation Types: Hail, Graupel, Ice Pellets and Snow
   vii. Snow Height, density, and cover
   viii. Solar Radiation

3.5.3 Metrics in idle condition

The status of the testbed in an "awaiting for workload" condition is reported in the following image:

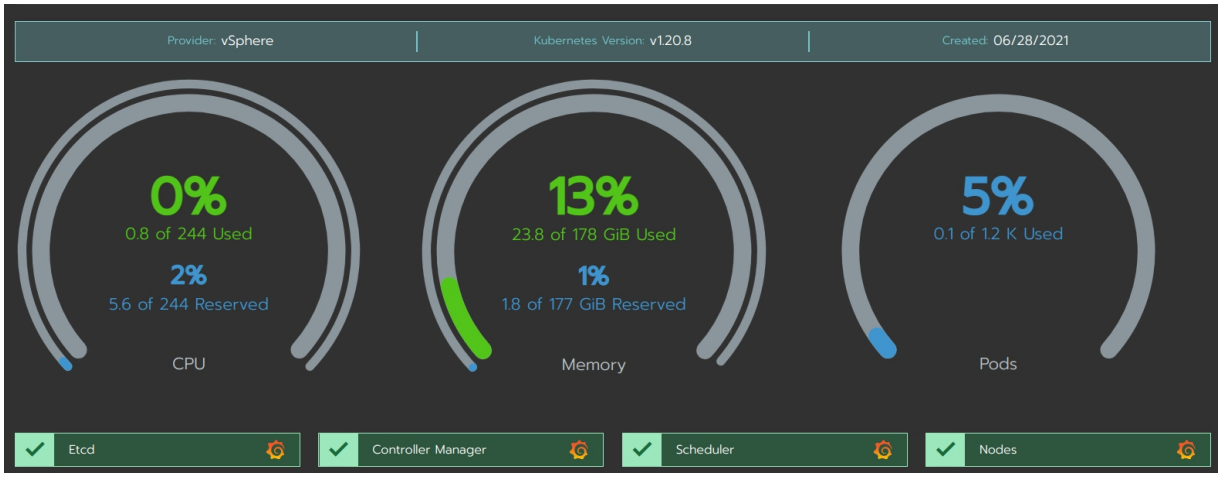
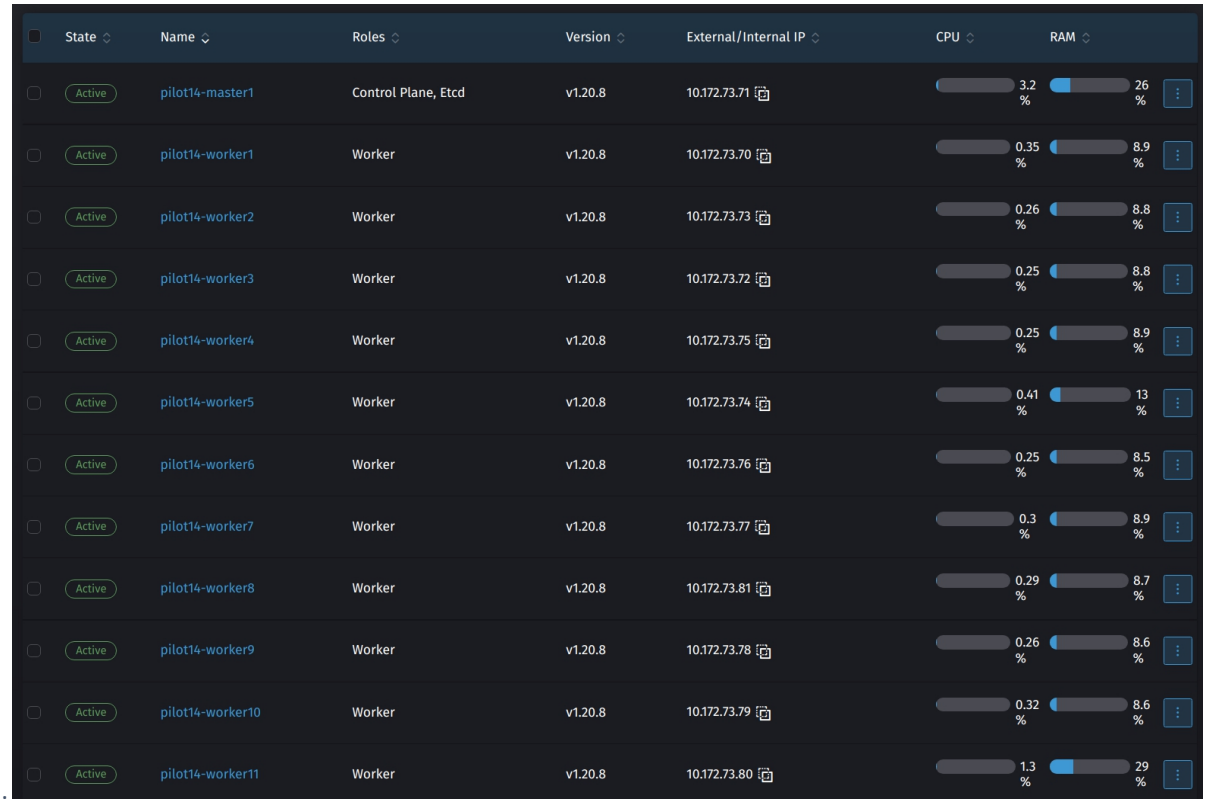


Figure 26 – Resources utilization by Pilot #14



State	Name	Roles	Version	External/Internal IP	CPU	RAM
Active	pilot14-master1	Control Plane, Etcd	v1.20.8	10.172.73.71	3.2 %	26 %
Active	pilot14-worker1	Worker	v1.20.8	10.172.73.70	0.35 %	8.9 %
Active	pilot14-worker2	Worker	v1.20.8	10.172.73.73	0.26 %	8.8 %
Active	pilot14-worker3	Worker	v1.20.8	10.172.73.72	0.25 %	8.8 %
Active	pilot14-worker4	Worker	v1.20.8	10.172.73.75	0.25 %	8.9 %
Active	pilot14-worker5	Worker	v1.20.8	10.172.73.74	0.41 %	13 %
Active	pilot14-worker6	Worker	v1.20.8	10.172.73.76	0.25 %	8.5 %
Active	pilot14-worker7	Worker	v1.20.8	10.172.73.77	0.3 %	8.9 %
Active	pilot14-worker8	Worker	v1.20.8	10.172.73.81	0.29 %	8.7 %
Active	pilot14-worker9	Worker	v1.20.8	10.172.73.78	0.26 %	8.6 %
Active	pilot14-worker10	Worker	v1.20.8	10.172.73.79	0.32 %	8.6 %
Active	pilot14-worker11	Worker	v1.20.8	10.172.73.80	1.3 %	29 %

Figure 27 – Pilot #14 nodes list.

4 Conclusions

One of the key aspects that emerges from the previous pages is how the task T6.5 is extremely focused on enabling the Pilots to move to the “INFINITECH way” software development methodology, exploiting a dedicated infrastructure represented by the NOVA testbed.

During this second iteration, the most important cloud-native INFINITECH artifacts and blueprints have been produced and consumed to launch and maintain Pilot’s sandboxes on NOVA, and more are up to come, counting 26 Kubernetes deployment assets and 40+ configuration objects. The Continuous Delivery pipelines are in place, and they are ready to reference the NOVA infrastructure to proceed with the automation behind the rolling updates. This is the intermediate stage of the whole task picture, where the partners and the Pilots have migrated the software successfully, all the benchmarking tools are ready and all the pre-requirements are in place for the next iteration, in which the main subject will be a full CI/CD compliance, the analysis of metrics, and the final optimizations.

The documents D6.10, D6.11 and D6.12 are an explicative reference for every professional who wants to approach to the “INFINITECH way” using an on-premise infrastructure as target environment, as the content can be considered as a consistent migration guide from a legacy approach to a rapid prototyping and time saver development process.

Appendix A: Literature

[1] «Kubernetes,» [Online]. Available: https://kubernetes.io.

[2] «Packer,» [Online]. Available: https://www.packer.io/.

[3] «Docker,» [Online]. Available: https://www.docker.com.

[4] «RKE,» [Online]. Available: https://rancher.com/products/rke/ .

[5] «CNCF,» [Online]. Available: https://www.cncf.io.

[6] «HELM,» [Online]. Available: https://helm.sh/.

[7] «Nginx,» [Online]. Available: https://nginx.org/.

[8] «Prometheus,» [Online]. Available: https://prometheus.io/.

[9] «Grafana,» [Online]. Available: https://grafana.com/.

[10] Rancher, «Rancher Metrics,» [Online]. Available: https://rancher.com/docs/rancher/v2.x/en/monitoring-alerting/v2.0.x-v2.4.x/cluster-monitoring/cluster-metrics/.

[11] https://www.fiware.org/developers/smart-data-models/

[12] https://fiware-orion.readthedocs.io/en/master/

[13] https://fiware.github.io/specifications/ngsiv2/stable/

[14] https://quantumleap.readthedocs.io/en/latest/

[15] https://app.swaggerhub.com/apis/smartsdk/ngsi-tsdb/0.1

[16] https://konghq.com/kong/

[17] https://sumo.dlr.de/docs/index.html

[18] https://opendata.aemet.es/

[19] Links, Skip. "High-Resolution Land Use and Land Cover Mapping".

[20] https://cgiarcsi.community/data/srtm-90m-digital-elevation-database-v4-1/

[21] https://soilgrids.org/#!/?layer=TAXNWRB_250m&vector=1

[22] https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/global-forcast-system-gfs

[23] NCEP. (2015). NCEP GFS 0.25 Degree Global Forecast Grids Historical Archive. Research Data Archive at the National Center for Atmospheric Research, Computational and Information Systems Laboratory. . U.S.: Weather Service/NOAA/U.S.

[24] Thompson, Gregory, et al. "Explicit forecasts of winter precipitation using an improved bulk microphysics scheme. Part II: Implementation of a new snow parameterization." Monthly Weather Review 136.12 (2008): 5095-5115.

[25] Berg, Larry K., et al. "Evaluation of a modified scheme for shallow convection: Implementation of CuP and case studies." Monthly Weather Review 141.1 (2013): 134-147.

[26] Iacono, Michael J., et al. "Radiative forcing by long-lived greenhouse gases: Calculations with the AER radiative transfer models." Journal of Geophysical Research: Atmospheres113.D13 (2008).

[27] Hong, Song-You, Yign Noh, and Jimy Dudhia. "A new vertical diffusion package with an explicit treatment of entrainment processes." Monthly weather review 134.9 (2006): 2318-2341.

[28] Jiménez, Pedro A., et al. "A revised scheme for the WRF surface layer formulation." Monthly Weather Review 140.3 (2012): 898-918.

[29] Niu, Guo‐Yue, et al. "The community Noah land surface model with multiparameterization options (Noah‐MP): 1. Model description and evaluation with local scale measurements." Journal of Geophysical Research: Atmospheres 116.D12 (2011).

[30] https://rancher.com/docs/rancher/v2.5/en/installation/requirements/#hardware-requirements