Ref. Ares(2022)5050004 - 11/07/2022

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

D6.12 – Sandboxes for FinTech and InsuranceTech Innovators - III

Revision Number	3.0
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	2022-06-30
Delivered Date	2022-07-11
Internal Reviewers	NOVA, CCA
Quality Assurance	INNOV
Acceptance	Coordinator Accepted
EC Project Officer	Beatrice Plazzotta
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

© INFINITECH Consortium

Contributing Partners

Partner Acronym	Role[1]	Author(s)[2]
ENG	Lead Beneficiary	Domenico Messina – Susanna Bonura
HPE	Contributor	
NOVA	Contributor	
JRC	Contributor	
ATOS	Contributor	
WEA	Contributor	
AGRO	Contributor	
GFT	Contributor	
UBI	Contributor	
LXS	Contributor	
CTAG	Contributor	
CPH	Contributor	
DYN	Contributor	
FTS	Contributor	
GEN	Contributor	
PRIVE	Contributor	
RB	Contributor	
UNP	Contributor	
UPRC	Contributor	
INNOV	Contributor	
NOVA, CCA	Internal reviewers	
INNOV	Quality Assurance	

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

[2] Can be left void

Revision History

Version	Date	Partner(s)	Description
0.1	2022-05-05	ENG and contrib. partners	ToC Version
0.2	2022-05-20	AGRO	Contribution merged
0.3	2022-05-27	INNOV	Contribution merged
0.4	2022-05-31	ATOS, UBI	Contribution merged
0.5	2022-06-10	WEA, LXS	Contribution merged
0.6	2022-06-16	CTAG	Contribution merged
1.0	2022-06-27	ENG	First Version ready for Internal Peer Review
1.1	2022-06-30	NOVA, CCA, INNOV	Version after Internal Peer Review and QA
2.0	2022-07-05	HPE, GFT	Overall review
3.0	2022-07-11	ENG	Version for Submission

Executive Summary

D6.12 - Sandboxes for FinTech and InsuranceTech innovators III, is the third and last report of its series, concluding the walk-through for the process of tailoring sandboxes using a dedicated infrastructure.

It addresses the subject of the environment observability, which is one of the main objectives of the latter part of the activities within task T6.5. Observability is one of the key aspects for keeping distributed systems healthy and guaranteeing the availability of the services that run within our environment, in the specific case of sandboxes hosted in the NOVE infrastructure. It allows users to deduct performance characteristics from the internal components that belong to the system, detect bottlenecks, issues, and take mitigation and fix actions fast. From the report, several aspects related to the software runtime environment can be analyzed, as well as some anomalies in the processes that support the software development. For example, if a specific build step takes too long to complete, it could be a symptom of bad practice or an issue in the CI/CD automation. The study aimed to detect possible bottlenecks, overcommitted nodes within the cluster used by the pilots, and resources under pressure; this will represent the starting point for mitigating the issue and solving the problem that may occur more than once across all the sandboxes if, the same critical conditions will show up during the environment lifetime.

The data collected, relating to this aspect, reveal that pilot and tech proxies have been able to run their software without inconveniences or restrictions imposed by the system, which is yet another demonstration of the possibility to tailor dedicated INFINITECH Sandboxes in an on-premise environment, such as Nova's testbed, without any hassle. INFINITECH's NOVA testbed runs a constellation of software providing an observable environment for microservice-based and cloud-native architecture, adopting cloud computing best practices and appropriate runtime isolation mechanisms, making it a good reference example for easily reproducing such a complex system on other infrastructures so that they can be ready to be used, in line with the INFINITECH reference architecture and compliant with the "INFINITECH way" principles.

Data have been collected from the sandboxes running in the NOVA testbed and are accessible from the URL https://rancher.vps.uninova.pt/g/clusters . The administrative account has the privileges needed to explore each pilot sandbox under the technical perspective, managing resources and runtime environments, and for what is required regarding the CI/CD stack, timelines and trends, as can be explored in detail from: https://jenkins.infinitech-h2020.eu.

1 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

2 Introduction

D6.12 Sandboxes for FinTech and InsuranceTech innovators III, is the third and last report of its series, concluding the walk-through for the process of tailoring sandboxes to a dedicated infrastructure. In the previous two document releases, a description of the testbed bootstrap, and the sandbox build planning, mapping, and implementation was presented. The only missing phase, to conclude the whole process, is about the environment observability, which is one of the main objectives of the latter part of the activities within task T6.5 as well as one of the subjects exposed in the next few pages. The supporting software adopted for this purpose can be reached via the websites https://rancher.vps.uninova.pt/g/clusters and https://jenkins.infinitech-h2020.eu .

2.1 Objective of the Deliverable

This deliverable is a report that digs into sandbox analytics and answers questions like "how are sandboxes running?", "Are they performing well?", "could they be improved?" and "are they supporting our developers in the right way?". Observability, detectability, monitoring, and analysis are fundamental tasks to keep a distributed system alive and, most importantly, healthy and available. Sandbox tailoring adopts these principles, therefore a series of analytics will be executed using some specific tools that have been installed into the testbed and that will be exploited to gather and study all the pieces of information to keep constant track of all the live workload objects that are running within the sandbox. The study aims to detect possible bottlenecks, overcommitted nodes within the cluster used by the pilots, and resources under pressure; this will represent the starting point for mitigating the issue and solving any problem that may occur more than once across all the sandboxes if, the same critical conditions will show up during the environment lifetime. Once again, all the NOVA sandboxes will be taken into consideration in order to see if there are specific emerging needs which, in case, will be exposed in the sandbox-dedicated section.

2.2 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.

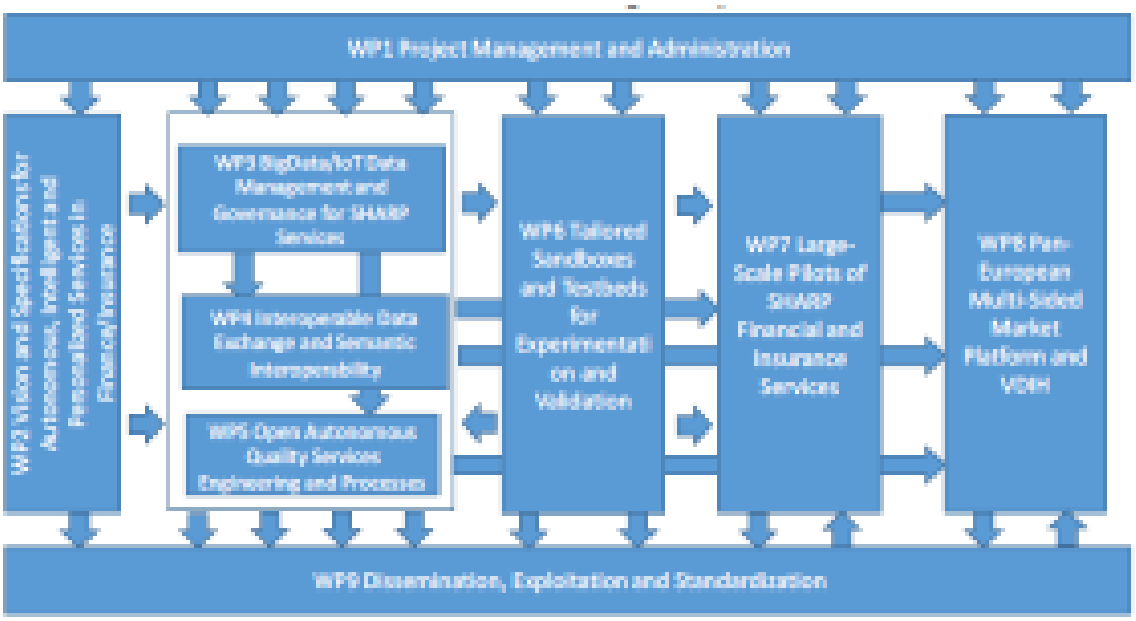

Figure 1 - INFINITECH Work Breakdown Structure

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.

2.3 Structure

Besides the introductory chapter that provides the overview and the objectives of this report, the deliverable is articulated into two other chapters: "INFINITECH in-cluster sandbox analytics" and "Optimizations and environment improvements". The first of the two chapters is a per-sandbox analysis that starts with the cluster topology overview so that the reader can have it always in sight, it introduces a list and a description of the active Kubernetes workload objects coming with Kubernetes-related metrics, it exposes statistics and resource consumption trends and finally, it proposes the analysis of the CI/CD related context with timeline and trends inspection.

The "Optimizations and environment improvements" chapter is about overcoming possible issues detected and possible known issues that may occur within INFINITECH sandboxes during their lifetime

Last but not least, the concluding chapter will be a recap of the overall work behind the observability, detectability, and analysis tasks to draw the necessary remarks.

3 INFINITECH in-cluster sandbox analytics

All the tools used for scraping metrics from the sandbox have already been presented in chapter 2 of the deliverable *D6.11 – Sandboxes for FinTech and InsuranceTech Innovators III*. For the sake of a quick recap, the Rancher dashboard and the Jenkins dashboard are used for compiling such a report. Rancher dashboard works off the shelf with the Prometheus metrics system and the Grafana monitoring UI which are running in the testbed. Both the dashboards provide an overview first and all the required details on demand if the users want to go deeper in the exploration of the analytics. For example on the rancher dashboard, once one of the clusters is selected, the system provides the overview:

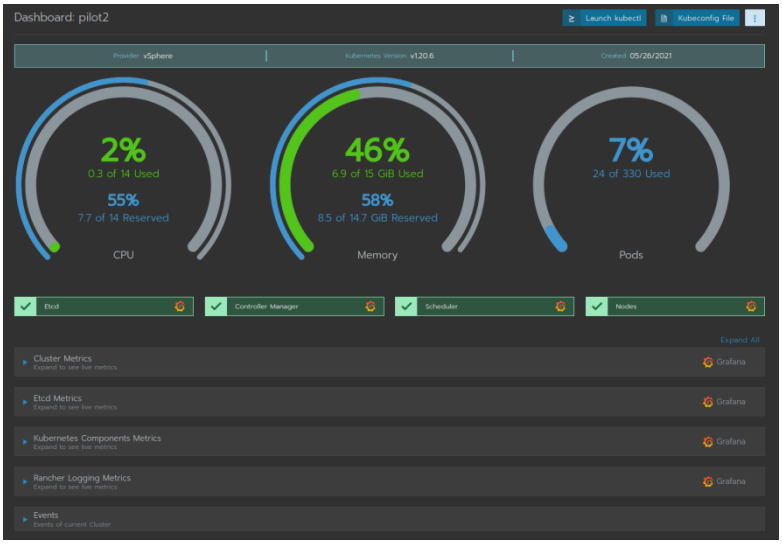

Figure 2 - Rancher's landing dashboard

Then, the user can explore in-depth the specific aspects she wants to investigate, such as the node load average over time:

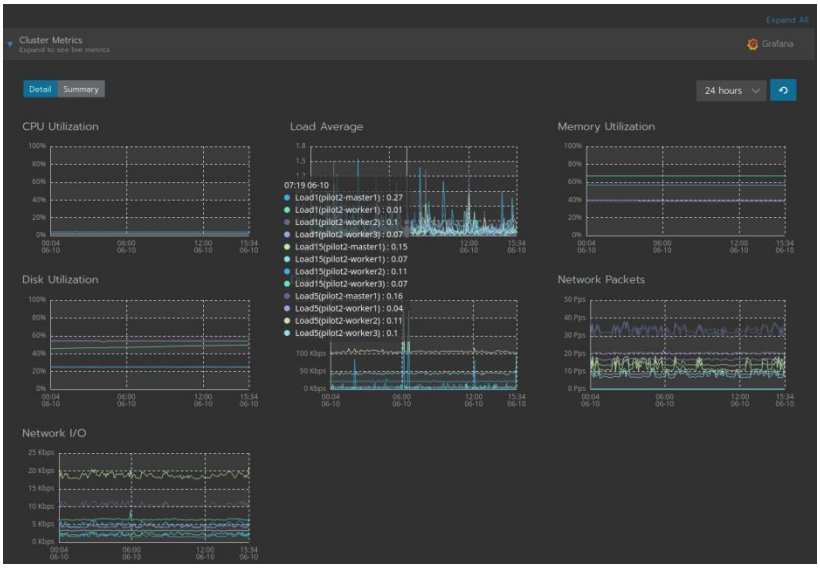

Figure 3 - Rancher's cluster metrics dashboard

The user can go further to inspect the Kubernetes objects (such as the running pods) using the cluster explorer.

Figure 4 - Rancher's Kubernetes dashboard

Moving on to the Jenkins dashboard, to get access to pipelines' statistics, users can click on a specific pipeline to get access to the execution history in the stage view.

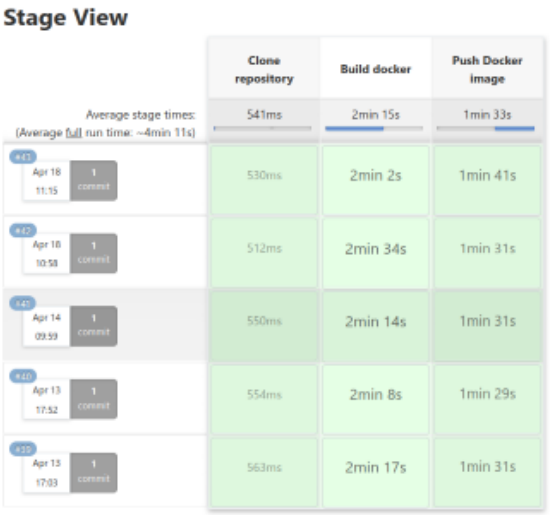

Figure 5 - Jenkins' stage view

The *stage* view shows a table with time-related details of the stage execution. The table is composed of trigger events in the row entries. For each entry, the duration of the steps is reported and the average stage time is highlighted right below the header of the table.

3.1 Pilot#2 Real-time risk assessment in Investment Banking: scraping metrics

3.1.1 Cluster topology

Pilot#2 uses a Kubernetes v1.20.6 cluster made of one control plane node and 3 worker nodes interconnected. They have the following characteristics:

Table 1 - Pilot#2 cluster nodes

Name	Role	CPU (vCPU)	Ram (GiB)
pilot2-master1	Control Plane	4	3.7
pilot2-worker1	Worker	4	3.7
pilot2-worker2	Worker	4	3.7
pilot2-worker3	Worker	6	7.6

In terms of overall average workload running in the cluster as a whole, considering also the resource consumed by the support software necessary to get the Kubernetes environment running properly, plus the container for all the pluggable capabilities, such as metrics servers, network sidecar containers etc. the situational picture is the following:

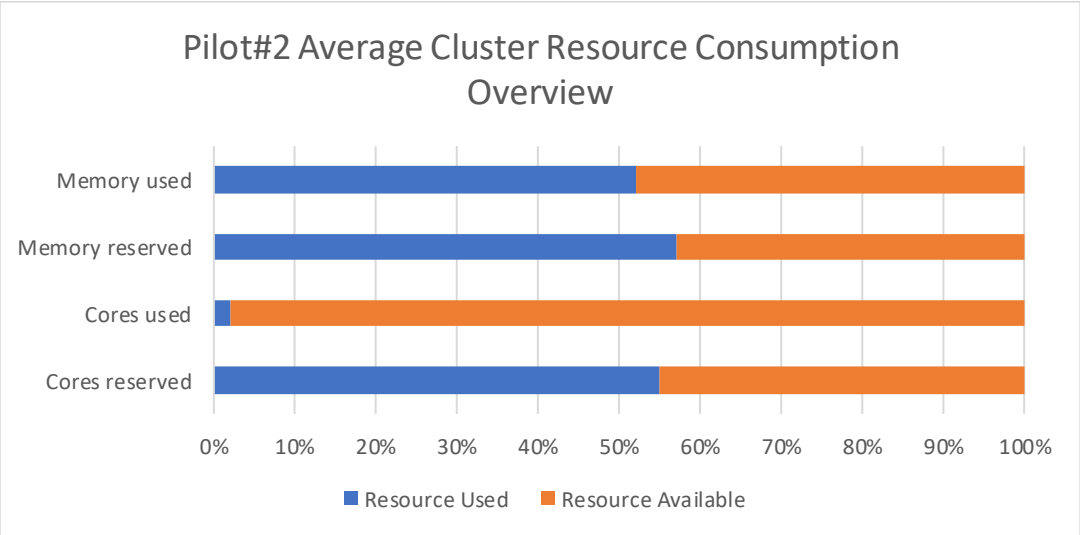

Figure 6 - Pilot#2 average cluster resource consumption overview

During heavy workload times, core consumption spans between 50% and 70%, with memory utilisation exceeding 80%. However, the Infinitech Way DevOps enables timely resource allocations among the deployed sandboxes according to their needs.

3.1.2 Active Kubernetes objects

The status of the Kubernetes objects declared and applied for pilot#2 is recapped in the chart below:

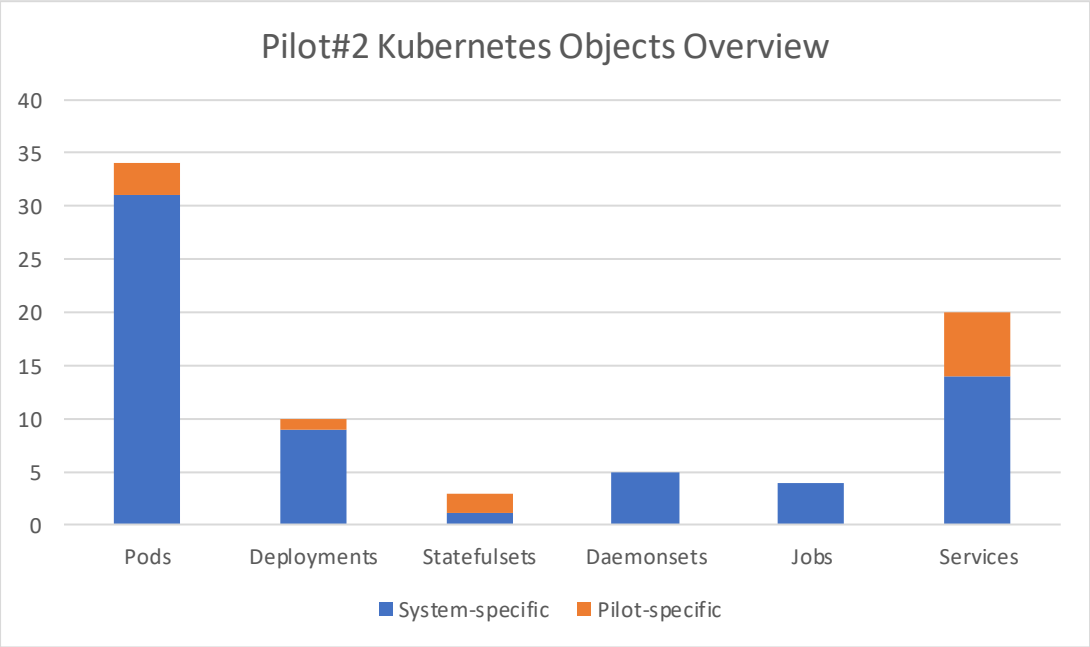

Figure 7 - Pilot#2 Kubernetes objects overview

3.1.3 Statistics and resource consumption

To have a look in detail into the running deployment unit, the pods, here is a comparison of the CPU request and CPU cap for the pilot-specific running pods in the cluster:

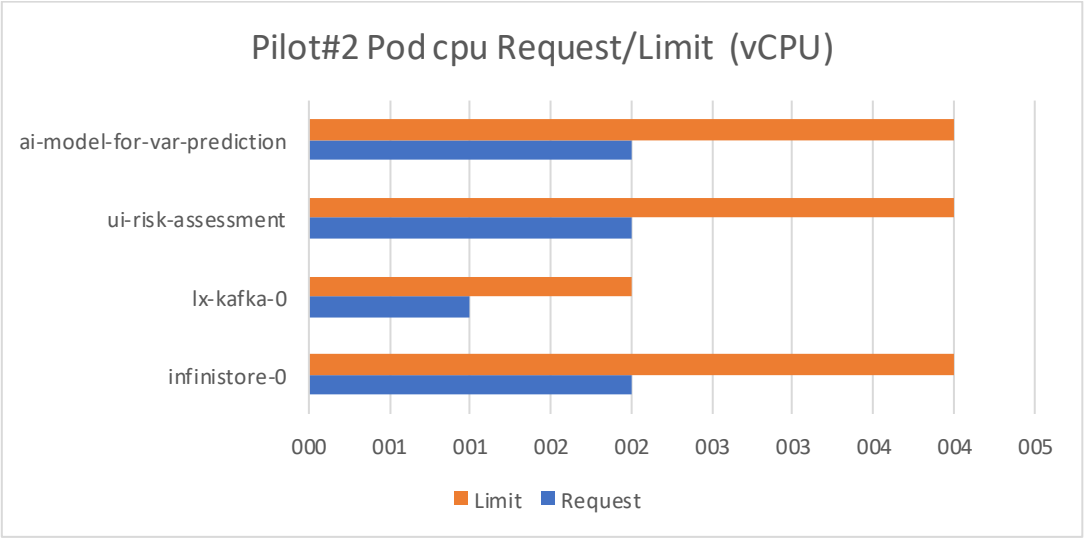

Figure 8 - Pilot#2 Pod cpu request/limit

The same comparison can be checked regarding the memory requests and limits:

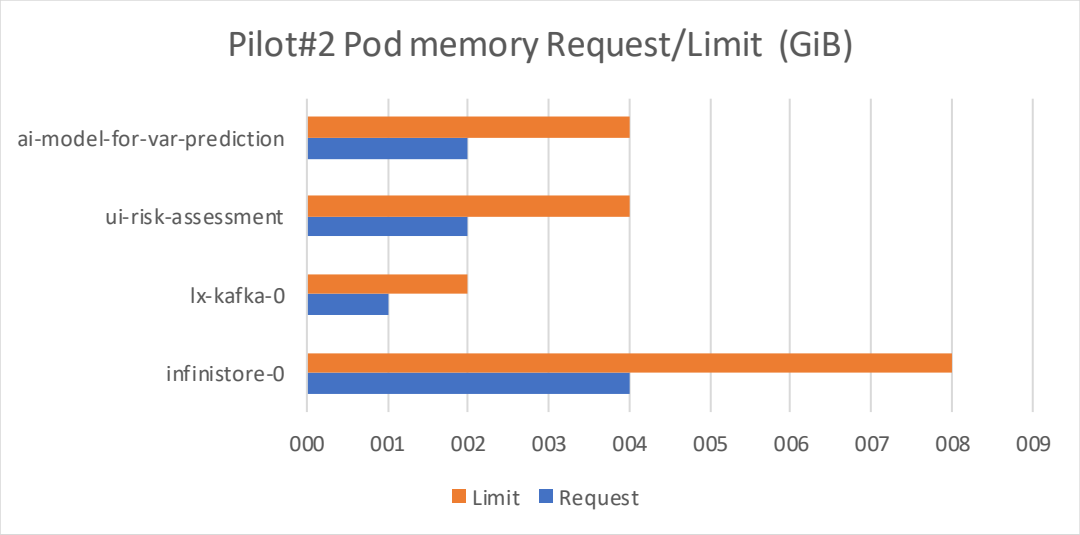

Figure 9 - Pilot#2 Pod memory request/limit

Here is the disk and network I/O summary in a 30-days time window:

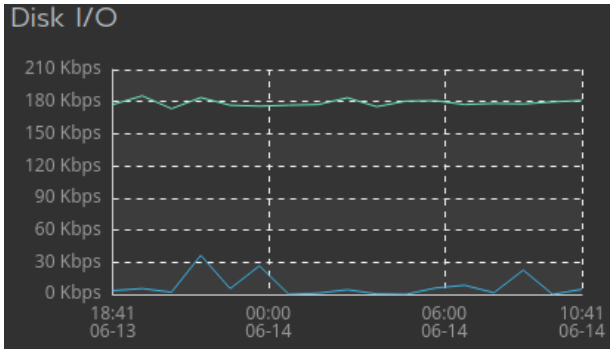

Figure 10 - Pilot#2 disk I/O trend

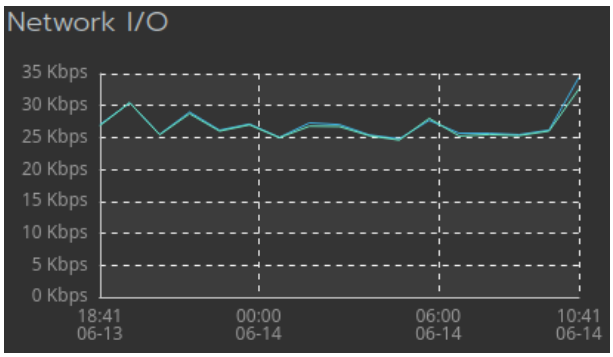

Figure 11 - Pilot#2 network I/O trend

3.1.4 CI/CD timeline and trends

Pilot#2 has 2 active complete pipelines in the INFINITECH's Jenkins instance split into three steps: clone repository, build docker, push docker image, and deploy at NOVA infrastructure. Jenkins reports the following average stage runtimes for each pipeline:

1) Deploy-ai-model-for-var-prediction has a stage of roughly 7 minutes.

2) Deploy-ui-risk-assessment-based-on-var has a stage of 9min and 20s.

In addition, there are 2 more pipelines only for deploying Infinistore and Kafka instances in Pilot#2 cluster. At the same time, the Data Management layer of Infinitech manages the respective pipelines for building these services. Since the artifacts of Infinistore and Kafka are already built, their deployment typically requires less than 2 seconds.

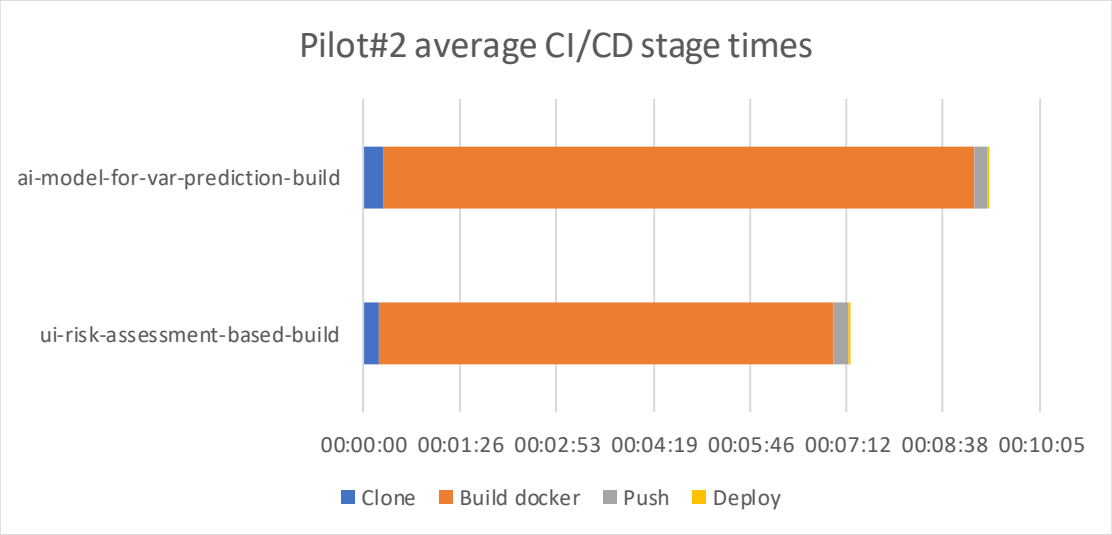

Figure 12 - Pilot#2 average CI/CD stage times

3.2 Pilot#11 - Personalized insurance products based on IoT connected vehicles: scraping metrics

3.2.1 Cluster topology

Pilot#11 relies on an internally managed infrastructure to run and achieve the pilot's targets. This isolated instance is due to the characteristics of some of the datasets managed to build the AI-powered driving profiling tool. Specifically, the data from real drivers used to train the models are considered sensitive information, and it is preferred to be kept within the strict control of the pilots' partners. In ay case, this internal deployment, described in detail in D6.11, follows strictly the INFINTECH guidelines, based on:

· Kubernetes V1.18.15 cluster, with 5 nodes, 24 cores, 44 GB of RAM, and 4TB of storage capabilities.

· Docker images, distributed through an open repository, to manage the services and updates of the Smart Fleet components.

· Internal GitLab (managed by ATOS) to collaborate in the code development of each of the services composing the Pilot#11 plus supporting the CI/CD pipelines.

In parallel to its internal framework, Pilot#11 supports a clone of the Smart Fleet framework and the AI Driving Profiling model deployed in the INFINITECH (NOVA) premises, including the same set of services and components (same versions and same docker images) running on the main instance, but without the datasets coming from the real drivers. This testbed instance is used internally by Pilot#11 partners to simulate big traffic datasets (using the SUMO instance and the NGSI adaptors) and for external INFINITECH partners to access potential useful datasets and to explore the capabilities of the Pilot#11 framework. The data shown in this section refers to this testbed infrastructure.

Pilot#11 Testbed uses a Kubernetes v1.20.8 cluster made of one control plane node plus 5 worker nodes interconnected, with the following schema:

Table 2 - Pilot#11 cluster nodes

Name	Role	CPU (vCPU)	Ram (GiB)
pilot11-master1	Control Plane	6	7.6
pilot11-worker1	Worker	6	17
pilot11-worker2	Worker	6	17
pilot11-worker3	Worker	6	17
pilot11-worker4	Worker	6	17
pilot11-worker5	Worker	6	17

On its basic load, this testbed runs the service that captures periodically (every thirty minutes) and stores internally the weather information reported by the weather stations involved in the pilot. Additionally, it also replicates, from the main instance, the information from traffic alerts. These basic (and permanent) workload is shown in the figure below.

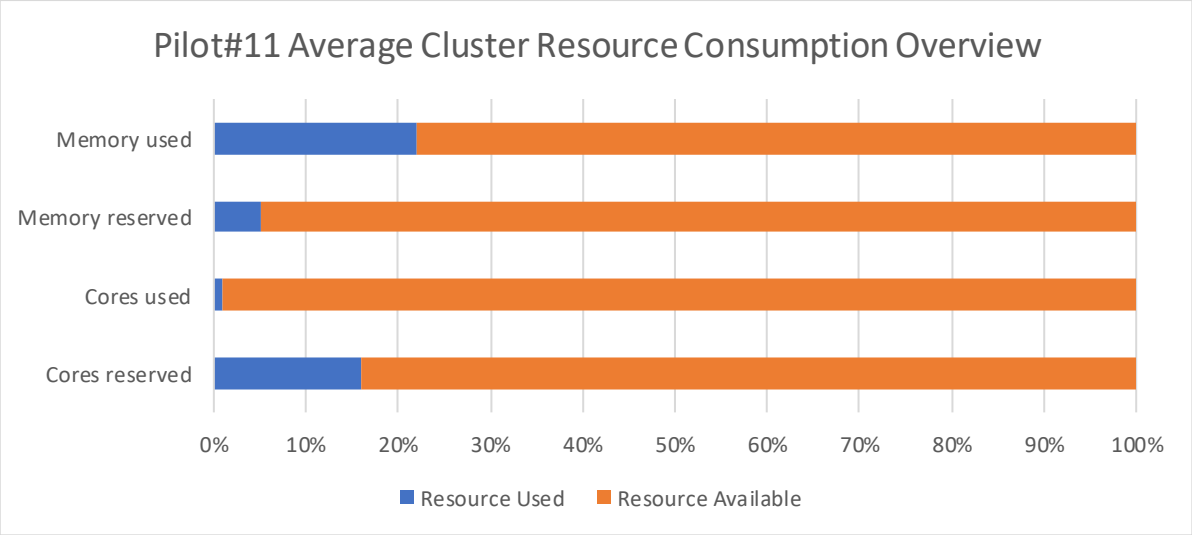

Figure 13 - Pilot#11 average cluster resource consumption overview

Additionally, the testbed supports occasionally some traffic simulations from its SUMO (Urban Mobility Simulator customized version) that increase substantially the overall workload, stressing the context broker and the database storage by generating 2k new registries per second. This pushes the overall memory used to 60-70% and the used cores to 50%.

3.2.2 Active Kubernetes objects

The figure below summarises the set of Kubernetes objects declared and applied for pilot#11, which represents the components shown in D6.11 for the Smart Fleet framework.

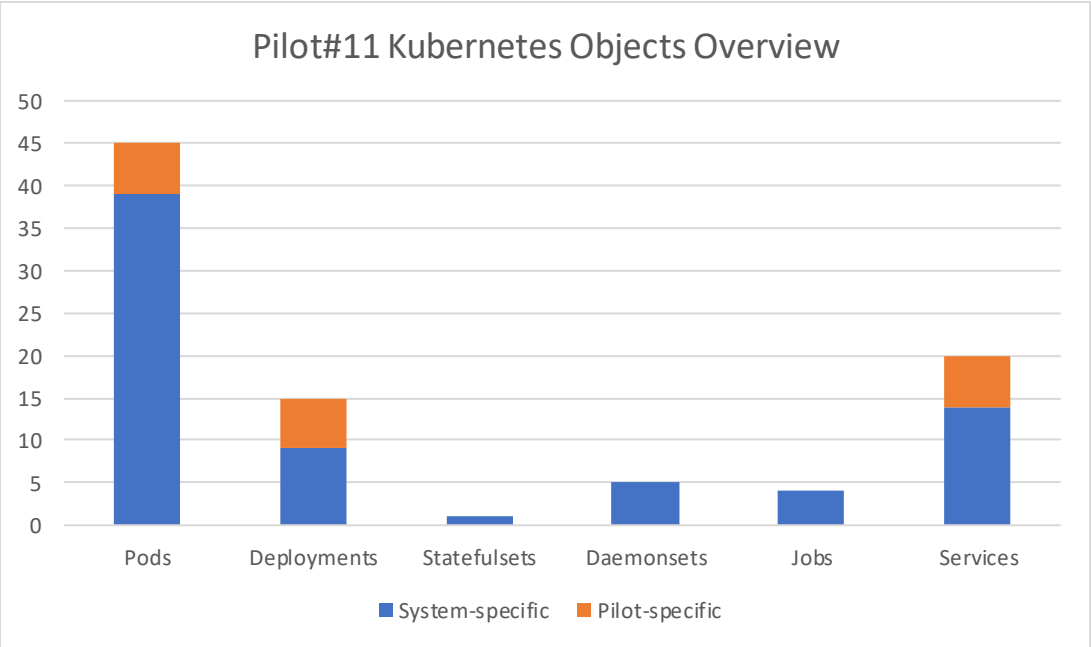

Figure 14 - Pilot#11 Kubernetes objects overview

3.2.3 Statistics and resource consumption

The following figures represent the main components of the Smart Fleet running on the Pilot#11 NOVA tested. In terms of cores required, the "sumoserver" component that represents the Urban Mobility Simulator (SUMO) is the most consuming one, as it requires extra processing power to perform simulations. The rest of the components represents a low but constant workload, but, in the case of the CrateDB service, require extra RAM memory to ensure the proper process of the data queues to store all historical data.

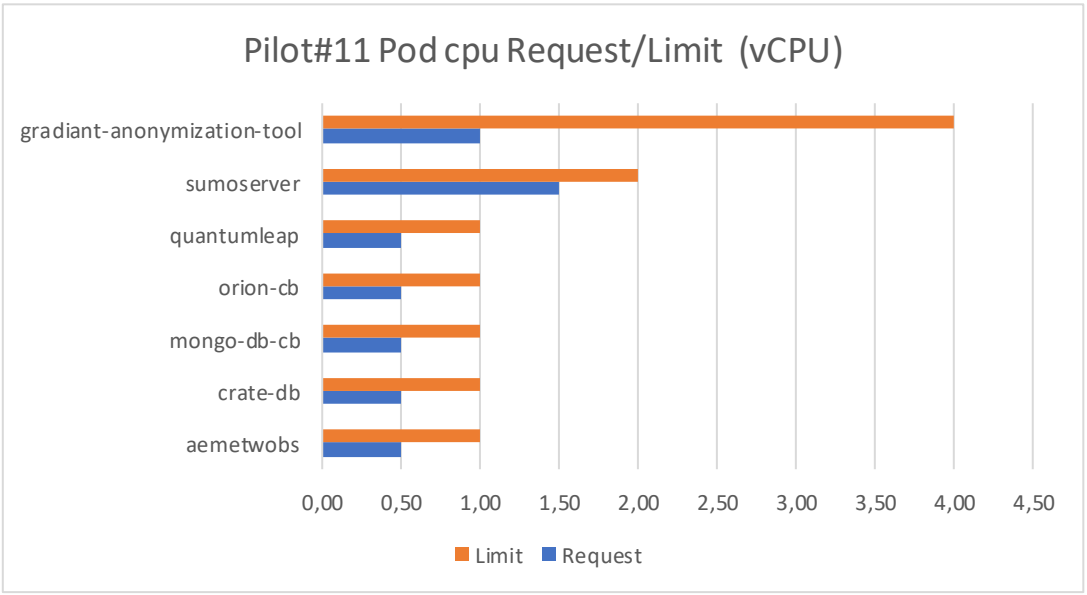

Figure 15 - Pilot#11 Pod cpu request/limit

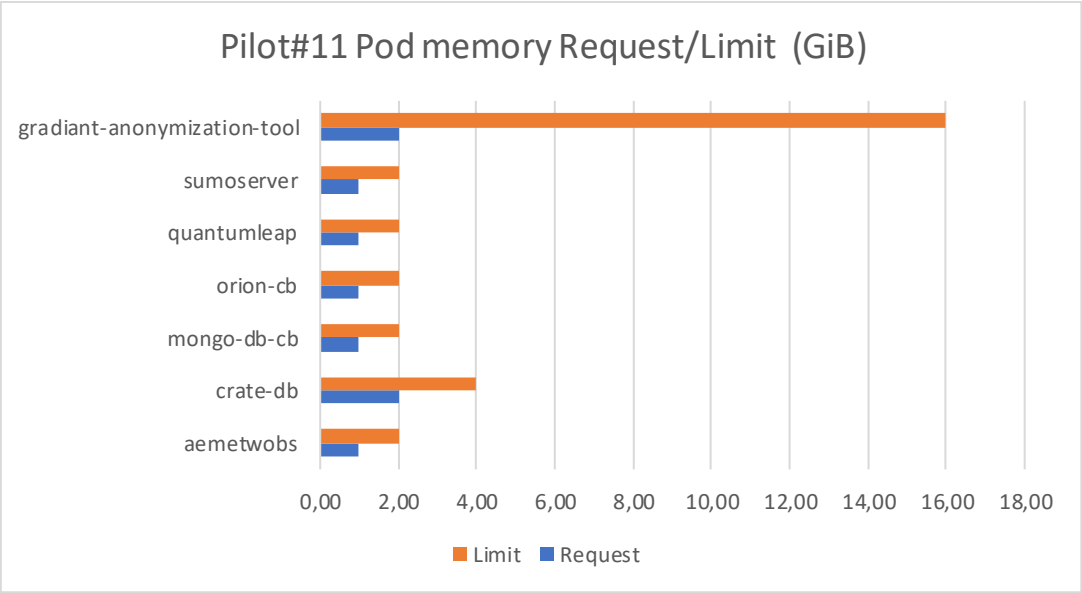

Figure 16 - Pilot#11 Pod memory request/limit

3.2.4 CI/CD timeline and trends

In the case of Pilot#11, as mentioned in the section 3.2.1, the main environment is running in a private premise, alongside a private GitLab instance for hosting the code. The cloned environment in NOVA, which is available to run simulations, replicates the weather and the alerting system and it gets updated modifying the YAML kubernetes specification files created in the sandbox.

3.3 Pilot#12 - Real World Data for Novel Health-Insurance products: scraping metrics

3.3.1 Cluster topology

Pilot#12 uses a Kubernetes v1.20.6 cluster made of one control plane node and 5 worker nodes interconnected. They have the following characteristics:

Table 3 - Pilot#12 cluster nodes

Name	Role	CPU (vCPU)	Ram (GiB)
pilot12-master1	Control Plane	6	7.6
pilot12-worker1	Worker	4	16
pilot12-worker2	Worker	4	16
pilot12-worker3	Worker	4	16
pilot12-worker4	Worker	4	16
pilot12-worker5	Worker	4	3.7

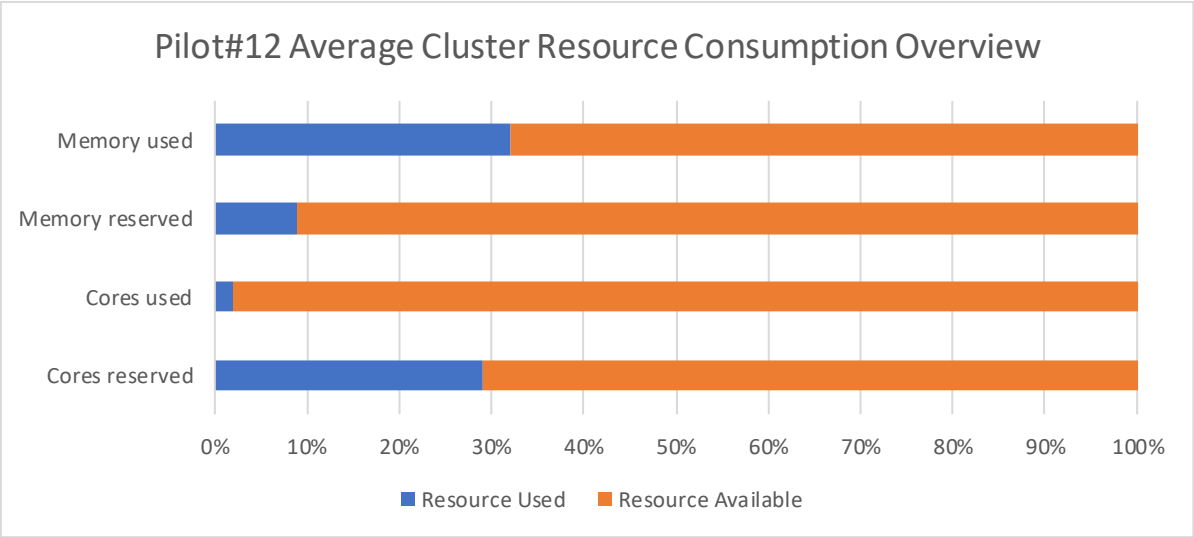

Figure 17 - Pilot#12 average cluster resource consumption overview

3.3.2 Active Kubernetes objects

The status of the Kubernetes objects declared and applied for pilot#12 is presented in the chart below:

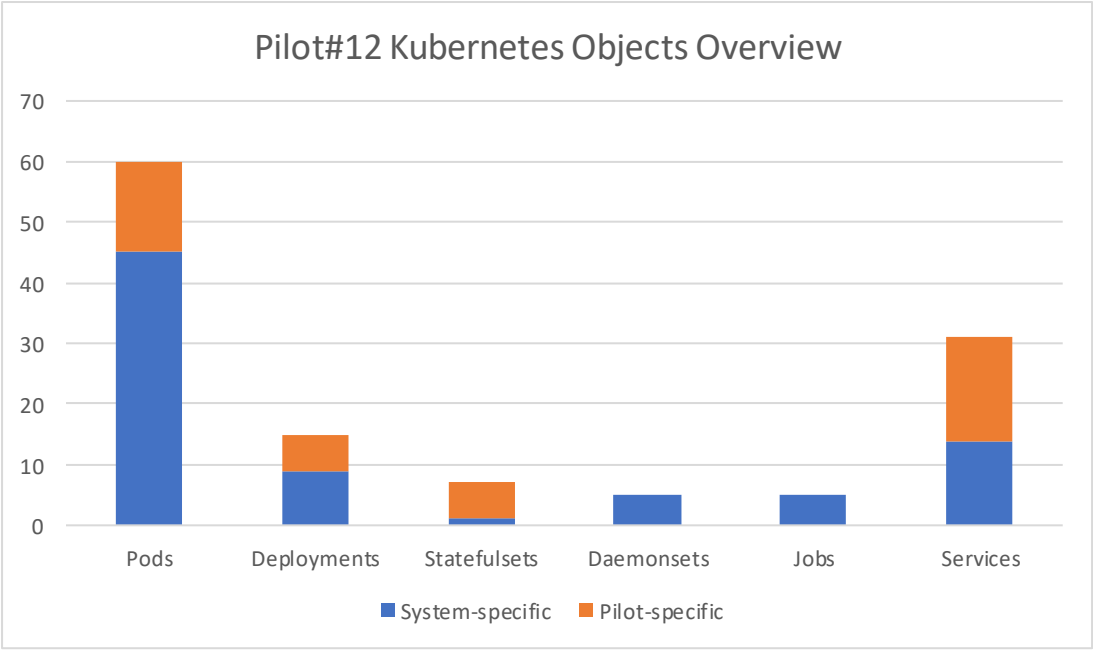

Figure 18 - Pilot#12 Kubernetes objects overview

3.3.3 Statistics and resource consumption

Having a look at the running pods for pilot#12, here is a comparison of the CPU requests and CPU limits:

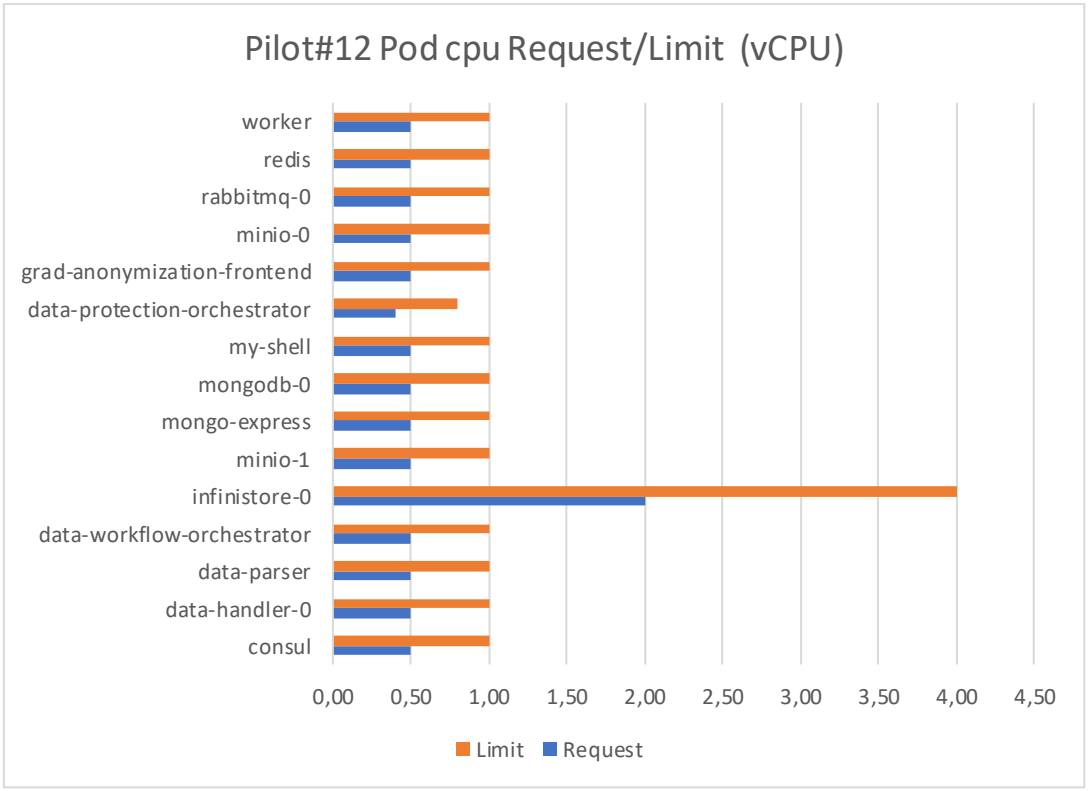

Figure 19 - Pilot#12 Pod cpu request/limit

The same comparison can be performed regarding the memory requests and limits:

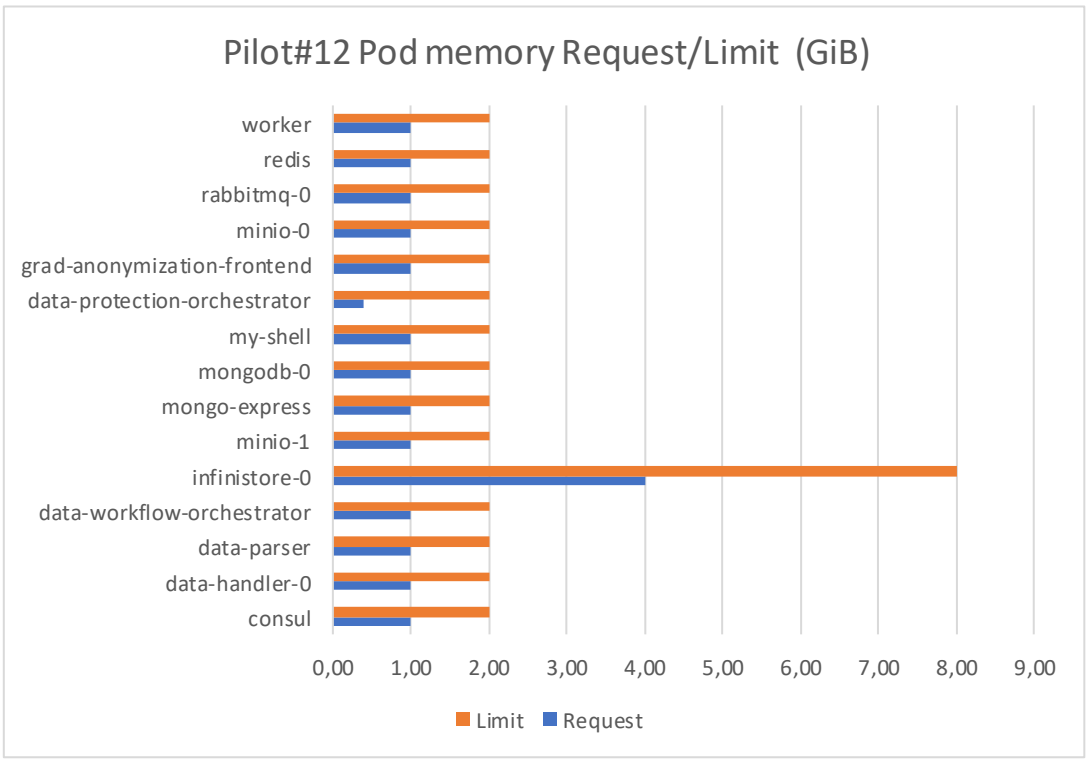

Figure 20 - Pilot#12 Pod memory request/limit

3.3.4 CI/CD timeline and trends

Pilot#12 has 3 active pipelines in INFINITECH's Jenkins instance:

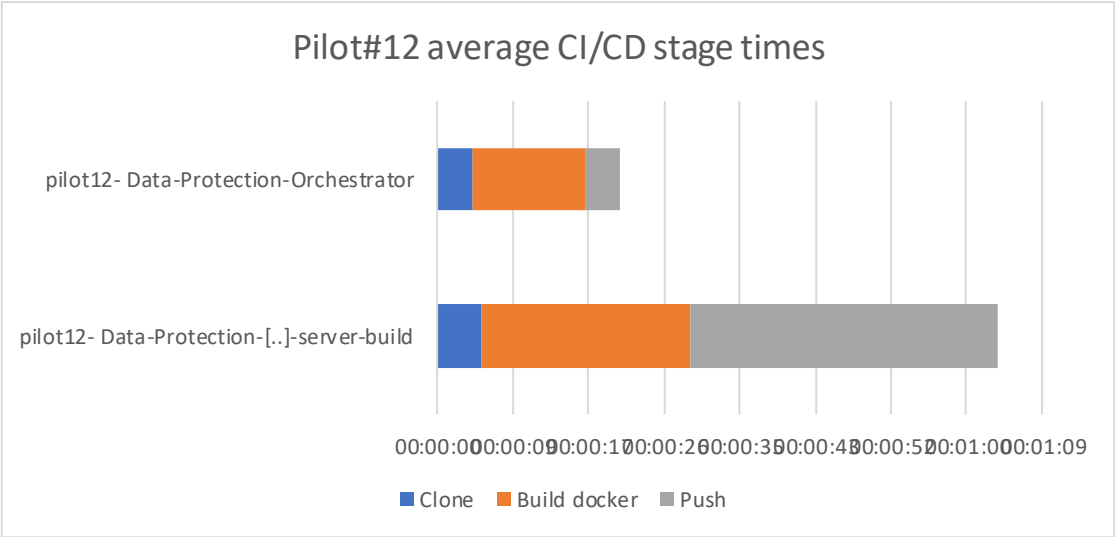

Figure 21 - Pilot#12 average CI/CD stage times

3.4 Pilot#13 - Alternative/automated insurance risk selection - product recommendation for SME: scraping metrics

3.4.1 Cluster topology

Pilot#13 uses a Kubernetes v1.20.6 cluster made of one control plane node and 5 worker nodes interconnected. They have the following characteristics:

Table 4 - Pilot#13 cluster nodes

Name	Role	CPU (vCPU)	Ram (GiB)
pilot13-master1	Control Plane	6	7.6
pilot13-worker1	Worker	4	16
pilot13-worker2	Worker	4	16
pilot13-worker3	Worker	4	3.7
pilot13-worker4	Worker	6	7.6
pilot13-worker5	Worker	6	7.6

In terms of overall average workload running in the cluster as a whole, considering also the resource consumed by the support software necessary to get the Kubernetes environment running properly, plus the container for all the pluggable capabilities, such as metrics servers, network sidecar containers etc. the situational picture is the following:

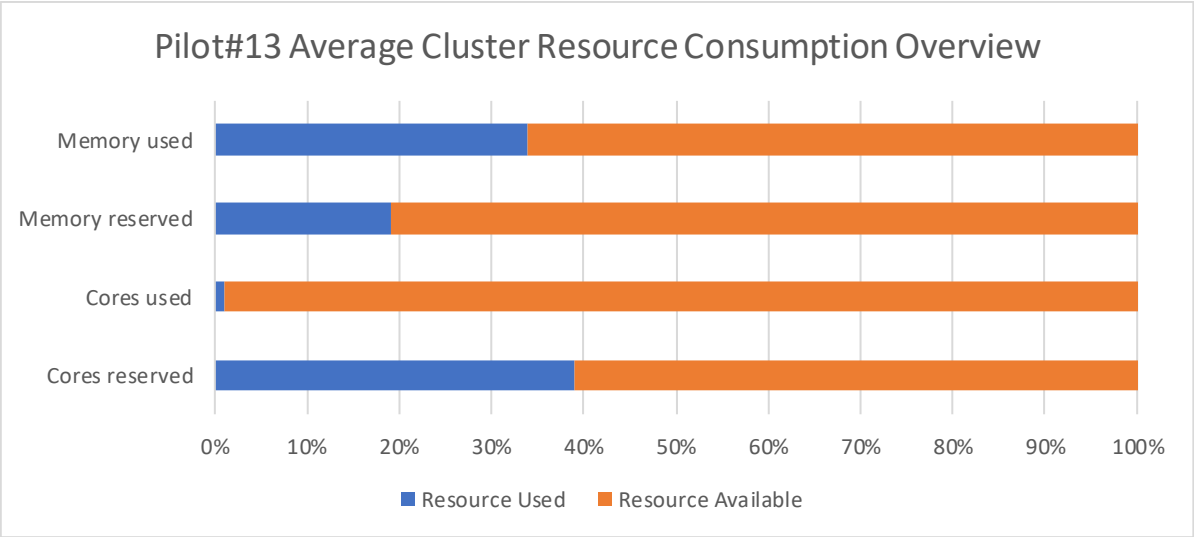

Figure 22 - Pilot#13 average cluster resource consumption overview

3.4.2 Active Kubernetes objects

The status of the Kubernetes objects declared and applied for pilot#13 is recapped in the chart below:

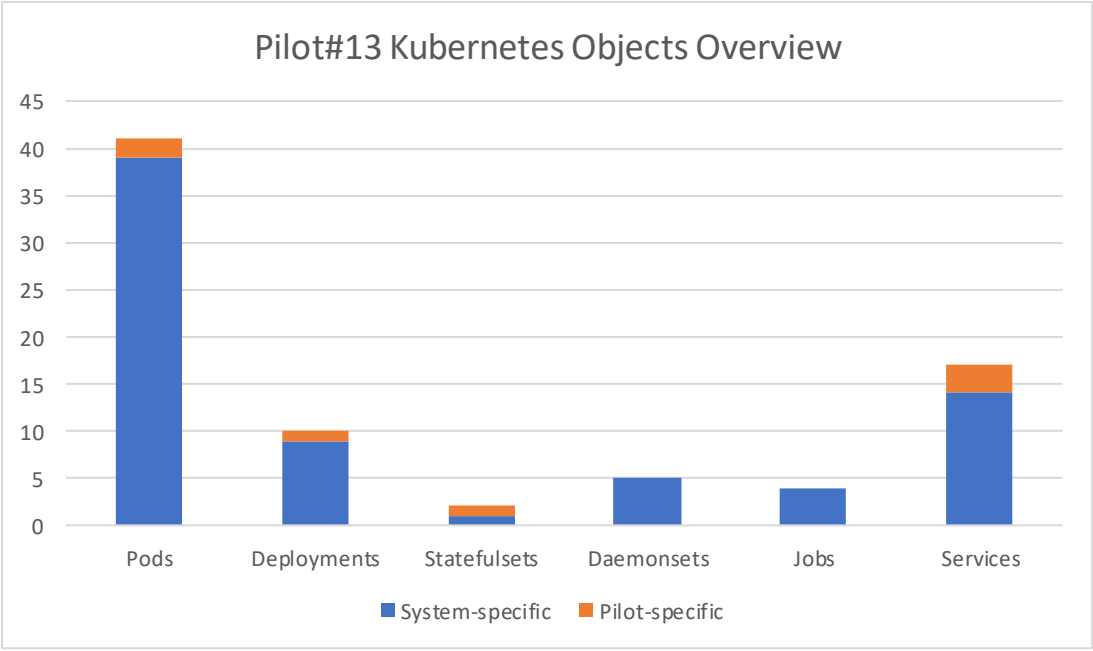

Figure 23 - Pilot#13 Kubernetes objects overview

3.4.3 Statistics and resource consumption

To have a look in detail into the running deployment unit, the pods, here is a comparison of the CPU request and CPU cap for the pods in the cluster:

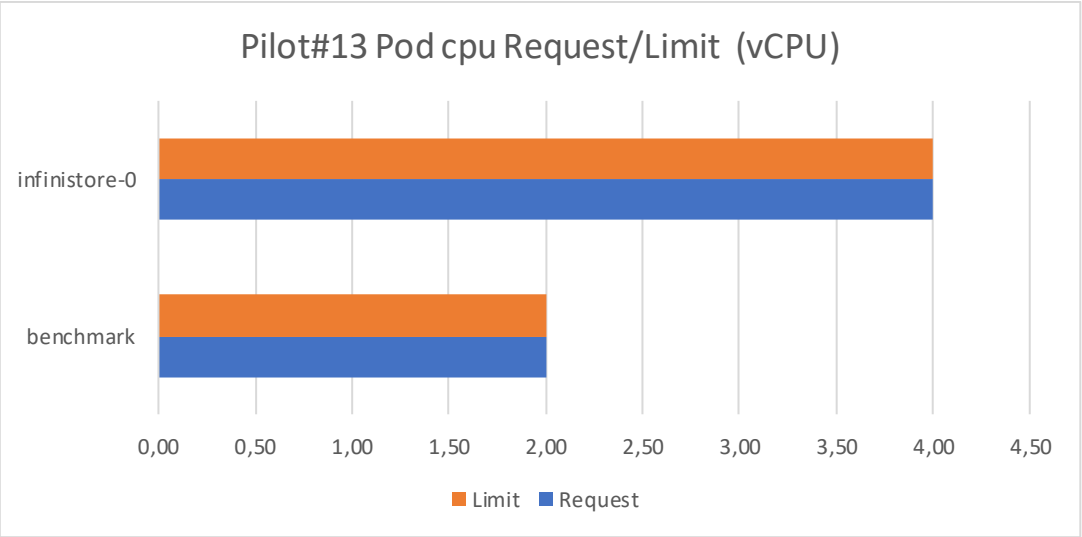

Figure 24 - Pilot#13 Pod cpu request/limit

The same comparison can be checked regarding the memory requests and limits:

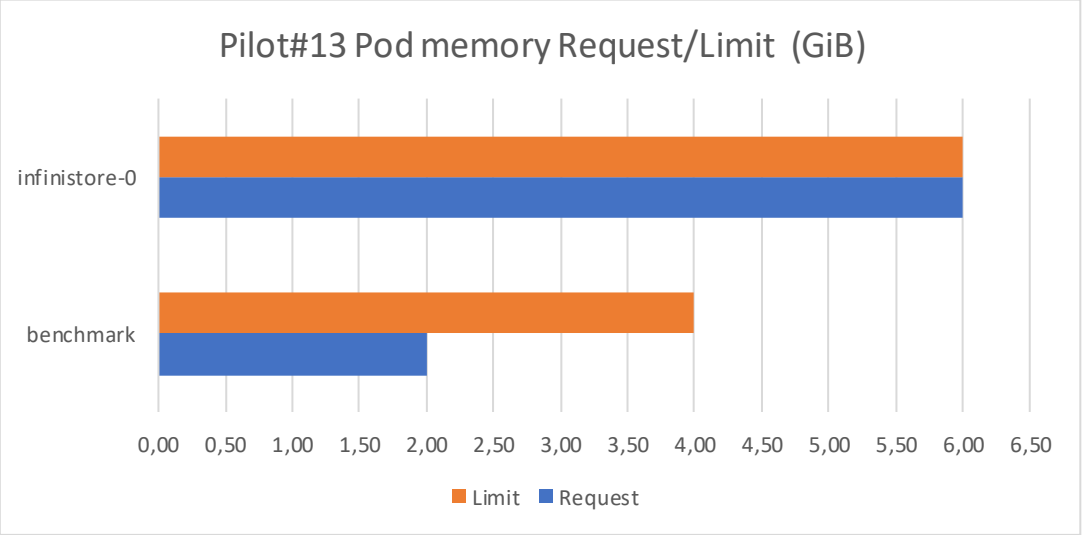

Figure 25 - Pilot#13 Pod memory request/limit

3.4.4 CI/CD timeline and trends

Pilot#13 does not use any pilot-specific component deployed into the INFINITECH sandbox. All components related to its integrated solution are being kept on-premise and only use the INFINISTORE that is deployed into the provided sandbox. As a result, there is no pilot-specific CI/CD pipeline for Pilot#13. The Data Management layer of INFINITECH includes the INFINISTORE and manages the respective pipelines for building the datastore. Since this artifact is already built, its deployment typically requires a couple of seconds. It is important to highlight that the benchmark artifact that has been shown in the previous figures of this pilot, is only used internally for benchmarking purposes, but it is not part of the integrated solution.

3.5 Pilot#14 - Big Data and IoT for the Agricultural Insurance Industry: scraping metrics

3.5.1 Cluster topology

Pilot#14 uses a Kubernetes cluster made of one control plane node and 10 worker nodes interconnected. They have the following characteristics:

Table 5 - Pilot#14 cluster nodes

Name	Role	CPU (vCPU)	Ram (GiB)
pilot14-master1	Control Plane	24	17
pilot14-worker1	Worker	24	17
pilot14-worker2	Worker	24	17
pilot14-worker3	Worker	24	17
pilot14-worker4	Worker	24	17
pilot14-worker5	Worker	24	17
pilot14-worker6	Worker	24	17
pilot14-worker7	Worker	24	17
pilot14-worker9	Worker	6	17
pilot14-worker10	Worker	6	17
pilot14-worker11	Worker	24	17

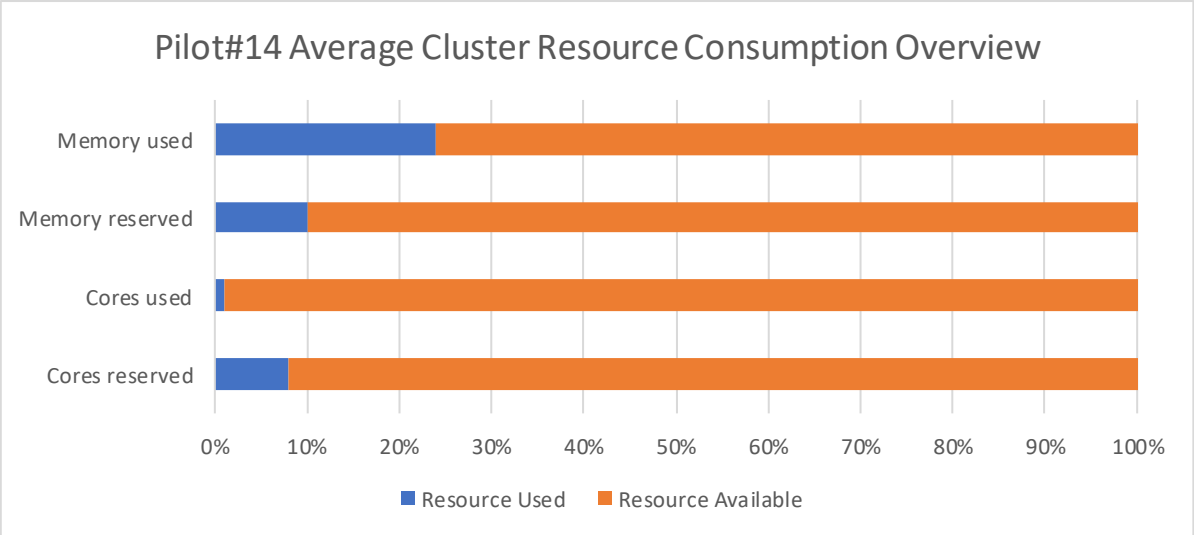

Figure 26 - Pilot#14 average cluster resource consumption overview

3.5.2 Active Kubernetes objects

The status of the Kubernetes objects declared and applied for pilot#14 is recapped in the chart below:

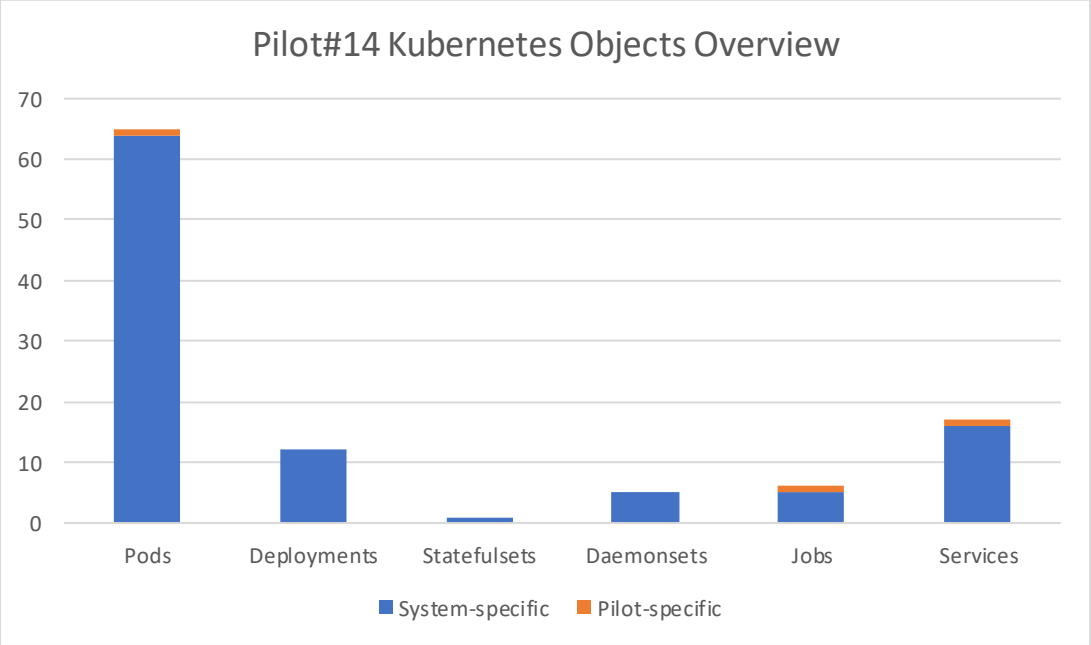

Figure 27 - Pilot#14 Kubernetes objects overview

3.5.3 Statistics and resource consumption

A detailed look into the running deployment unit, the pods, is given as a comparison of the CPU request and CPU cap for the pilot specific running pods in the cluster:

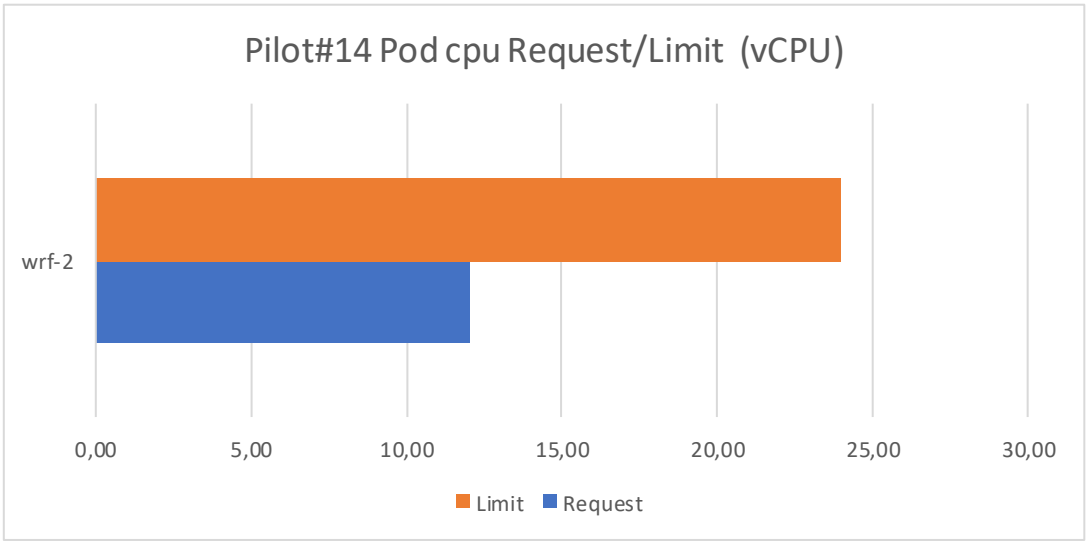

Figure 28 - Pilot#14 Pod cpu request/limit

The same comparison can be checked regarding the memory requests and limits:

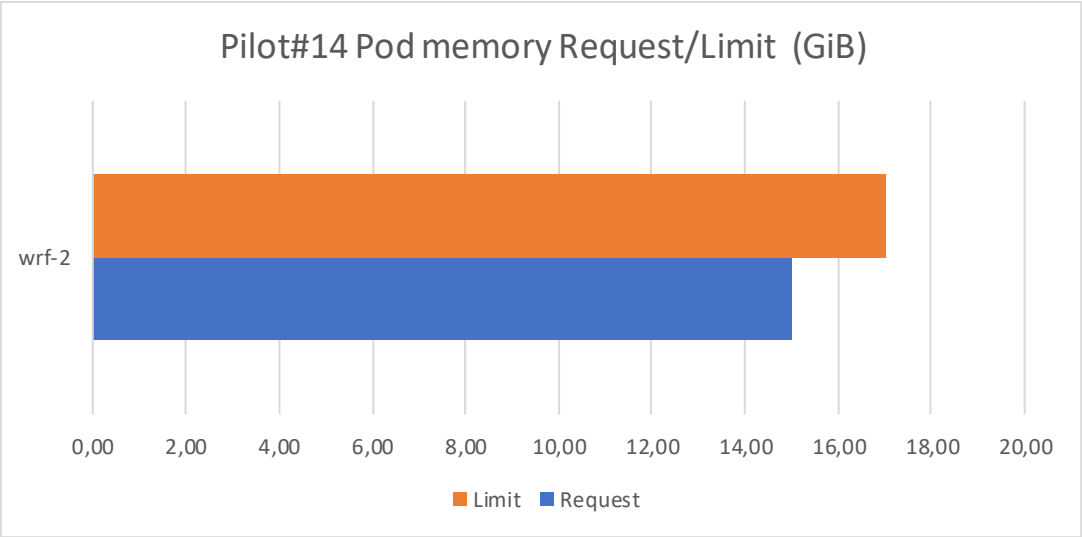

Figure 29 - Pilot#14 Pod memory request/limit

3.5.4 CI/CD timeline and trends

Pilot#14 has three active pipelines in INFINITECH that are not managed by Jenkins. Pilot14 has an internally managed stage split in three steps: Pull Image from private docker hub (Pipeline 1), then create volumes in Kubernetes to hold the data (input and output of the model Pipeline 2) and finally launch the containerized weather forecasting model (Pipeline 3).

3.6 Blockchain sandbox: scraping metrics

3.6.1 Cluster topology

The blockchain-enabled sandbox is the latest sandbox hosted on Nova, it is implemented with a Kubernetes cluster v1.20.11 with one control plane node and 3 workers. Their specifications are reported in the table below:

Table 6 - Blockchain cluster nodes

Name	Role	CPU (vCPU)	Ram (GiB)
blockchain-master1	Control Plane	4	7.64
blockchain-worker1	Worker	8	17
blockchain-worker2	Worker	8	17
blockchain-worker3	Worker	8	17

A snapshot of the cluster context reports this usage:

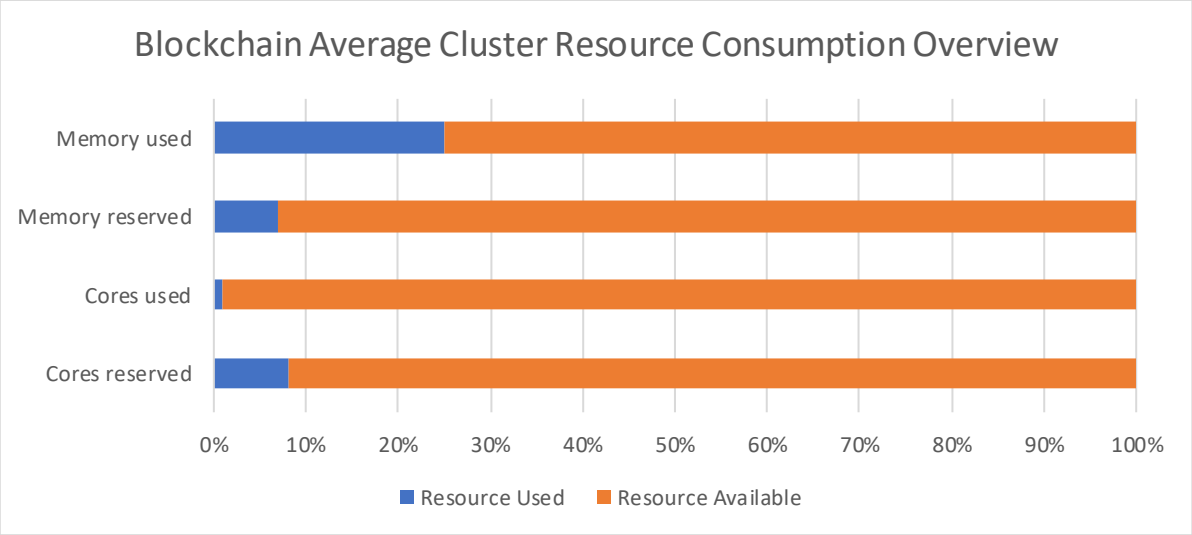

Figure 30 - Blockchain average cluster resource consumption overview

In this case, we refer to the "idle condition" a condition in which the system is not processing any transactions on the blockchain and the dApps are not receiving any requests. As it is possible to deduce from the Blockchain Average Cluster Resource Consumption Overview chart, the idle conditions are very lightweight and the CPU and memory load is mostly influenced by the blockchain core components (orderers, peers, CAs), that push the resource requests up to 40% - 60% of the capability.

3.6.2 Active Kubernetes objects

The blockchain-dedicated cluster has the following object distribution:

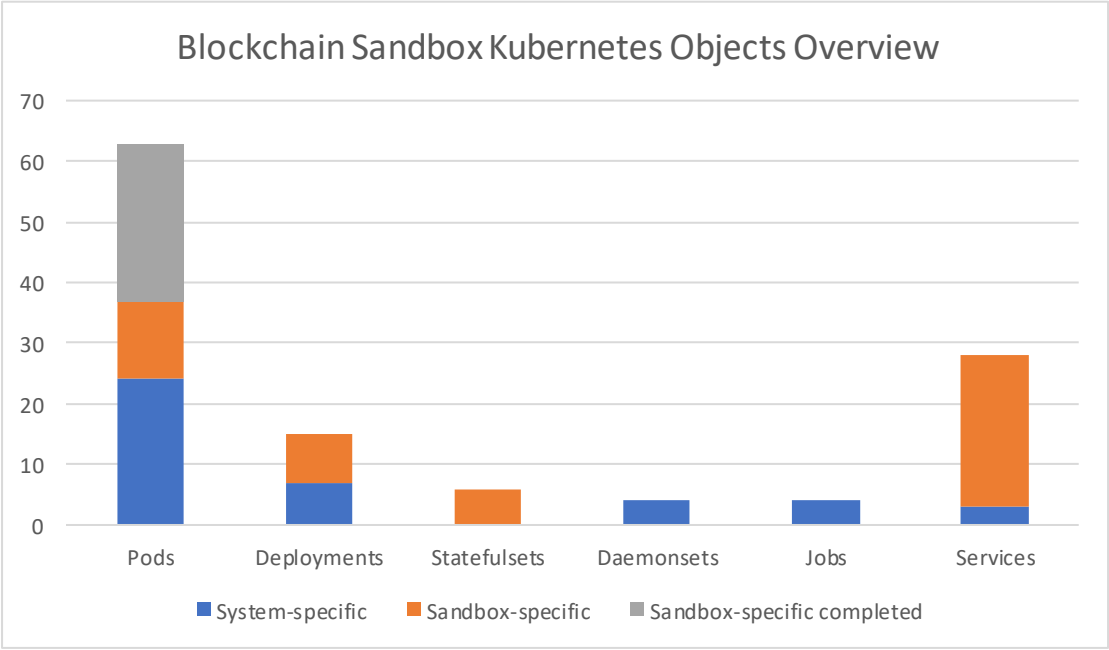

Figure 31 - Blockchain sandbox Kubernetes objects overview

3.6.3 Statistics and resource consumption

It is time to dive into the sandbox-specific pods running in the cluster with a request/limit comparison. There are 13 elements to present. Among these elements, there's the hyperledger fabric distribution that brings into the system two peers representing two different organizations, one orderer and a certification authority module for each organization.

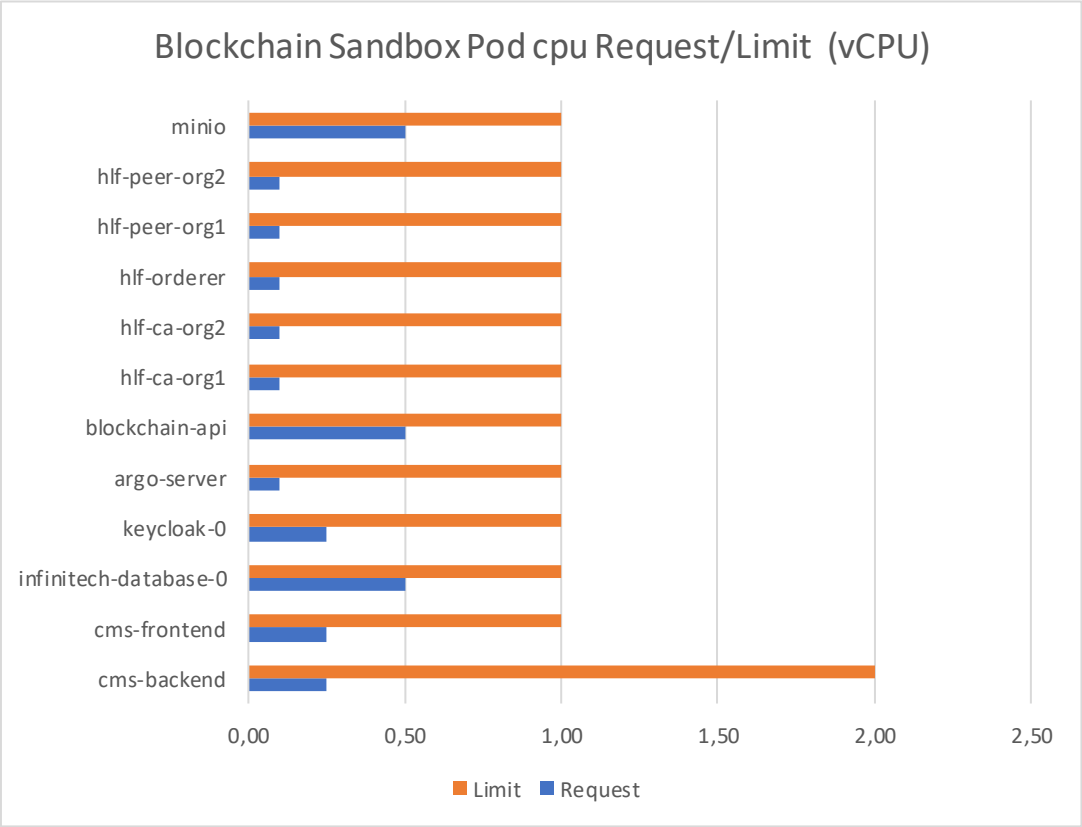

Figure 32 - Blockchain sandbox Pod cpu request/limit

It is a distribution of the resources uniform enough, also considering that many of the components (Hyperledger Fabric ones) are handled via the same helm release. Memory has the following allocation:

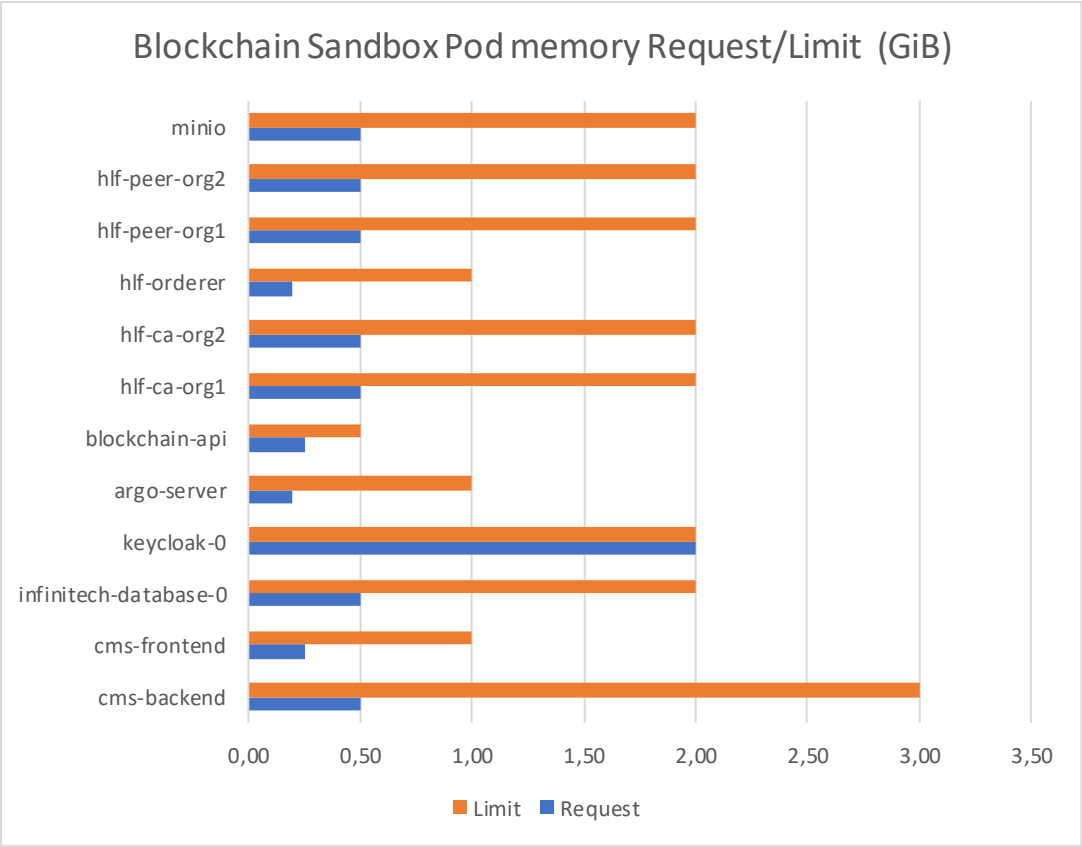

Figure 33 - Blockchain sandbox Pod memory request/limit

3.6.4 CI/CD timeline and trends

Unlike the other sandboxes which are supported by CI/CD pipelines defined in Jenkins, the blockchain sandbox makes use of Argo Workflows, a tool that comes from the Argo project suite (https://argoproj.github.io). In a few words, Argo Workflows is a workflow engine for Kubernetes clusters that comes with a set of custom resource definitions that allows defining workflows natively using DAGs.

In this specific case, Argo Workflows is used to perform maintenance operations in the Hyperledger Fabric installation running in the Blockchain Sandbox hosted by NOVA. Operations such as blockchain channel topology updates, updates and creation of the chaincode, injection of the cryptomaterial, are easily triggered using the *helm upgrade* and *argo submit* commands.

In the lifetime of the sandbox, until the creation of this report, 21 chaincode updates have been triggered (it is possible to check the statistics using the *argo list* command):

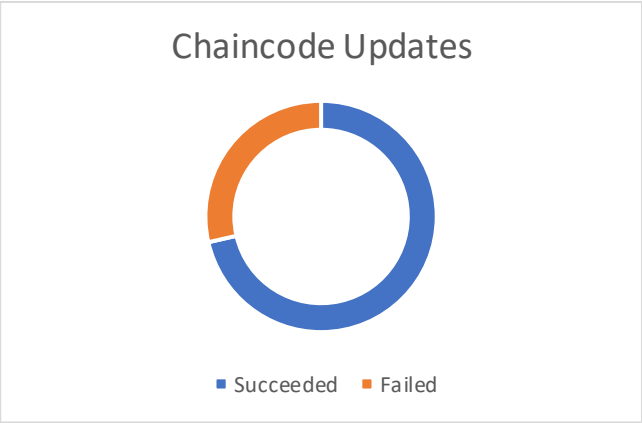

Figure 34 - Blockchain chaincode updates

4 Optimizations and environment improvements

The subdivision of a monolithic application into several microservices, on the one hand, facilitates their development and deployment, on the other hand, makes their management extremely complex due to a large number of microservices themselves.

This complexity makes the correct management of communication, load balancing, tracing and faults challenging, and often a simple Load Balancer service is not enough to manage this scenario. In order to overcome these challenges, it could be useful to use a Service Mesh.

Service Mesh is designed to meet the need to manage these complex environments based on microservices with high volumes of traffic. Some of the features it provides to facilitate the management of these environments are:

- Load balancing: It allows to have a load balancing based on Layer 7 instead of only classic Layer 4;
- Encryption: It allows to encrypt all the requests and responses both towards services and among services, the most used technology is mTLS (mutual TLS);
- Circuit breaker pattern: It allows managing automatically faults in microservice instances;
- Observability: It provides through metrics, logs, and distributed traces a better understanding of service behavior.

4.1.1 Technology architecture

The Service Mesh for its operation takes advantage of the logical separation that exists between the "Data plane", namely where the traffic takes place between services and the "Control plane" where all logic and policies are saved. In practice, the Service Mesh creates a proxy for each instance or microservices, these proxies within Kubernetes are called "sidecars". The sidecar duty is to intercept all calls towards the microservice by applying the policies defined by the engine deployed in the Control plane of the system.

The Istio Architecture is shown in Figure 35.

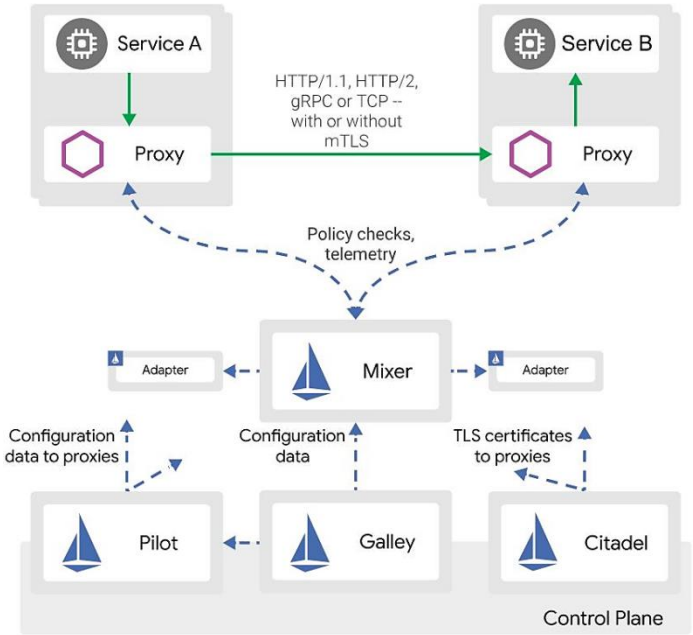

Figure 35 - Istio Architecture

In particular:

- **Proxy**: it is based on Envoy open source proxy developed in C++ ;
- **Mixer**: Enforces access control and usage policies across the service mesh, and collects telemetry data from the Envoy proxy and other services;
- **Pilot**: Provides service discovery for the Envoy sidecars, traffic management capabilities for intelligent routing (e.g. canary rollouts), and resiliency (e.g. retries, circuit breakers);
- **Citadel**: Enables strong service-to-service and end-user authentication with built-in identity and credential management;
- **Galley**: It is Istio's configuration validation, ingestion, processing and distribution component.

4.1.2 Service distribution and configuration

The Istio API Gateway relies on Kubernetes for scaling, high availability and persistence.

All configuration is stored directly in Kubernetes; there is no database. The Istio API Gateway is packaged as a single POD called "istio-ingressgateway" as well as other Istio components (Mixer, Pilot,Citadel and Galley).

By default, the API Gateway is deployed as a Kubernetes deployment and can be scaled and managed like any other Kubernetes deployment.

4.2 Logging

4.2.1 Service overview

In a microservices-based environment, it is crucial to have a robust and high-performance solution that can track, retrieve and analyze all the logs in a simplified way. The INFINITECH components are compliant with the twelve factor[3] advice, so the EFK (**E**lasticsearch**F**luentd**K**ibana) stack could be a good choice as a centralized logging solution.

4.2.2 Technology architecture

Elasticsearch is a real-time, distributed, and scalable search engine that allows for full-text and structured search. It is commonly used to index and search through large volumes of log data.

Elasticsearch is commonly deployed alongside Kibana, a powerful data visualization frontend and dashboard for Elasticsearch. Kibana allows you to explore your Elasticsearch log data through a web interface and build dashboards.

In Kubernetes, containerized applications that log to stdout and stderr have their log streams captured and redirected to JSON files on the nodes. The Fluentd is a popular open-source Data Collector that tails, transforms, and ships these log files to the Elasticsearch backend.

The EFK stack is shown in Figure 36.

[3] https://12factor.net/

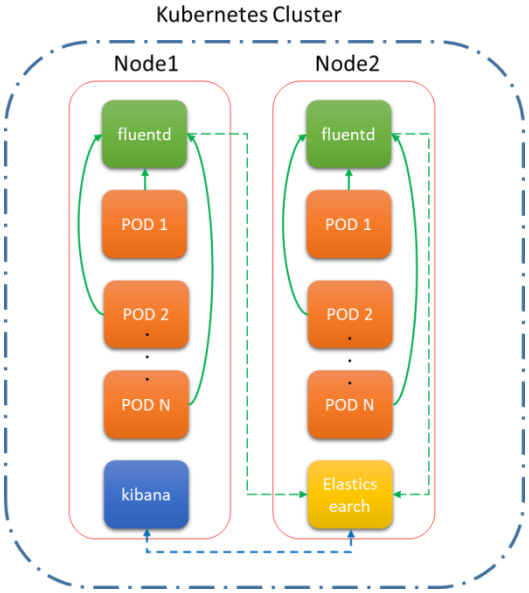

Figure 36 - EFK Stack

Regarding the format of log lines, the specification should be:

- within a single service, the service must use the same format for all log lines. This is required for parsing the text into fields to be fed into Elasticsearch;
- all log lines must contain at least: timestamp, log level and message;
- the timestamp is in a ISO8601-like format, e.g. "2019-05-15 21:49:01.814";
- services may add more fields to the log line (e.g. the name of the internal class/component) if required.

For example, a typical log line for application looks like:

```
2019-05-15 21:49:01.527 INFO 1 --- [https-jsse-nio-8443-exec-4]
o.s.web.servlet.DispatcherServlet : FrameworkServlet 'dispatcherServlet':
initialization started
```

Where the first field is the timestamp, followed by the log level, the process id, the thread name, the logger name and finally the log message.

4.2.3 Service distribution and configuration

The deploy the EFK stack rely on the statefulSet Kubernetes functionality in combination with Headless Service for each Elasticsearch node. With the statefulSet the name of the pod is not random, instead each pod gets an ordinal name. This, combined with the Headless Service, allows pods to have stable identification. In addition, pods are created one at a time instead of all at once, which can help when bootstrapping a stateful system.

For Fluentd DaemonSet Kubernetes must be used to ensure that all nodes run a copy of a Pod. As soon as nodes are added to the cluster, fluentd pods are added to them. As nodes are removed from the cluster, those Pods are deleted. Kibana instead, is deployed as simple Pod through Kubernetes deployment.

5 Conclusions

Observability is one of the key aspects for keeping distributed systems healthy and guaranteeing the availability of the services that run within our environment, in the specific case, the sandboxes hosted in the Nova infrastructure. It allows users to deduct performance characteristics from the internal components that belong to the system, detect bottlenecks, issues, and take mitigation and fix actions fasts.

From the report, several aspects related to the software runtime environment can be analyzed, as well as some anomalies in the processes that support the software development. For example, if some build step takes too long to complete, it could be a symptom of bad practice or an issue in the CI/CD automation.

The data we have collected reveal that pilot and tech proxies have been able to run their software without inconveniences or restrictions imposed by the system, which is yet another demonstration of the possibility to tailor dedicated INFINITECH Sandboxes in an on-premise environment, such as Nova's testbed, without any hassle.

INFINITECH's Nova testbed runs a constellation of software providing an observable environment for microservice-based and cloud-native architecture, adopting cloud computing best practices and appropriate runtime isolation mechanisms, making it a good reference example for easily reproducing such a complex system on other infrastructures so that they can be ready to be used, in line with the INFINITECH reference architecture and compliant with the "INFINITECH way" principles.