

D5.4 - Platform and building blocks for Federated AI

Deliverable Id:	D5.4
Deliverable name:	Platform and building blocks for Federated AI
Status:	Draft
Dissemination level:	Public
Due date of deliverable:	2022-08-31 (M24)
Actual submission date:	2022-08-26 (M24)
Work package:	WP5 "AI and Safe Autonomous Decisions"
Organization name of lead contractor for this deliverable:	IKERLAN
Authors:	Juan Manuel Besga, IKER Kevin Villalobos, IKER Mikel Irastorza, IKER Vahid Mohsseni, UOULU Huong Nguyen, UOULU Mickaël Bettinelli, UOULU Matthieu Chailloux, UOULU Harisyam Manda, AVL Alexander Flick, PLC2 Alfonso González, ZYLK Sergio Martín, ZYLK Enrico Ferrari, RULEX
Reviewers:	Nadia Caterina Zullo Lasala, ROT Amal Alrish, ROT Enrico Ferrari, RULEX
Abstract:	

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

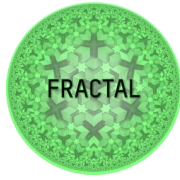
This deliverable is a part of FRACTAL WP5 Task 5.2 developing a runtime platform to deploy, test and run the AI algorithms developed in the project. The deliverable provides technical details about the implementation aspects of the different modules (installation, configuration, etc.) as well as instructions for their use and customization by the future users of the cloud platform, complementing the deliverable D5.2 named "Intermediate platform for Federated AI", in which cloud platform logical and functional descriptions were given.



This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056



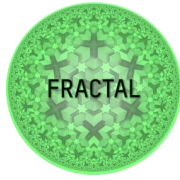
Co-funded by the Horizon 2020 Programme of the European Union under grant agreement No 877056.



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

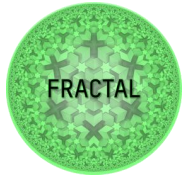
Contents

1. History	6
2. Summary	7
3. Introduction	8
4. Cloud platform overview	11
4.1 Cloud platform modules	13
4.1.1 Data ingestion	14
4.1.2 Raw data storage	14
4.1.3 Data transformation	14
4.1.4 Repositories	14
4.1.5 Machine learning workflows orchestration.....	15
4.1.6 Workflow management	15
4.1.7 Models preparation for FRACTAL Edge.....	15
4.2 Platform infrastructure.....	16
4.2.1 Public cloud services provider	16
4.2.2 Public cloud services	17
5. Cloud platform modules implementation	21
5.1 Module development and deployment guidelines	21
5.2 Data ingestion	21
5.2.1 Kafka platform with Strimzi	21
5.3 Raw data storage	26
5.4 Data preprocessing and feature extraction	32
5.5 Dataset repository and feature store	37
5.5.1 lakeFS	37
5.5.2 MinIO	39
5.5.3 Feast	42
5.6 Model repository	44
5.6.1 DVC.....	45
5.6.2 MLBuffet	47
5.6.3 MLflow	48



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

- 5.7 Image repository49
 - 5.7.1 Harbor50
 - 5.7.2 Deploying OVH Managed Private Registry50
 - 5.7.3 Configuring OVH Managed Private Registry51
- 5.8 ML orchestration55
 - 5.8.1 MLBuffet55
 - 5.8.2 MLflow56
 - 5.8.3 Kubeflow56
- 5.9 Workflow management59
 - 5.9.1 Airflow60
- 5.10 Model preparation for FRACTAL Edge63
- 5.11 Platform infrastructure65
 - 5.11.1 Configuring the OVH Managed Kubernetes65
 - 5.11.2 Accessing the OVH Managed Kubernetes Service66
 - 5.11.3 Deploy and access the Kubernetes Dashboard67
- 6. Cloud platform use guidelines72
 - 6.1 Data ingestion72
 - 6.1.1 Kafka platform72
 - 6.2 Raw data storage75
 - 6.3 Data preprocessing & feature extraction76
 - 6.4 Dataset repository and feature store79
 - 6.4.1 lakeFS79
 - 6.4.2 Feast82
 - 6.5 Model repository84
 - 6.5.1 DVC85
 - 6.5.2 MLBuffet86
 - 6.5.3 Kubeflow and MLflow87
 - 6.6 Harbor Private Registry89
 - 6.6.1 Using Harbor in FRACTAL Cloud Platform89
 - 6.6.2 Using the Image Repository91
 - 6.6.3 Using the Helm Chart Museum92
 - 6.7 ML orchestration94



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

6.7.1 MLBuffet94

6.7.2 Kubeflow and MLflow96

6.8 Workflow management99

6.8.1 Airflow99

6.9 Model preparation for FRACTAL Edge 101

6.9.1 Triggering a preparation run 101

6.9.2 Upload model into MLflow Repository within DAG 101

6.10 Upload configuration scripts to FRACTAL Cloud Platform 102

6.10.1 Deployment of a service in Kubernetes with YAML file 102

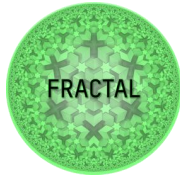
6.10.2 Deployment of a service in Kubernetes using a Helm Chart..... 103

7. Conclusions..... 105

8. List of figures 106

9. List of tables 108

10. List of abbreviations 109



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

1. History

Version	Date	Modification reason	Modified by
0.1	2022-03-28	Initial draft	Juan Manuel Besga (IKER), All authors
0.2	2022-06-14	Final version	Juan Manuel Besga (IKER), All authors
0.3	2022-06-18	Final polishing before the internal review	Mikel Irastorza, Juan Manuel Besga (IKER)
0.4	2022-07-28	Final version with modifications suggested by internal reviewers	Mikel Irastorza, Juan Manuel Besga (IKER)
1.0	2022-08-26	Final clean-up, delivered version	Mikel Irastorza, Juan Manuel Besga (IKER)

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

2. Summary

This document presents the specification of the cloud platform for FRACTAL AI, toolkits, and custom and pre-trained models for AI-based Algorithms developed in other tasks. It complements to the deliverable D5.2 named "Intermediate platform for Federated AI", in which cloud platform logical and functional descriptions were given, providing technical details about the cloud platform modules, or building blocks. In this deliverable the cloud platform modules are presented from two points of view. On the one hand, it covers the implementation aspects of the different modules (installation, configuration, etc.), while on the other hand it includes instructions for the use, customization, etc., of the modules by the future users of the cloud platform.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

3. Introduction

As mentioned in deliverable D5.1 “Specification of AI methods for use case applications”, three machine learning approaches will be considered in FRACTAL Project to meet the requirements of a variety of machine learning models for the different use cases: centralised, distributed and federated learning.

In the centralized learning approach, data is centralized into a common dataset, on the top of which different machine learning models are built and trained, actions that usually are performed in a cloud platform.

In the distributed learning approach, there are several nodes, and each node builds its own machine learning model which is trained with the data captured by the nodes (locally). In this approach, a pre-built model can be trained in the cloud platform with a common dataset, after which this model can be deployed to the different edge nodes where this model can be retrained with local data.

Finally, in the federated learning approach, the nodes learn collaboratively from a shared model while keeping their own training data locally. The shared model is first trained in a centralized way using a large-scale centralized dataset and then, the distributed nodes download the model and improve it by using their own local data. Eventually, nodes send models related data (such as performance indicators, weights, parameters, etc.) to the centralized node where are combined with the shared model, to improve the overall performance of the models. Then, this shared model is sent back to the distributed nodes where it could be fine-tuned again with local data. This way, federated learning ensures to keep the generalization capabilities of models built over a large-scale dataset, while keeping the privacy of sensitive (and critical) data and a low latency for real-time predictions or stream data processing.

In Figure 1, these three approaches are shown.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

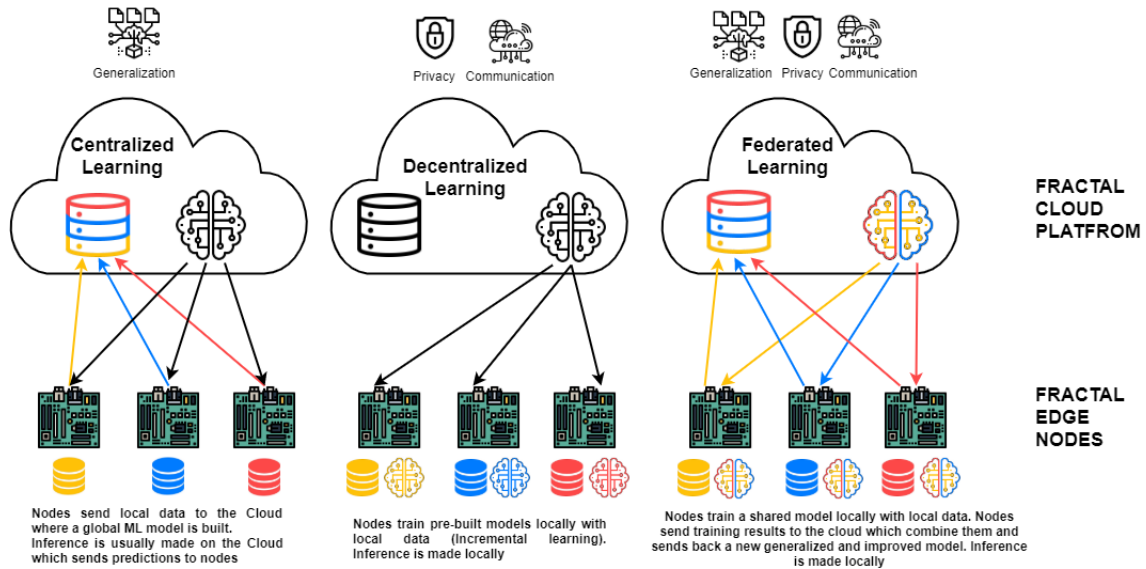


Figure 1. Learning Approaches in FRACTAL: (left) Centralized Learning; (centre) Decentralized Learning; (right) Federated Learning

The FRACTAL System, which provides the necessary infrastructure needed for this three machine learning approaches, is an Edge-oriented platform which eventually would require Cloud support for heavy resource-demanding tasks like video-processing, heavy ML model training or large data storage (historical data, for example).

This deliverable reports on the technical definition of the FRACTAL Cloud Platform and its modules or building blocks, developed in task T5.2 “FRACTAL AI Platform”, while in the deliverable D6.1 “FRACTAL processing node design and implementation”, the architecture of the Fractal Edge Node and its building blocks are described.

This deliverable complements the deliverable D5.2 named “Intermediate platform for Federated AI”, which focuses on the logical and functional descriptions of FRACTAL Cloud Platform modules, providing a detailed technical description of the implementation of these modules or building blocks, and giving guidelines for their use for providing support to the FRACTAL Edge Nodes.

This document is organized as follows. In Section 4, an overview of the FRACTAL AI cloud platform, its modules, and its relationship with other elements of the FRACTAL System, such as the FRACTAL Edge Nodes, is provided. Also, a summary of the platform modules and their functionality is presented (for more detailed information, see deliverable D5.2 “Intermediate platform for Federated AI”).

From this point on, in Section 5, a detailed technical description of the platform modules or building blocks is provided, with emphasis on their implementation.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

Section 6 presents instructions and guidelines for the use of the modules implemented in the cloud platform, in order to configure, customize and prepare them to give the required support to FRACTAL nodes.

Finally, some conclusions close the deliverable, summing up the main achievements.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

4. Cloud platform overview

In FRACTAL System, the edge nodes will be in charge of performing the critical computations in a timely and power-efficient manner, while the cloud infrastructure will be available to support the edge in any operation for which the edge is limited or not performant enough. Thus, the main identified tasks for the cloud will be the most demanding workloads like data storage, big data processing and model training.

In this manner, depending on the use case, the cloud components that will be used will vary according to the workload demanded by the use case or the edge capabilities.

Figure 2 shows the FRACTAL Edge Node architecture and its relationship with the Cloud Platform.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

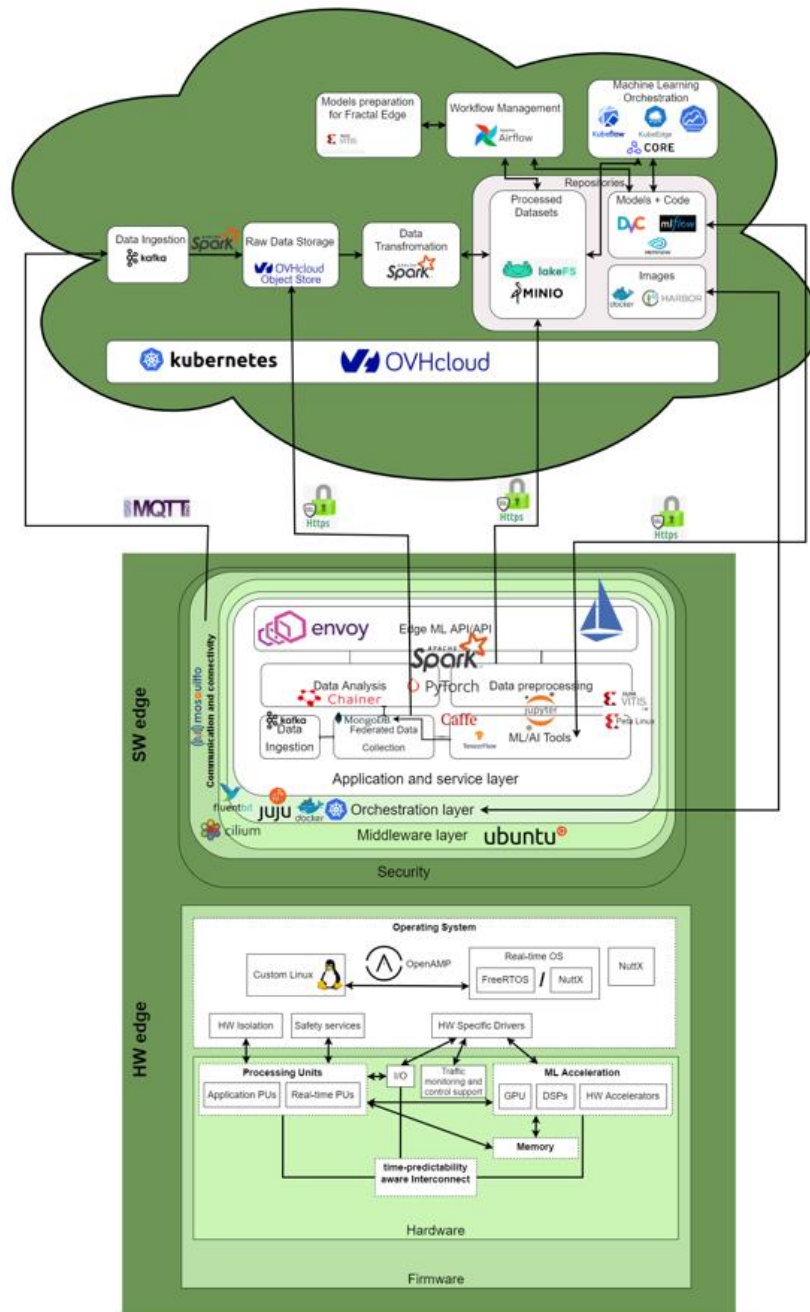


Figure 2: Relationship between the cloud platform and the edge nodes.

In the lower part of the edge node, the hardware architecture is shown, as well as the basic software elements (operating system, drivers, etc.). This architecture is a generic architecture that will have different implementations, depending on the hardware platform used in the node. Above the hardware and basic software layer (operating system, drivers, etc.), the FRACTAL Edge software layer is shown, which

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

will offer the different services necessary to perform the tasks to be carried out in it (data ingestion, data processing, inference, training/re-training of these models, etc.). The services offered by the edge node will depend on the hardware platform, so that certain services will not be available when using low-resource hardware platforms.

In cases where the node does not have enough resources to be able to perform the necessary tasks, the cloud platform described in this deliverable, which is represented in the upper part of Figure 2, could be used as a support. This cloud platform will communicate with the edge node using different protocols for the exchange of captured data, processed datasets and machine learning models.

4.1 Cloud platform modules

The FRACTAL Cloud Platform is composed of different services that are deployed over the infrastructure of a public cloud services provider. These services offer different functionalities related to the management of the data and machine learning workflows in the FRACTAL Project. In the following subsections, the main modules of the FRACTAL Cloud Platform, shown also in Figure 3, are overviewed.

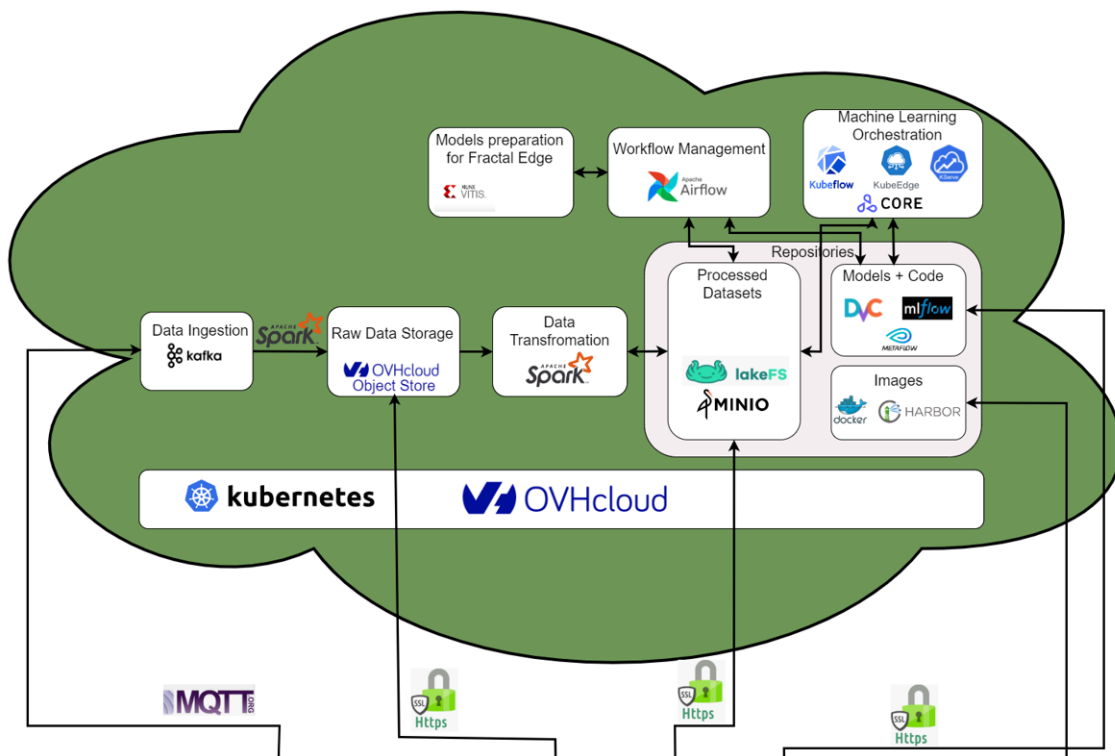


Figure 3: FRACTAL Cloud Platform modules.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

4.1.1 Data ingestion

This component provides the required functionality to ingest the data from sensors and data capture systems as soon as they are captured (i.e., in real time). This is one of the two approaches in which the cloud platform ingests data, *streaming data ingestion*. The other approach, *bulk data ingestion*, may involve the use of the Raw Data Storage module or the Processed Dataset Repository depending on whether the data uploaded are raw data or processed datasets.

4.1.2 Raw data storage

This component allows to store the raw data received in the FRACTAL Cloud Platform, through the services related to the data ingestion component, in the *streaming data ingestion approach*, or directly uploaded to the platform, in the *batch data ingestion approach*.

4.1.3 Data transformation

The data stored in the Raw Data Storage module, are retrieved by this module, which applies different data pre-processing techniques in order to transform them into datasets that are cleaned, prepared and optimized to be exploited, by using advanced data analytics techniques, such as machine learning algorithms. New transformations can also be performed on previously processed data.

4.1.4 Repositories

Different kinds of repositories are included into this component in order to allow the storage of some of the assets that will be used in the FRACTAL Cloud Platform.

4.1.4.1 Datasets repository

This module allows to store and manage the processed data sets that will be used to train different machine learning models. These data can be, either the result of a data transformation, or processed data uploaded directly to the cloud platform (*batch processed data ingestion*). The repository also provides enhanced functionalities related to datasets version control.

4.1.4.2 Models repository

This repository allows to store and manage the different ML models used in the project. The repository allows the storage not only of the code which allows to build up and deploy the models, but also of the parametrizations and configurations of models that have already been trained (frozen and pre-trained models). The repository also provides capabilities to be integrated with the different components in charge of machine learning workflow management.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

4.1.4.3 Container registry

This repository allows to store and deploy different services provided as containerized images. The images to be stored in this repository can serve for a variety of purposes such as containerized services involved in different components that compose the FRACTAL Cloud Platform, images with built-in machine learning models that are ready to be deployed on the edge, or containerized processes to process and manage data.

4.1.5 Machine learning workflows orchestration

This component is in charge of the management and orchestration of the workflows related to the machine learning models in the FRACTAL Project. Among the different tasks to be orchestrated by this component, some of the most relevant ones are the following:

1. The models training processes.
2. The evaluation and optimization (parameters fine tuning) of the trained models.
3. Serving the models in the cloud.
4. The integration of the different services of the platform with the model's storage systems.

4.1.6 Workflow management

This component is in charge of the management and orchestration of the different data workflows through the FRACTAL Cloud Platform. Among the different tasks to be orchestrated by this component, some of the most relevant ones are the following:

1. The storage of the ingested data into the platform.
2. The transformation of raw data into pre-processed datasets and their storage.
3. The integration of the different services of the platform with the data storage systems.
4. The scheduling of data-related workflows.

Additionally, this component will also be in charge of the orchestration of the different tasks that allow to prepare the models for being deployed on the edge.

4.1.7 Models preparation for FRACTAL Edge

In this module the preparation required to map ML models to the node computation resources is performed. Given the different capabilities of nodes, this preparation may involve only parameter encoding for model update after the re-training of an already mapped model. Still, it may also need to provide for the (re-)generation of the actual model executable for a given target.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

4.2 Platform infrastructure

The FRACTAL Cloud Platform is deployed over the infrastructure of a public cloud services provider, which offers different services that usually can be grouped in three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In the following subsections, details about this cloud services provider and the offered services on the top of which the FRACTAL Cloud Platform will be deployed, will be provided.

4.2.1 Public cloud services provider

There exist different providers offering cloud computing services that can be publicly consumed in an on-demand fashion. Among these providers, some of the most popular ones, that share the highest market quotas¹ are Amazon Web Services², Microsoft Azure³ and Google Cloud Platform⁴. However, there exist also other providers such as Alibaba Cloud⁵, IBM Cloud⁶ or OVH⁷.

In FRACTAL Project, OVH has been selected as cloud services provider, for developing the FRACTAL Cloud Platform. This selection has been motivated by two main reasons: on the one hand, considering the research and innovation nature of this project that has been funded by the European Commission, it seems reasonable to select an European cloud services provider. On the other hand, OVH is built upon OpenStack⁸, an open-source cloud computing platform, which favors the possibilities of migrating the platform and getting rid of vendor-locking.

4.2.1.1 OVH Public Cloud

With 30 datacenters distributed around the world and more than 1.6 million clients since 1999, OVH is the one of the largest cloud services providers in the world and largest one in Europe. As part of its public cloud solution, OVH offers its customers different resources available through the Internet. These resources are consumed in an on-demand fashion (resources are allocated and released as required) and in a 'pay as you go' business model, on which customers only pay for the resources they use. The services are offered with a high abstraction level from the subjacent infrastructure (i.e., when customers run a service, they are not aware of the infrastructure resources on which it runs). In the following subsection, the different services offered by OVH from each category will be reviewed.

¹ <https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/>

² <https://aws.amazon.com/>

³ <https://azure.microsoft.com/>

⁴ <https://cloud.google.com/>

⁵ <https://eu.alibabacloud.com/>

⁶ <https://www.ibm.com/>

⁷ <https://www.ovhcloud.com/>

⁸ <https://www.ovhcloud.com/es-es/public-cloud/why-ovh-public-cloud/> and <https://www.openstack.org/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

4.2.2 Public cloud services

The services offered by OVH can be grouped in three main categories: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) and *Software as a Service* (SaaS). The FRACTAL Cloud Platform will be leveraged by different services from these categories. In the following sections, the most relevant services and resources from each category will be presented. However, it is worth mentioning that these are not the unique resources involved in this platform. There are other services and resources (e.g., firewalls, public IPs, DNSs, private networks, etc.) that, although they will also be used, will not be presented since they are more related to the subjacent infrastructure resources, rather than to the FRACTAL platform.

4.2.2.1 Infrastructure Services (IaaS)

IaaS refers to on-demand provisioning of infrastructural resources, such as storage space, computing power, networks, and other fundamental computing resources. Among the different IaaS resources offered by OVH, some of the most relevant ones in the FRACTAL Cloud Platform are the following:

4.2.2.1.1 Compute Instances

Different *compute instances* will be used in different pools of nodes that conform a cluster on which the different services and applications of the FRACTAL Cloud Platform will run.

4.2.2.1.2 Block Storage

Block Storage resources allow to create storage volumes that can be associated with the compute instances or specific services of the platform.

4.2.2.1.3 Object Storage

The OVH *Object Storage* service manages data as objects and allows to expand the storage capabilities without having to add more hardware. This service is API compliant⁹ with AWS Simple Storage Service (S3) API¹⁰ which is an interesting property to be integrated with other services.

4.2.2.1.4 Load Balancer

OVH *Load Balancer*¹¹ service distributes the load between its different services, guaranteeing the scaling of the infrastructure in the face of increased traffic and ensuring fault tolerance and optimization of response times. In FRACTAL a *Load Balancer* will be used to provide public access to the different services running on the platform.

⁹ <https://blog.ovhcloud.com/ovhcloud-object-storage-clusters-support-s3-api/>

¹⁰ <https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html>

¹¹ <https://www.ovh.es/soluciones/load-balancer/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

4.2.2.2 Platform Services (PaaS)

PaaS refers to providing platform layer resources, including operating system support and software development frameworks. In the following subsections, the most relevant PaaS resources offered by OVH, that will be used in the FRACTAL Cloud Platform, are presented.

4.2.2.2.1 OVH Managed Kubernetes

OVH Managed Kubernetes allows to start a Kubernetes¹² cluster to orchestrate different containerized applications in the OVH cloud in a straightforward manner. Kubernetes is an open-source container-orchestration system for automating computer application deployment, scaling, and management. FRACTAL will use a Kubernetes cluster to deploy, manage and execute the different services developed alongside the different partners of the FRACTAL Project.

These services will be stored in the form of containerized docker images. These images, along with the Helm charts specifying their deployment and configuration will be stored and managed by the Harbor container registry. During the deployment, Kubernetes will access the registry to get the docker images and Helm charts to configure and deploy the containerized services in different pods of the Kubernetes cluster (see Figure 4 and Figure 58).

¹² <https://kubernetes.io/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

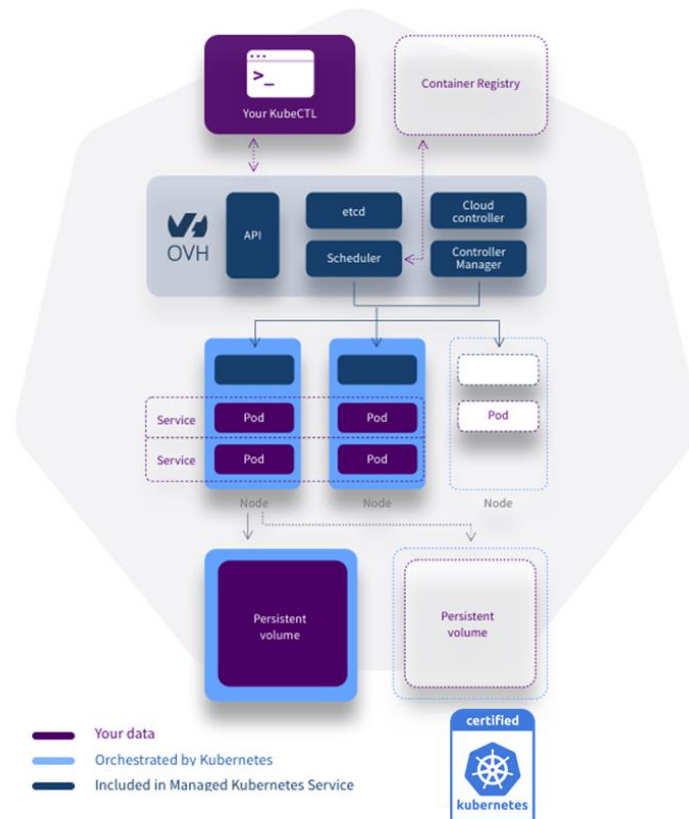


Figure 4: OVH Managed Kubernetes Service.

4.2.2.3 Software Services (SaaS)

Software as a Service (SaaS): SaaS refers to providing on demand applications over the Internet. SaaS providers offer fully developed, purpose-specific solutions to end users. The most relevant SaaS resources offered by OVH, that will be used in the FRACTAL Cloud Platform, are the following:

4.2.2.3.1 OVH Managed Container Registry

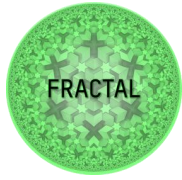
The OVH Managed Private Registry presented in Section 5.7.

4.2.2.3.2 Horizon Interface

The Horizon interface is the graphical management interface offered by OVH to manage OpenStack's resources. In FRACTAL, this interface will be used since some management functionalities are only available from this interface and not through OVH portal.

4.2.2.3.3 OVH API

The OVH API is a Web service allowing OVH customers to buy, manage, upgrade and configure OVH products without using the graphical customer



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

interface (OVH manager). In FRACTAL, this API will be used since some management functionalities are only available from it and not through OVH portal.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

5. Cloud platform modules implementation

This section presents the implementation details of cloud platform modules and the tools deployed in each one, starting with the guidelines proposed to develop and deploy the cloud platform modules.

5.1 Module development and deployment guidelines

During the development of the cloud platform, IKERLAN manages the Kubernetes service deployed over OVH on which the different services that compose the FRACTAL Cloud Platform will be deployed, so all components developed have to be deployed by IKERLAN. The steps proposed to develop and deploy components for the cloud platform are the following:

- The partners that develop different components should develop these components locally using an existing Kubernetes cluster or creating a new local Kubernetes cluster (using `microk8s`¹³ for example).
- Once the component is developed, deployed and tested locally, each partner will have to provide the corresponding configuration files (YAML or Helm charts¹⁴) and any additional instructions to deploy each service to IKERLAN. Examples of this configuration files are included on components description.
- IKERLAN will use those files to deploy the components on the FRACTAL Cloud Platform.

Any new component that needs to be developed should follow this procedure.

5.2 Data ingestion

5.2.1 Kafka platform with Strimzi

Data ingestion is the first step in any data processing. In FRACTAL Cloud Platform, data will typically be sent from IoT devices to Kafka brokers on the cloud. IoT devices will communicate with Kafka¹⁵ using MQTT¹⁶ (an open source publish/subscribe messaging protocol) and Kafka Connect (a connector which can establish connection between MQTT broker and Kafka). This requires an additional MQTT broker to be set up in Kubernetes (see Figure 5).

¹³ <https://microk8s.io/>

¹⁴ <https://helm.sh/>

¹⁵ <https://kafka.apache.org/>

¹⁶ <https://mqtt.org/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

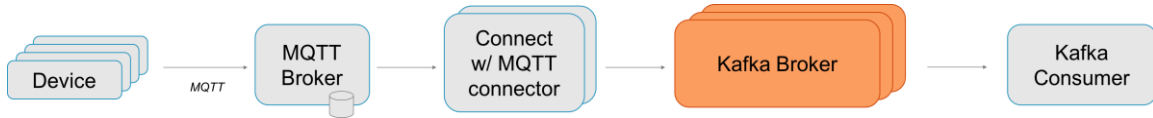


Figure 5: Data flow using MQTT broker with Connect

Another possible way to communicate from IoT devices is using MQTT broker which can stream data via Kafka Connect to the Kafka broker. MQTT broker is a go-to way for connecting devices across unreliable high latency and low network bandwidth environments in a stable way where only one way communication from IoT devices to Kafka is needed. This is a very common approach in the automotive industry, see Figure 26 with schematic of MQTT broker interfacing with Connect

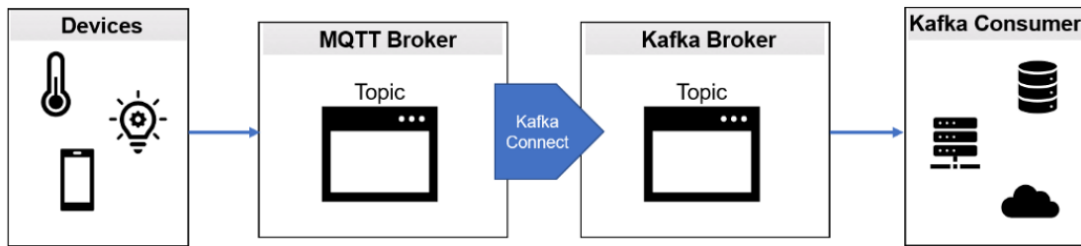


Figure 6: Schematic showing Kafka Connect bridging MQTT broker and Kafka broker

The IoT devices can either communicate with Kafka using the traditional approach of communication with the MQTT broker on kubernetes or using the Kafka-MQTT proxy as shown in Figure 7

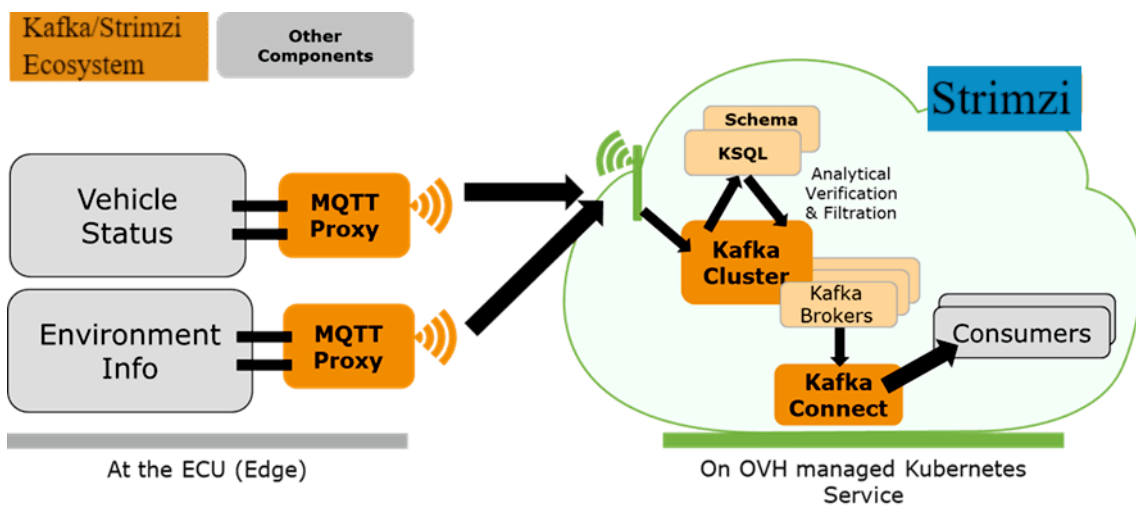


Figure 7: Communication from Edge to deployed Kafka service

Based on the above-mentioned architecture, Kafka cluster is deployed on Kubernetes using Strimzi operator, and Kafka Connect is added for connecting MQTT protocol

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

devices. Zookeeper¹⁷, an open-source Apache project that provides a centralized service for providing configuration information, naming, synchronization and group services over large clusters in distributed systems, is deployed to manage Kafka cluster.

5.2.1.1 Strimzi Operator

Strimzi¹⁸ provides a way to run an Apache Kafka cluster on Kubernetes in various deployment configurations. Then Kafka can be exposed outside Kubernetes using NodePort, Load balancer, Ingress and OpenShift Routes, depending on the needs, and these are easily secured using TLS.

The Kube-native management of Kafka is not limited to the broker. Kafka Topics can be managed by users, Kafka MirrorMaker and Kafka Connect using Custom Resources.

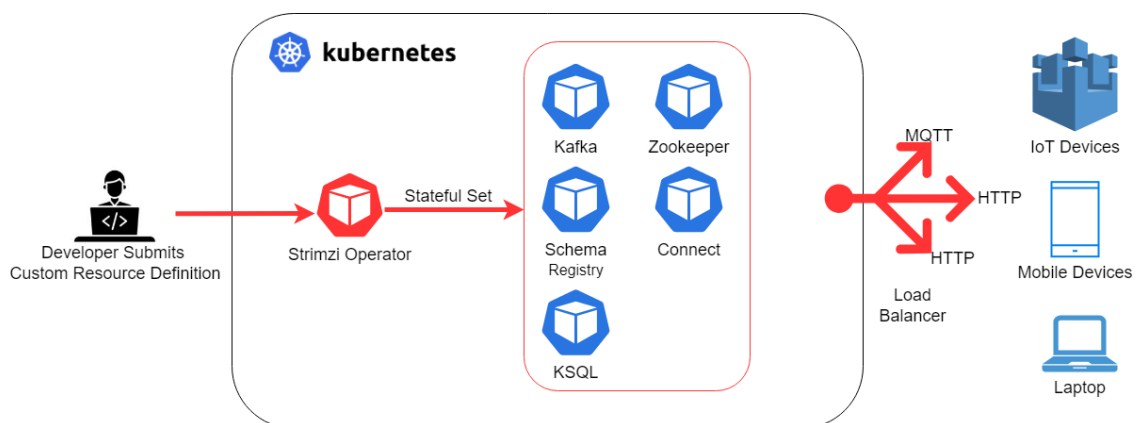


Figure 8: Strimzi Operator and Kafka Architecture in K8s

The IoT devices can either communicate with Kafka using the traditional approach of communication with the MQTT broker on Kubernetes or using the Kafka-MQTT proxy.

5.2.1.2 Pre-requisites Installation

Before installing Strimzi platform on Kubernetes, there are necessary tools that must be installed on the user's laptop. The following are the pre-requisites:

1. `kubectl` can be downloaded from Kubernetes-Tools (<https://kubernetes.io/docs/tasks/tools/>)
2. Install `helm` by using the instructions in the following link (<https://helm.sh/docs/intro/install/>)

¹⁷ <https://zookeeper.apache.org/>

¹⁸ <https://strimzi.io/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

3. Install Kafka-python library with ``pip install kafka-python`` for accessing kafka using python.

5.2.1.3 Strimzi Operator installation

Firstly, a namespace should be created for Kafka in Kubernetes and then Strimzi operator file should be installed in the same namespace:

```
kubectl create namespace kafka
```

The following steps are needed to install the components:

1. Obtain the ``kubeconfig`` file from the FRACTAL OVH cloud's Manage Kubernetes page as shown in Figure 9

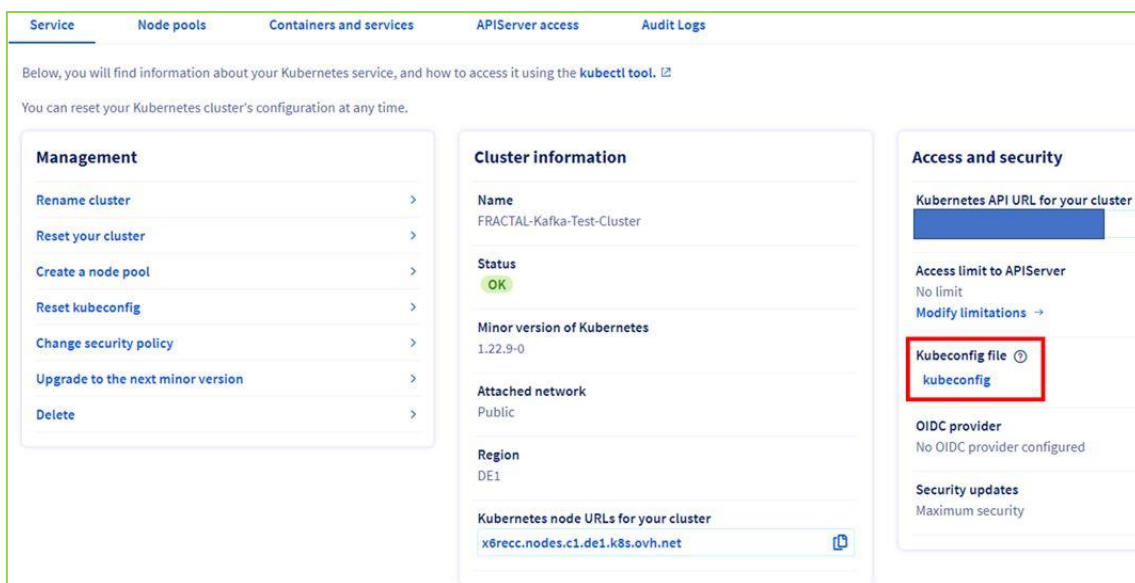


Figure 9: Obtaining the kubeconfig file

2. Use ``kubeconfig`` to set the config for kubectl present in the ``$HOME/.kube/config``
3. Verify the access to Kubernetes control plane using the command ``kubectl cluster-info`` in the terminal window
4. Then create the namespace ``kafka-stimzi`` to use it for deploying all the components and set it to default for the Kubernetes context using the commands

```
kubectl --kubeconfig your-kubeconfig create namespace kafka-stimzi
kubectl --kubeconfig your-kubeconfig config set-context --current --namespace kafka-stimzi
```

5. Add Strimzi for Kubernetes operator

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
kubectl create -f https://strimzi.io/install/latest?namespace=kafka -n kafka
```

6. Install Strimzi Kafka Platform on the Kubernetes cloud using the YAML file from the repo <https://github.com/harisymmuv/kafka-stream-ovh-fractal>

```
kubectl apply -f kafka.yaml -n kafka
```

With the completion of the above steps the confluent platform with all the components needed will be deployed. The status of the components can be checked using the following command

```
kubectl get pods -n <namespace>
```

```

d # ~/pr/f/configfiles > kubectl get svc -n kafka-strimzi
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)
fractal-kafka-cluster-kafka-0      LoadBalancer        10.3.13.230     152.228.251.97   9094:30700/TCP
fractal-kafka-cluster-kafka-bootstrap ClusterIP            10.3.69.180     <none>           9091/TCP,9092/TCP,9093/TCP
fractal-kafka-cluster-kafka-brokers ClusterIP            10.3.69.180     <none>           9090/TCP,9091/TCP,9092/TCP,9093/TCP
fractal-kafka-cluster-kafka-external-bootstrap LoadBalancer        10.3.233.111   152.228.251.147  9094:31645/TCP
fractal-kafka-cluster-zookeeper-client ClusterIP            10.3.106.65    <none>           2181/TCP
fractal-kafka-cluster-zookeeper-nodes ClusterIP            10.3.106.65    <none>           2181/TCP,2888/TCP,3888/TCP
fractal-ksql-server                 LoadBalancer        10.3.168.157   152.228.251.157  8088:31198/TCP
fractal-schema-registry-service     LoadBalancer        10.3.216.206   152.228.251.146  8081:32479/TCP
kafka-schema-registry-cp-schema-registry ClusterIP            10.3.222.114   <none>           8081/TCP,5556/TCP
ksql-server-cp-ksql-server          ClusterIP            10.3.216.66    <none>           8088/TCP,5556/TCP

```

Figure 10: Pods in the K8s cluster

Once the external load balancers are created, the DNS entries for Kafka brokers and the Kafka bootstrap service will be added to the DNS table which will expose all the deployed services to the external partners.

Name of the Service	Description	URL
Kafka Bootstrap Server	For communicating with Kafka brokers	http://kafka-bs.fractal-kafka.ovh/
Schema Registry	For sending validation schemas	http://schemaregistry.fractal-kafka.ovh/
KSQL DB	For SQL Like abstraction on streaming data	http://ksqldb.fractal-kafka.ovh/
Kafka Connect	For enabling connection from data sinks and sources to Kafka	http://connect.fractal-kafka.ovh/
Kafka REST Proxy	To access Kafka using REST methods	http://restproxy.fractal-kafka.ovh/

Table 1: Kafka platform access URLs

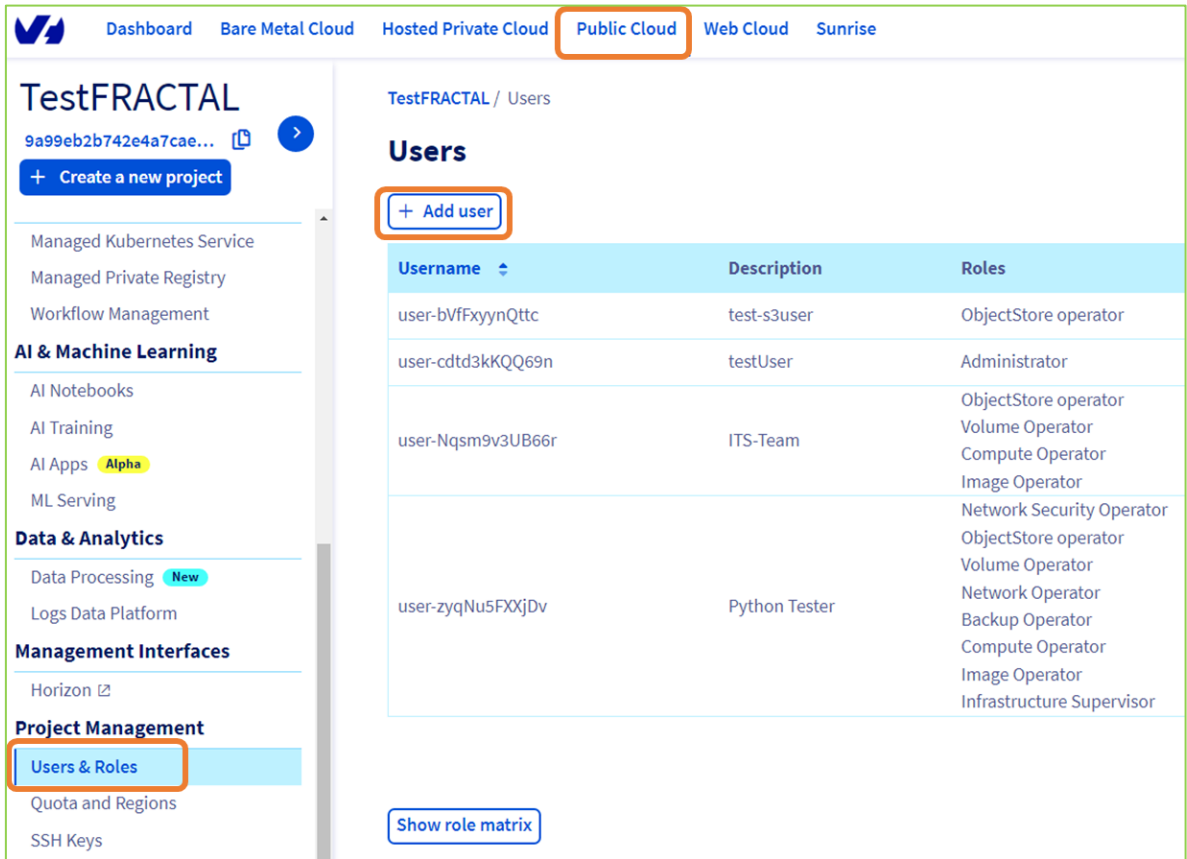
	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

5.3 Raw data storage

OVH Cloud Object Storage (OVH S3) service is the first persistent place where raw data will be stored in the FRACTAL Cloud Platform. In object storage, data is stored as standalone devices called “objects”. Each of these objects consists of the data, a unique identifier and the associated metadata. A typical use case for object storage is for catalogues of documents handled by applications, which provide static content including images, text files, tables, audio or video. OVH Object storage allows massive parallel read and write throughput of unstructured data objects. OVH offers RESTful S3 API to interact with its Object Storage, where the S3 API allows for accessing the objects programmatically and makes it easy to automate persistence or deletion of data objects from various applications. The OVH Object Storage service API is compliant with Amazon Web Services (AWS) Simple storage S3 API which enables the developers to use the CLI tools from Amazon namely `awscli` and using python libraries like `boto3` in data processing scripts.

For accessing OVH Object storage a user account has to be setup with appropriate permissions. This can be done using the OVH Cloud’s Public Cloud panel, add user and assign ‘ObjectStore operator’ role.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		



TestFRACTAL / Users

Users

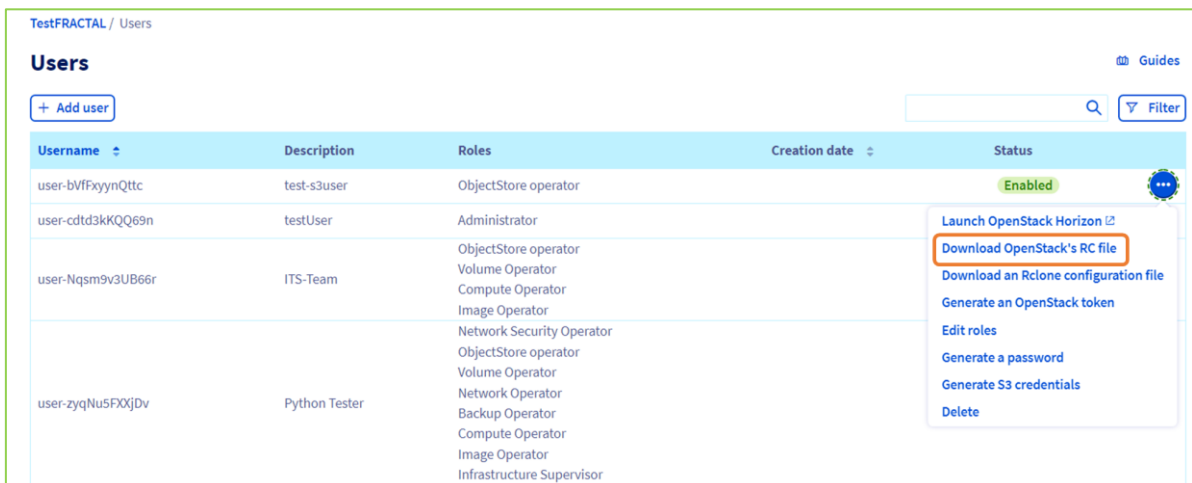
[+ Add user](#)

Username	Description	Roles
user-bVfFxyynQttc	test-s3user	ObjectStore operator
user-cdtD3kKQQ69n	testUser	Administrator
user-Nqsm9v3UB66r	ITS-Team	ObjectStore operator Volume Operator Compute Operator Image Operator
user-zyqNu5FXXjDv	Python Tester	Network Security Operator ObjectStore operator Volume Operator Network Operator Backup Operator Compute Operator Image Operator Infrastructure Supervisor

[Show role matrix](#)

Figure 11: User Access and Roles Dashboard

Download the Openstack RC file which is needed to setup the credentials for the user in the device from where the data is to be transferred



TestFRACTAL / Users

Users

[+ Add user](#)

Username	Description	Roles	Creation date	Status
user-bVfFxyynQttc	test-s3user	ObjectStore operator		Enabled
user-cdtD3kKQQ69n	testUser	Administrator		
user-Nqsm9v3UB66r	ITS-Team	ObjectStore operator Volume Operator Compute Operator Image Operator		
user-zyqNu5FXXjDv	Python Tester	Network Security Operator ObjectStore operator Volume Operator Network Operator Backup Operator Compute Operator Image Operator Infrastructure Supervisor		

- Launch OpenStack Horizon
- Download OpenStack's RC file**
- Download an Rclone configuration file
- Generate an OpenStack token
- Edit roles
- Generate a password
- Generate S3 credentials
- Delete

Figure 12: OpenStack RC file for accessing Horizon

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

5.3.1.1 Setting Openstack client

To access Object storage in FRACTAL cloud using python, the openstack client's python library must be installed. Install Openstack client using

```
pip install python-openstackclient
```

The following setup the OpenStack environment variables using the file downloaded openrc file from the above step

```
user@host:~$ source <user_name>-openrc.sh
Please enter your OpenStack Password for project <project_name> as user <user_name>:
user@host:~$
```

Figure 13: Installing python openstack client and setup

Then, setup a password of the user's choice to setup the environment variables. To access OVH S3 an access key and a secret key are required which can be created using the openstack API with the following command

```
openstack ec2 credentials create
```

After these steps, an output console shows the following credentials:

```
user@host:~$ openstack ec2 credentials create
+-----+
| Field  | Value
+-----+
| access | [REDACTED]
| links  | {u'self': u'https://auth.cloud.ovh.net/v3/users/d74d05ff121b44bea9216495e7f0df61/credentials/OS
|        | EC2/5a4d8b8d88104123a862c527ede5a3d3'}
| project_id | 20e124b71be141299e111ec26b1892fa
| secret  | [REDACTED]
| trust_id | None
| user_id | d74d05ff121b44bea9216495e7f0df61
+-----+
```

Figure 14: Creating local credentials for object storage access

Use the credentials shown in the console in AWS config file. First the AWS cli and cli-plugin endpoint have to be installed using

```
pip install awscli awscli-plugin-endpoint
```

Then, setup the AWS credentials using the following and copy the access and secret key from above

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
user@host:~$ cat ~/.aws/credentials
[default]
aws_access_key_id = <access_key>
aws_secret_access_key = <secret_key>
```

Figure 15: Editing credentials in awscli credentials file

After this, setup the AWS config using the following:

```
user@host:~$ cat ~/.aws/config
[plugins]
endpoint = awscli_plugin_endpoint

[profile default]
region = <region>
s3 =
  endpoint_url = https://s3.<region>.cloud.ovh.net
  signature_version = s3v4
s3api =
  endpoint_url = https://s3.<region>.cloud.ovh.net
```

Figure 16: Editing AWS config file to access OVH object storage

These steps will setup the access to OVH Object storage from a client. The access to OVH S3 can be verified using the following command

```
aws --profile default s3 ls
```

```
~$ aws --profile default s3 ls
2009-02-03 17:45:09 test-image-data
2009-02-03 17:45:09 testfractalobj2
2009-02-03 17:45:09 testfractalobj3
```

Figure 17: listing object storage containers in OVH cloud

5.3.1.2 Creating buckets on OVH Object Storage

To create a bucket in OVH Object Storage, Access the OVH cloud console and go to the Public Cloud page, to use the "Create Object Storage Container" as shown in Figure 18

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

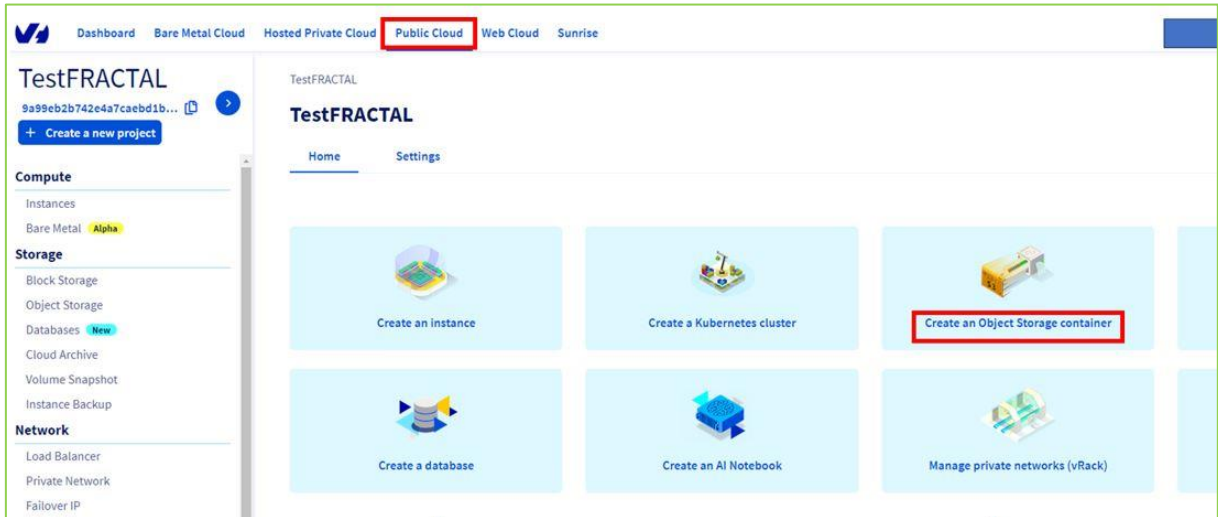


Figure 18: Object storage container dashboard in OVH cloud

Then access the Object Storage from the left pane to create an object container

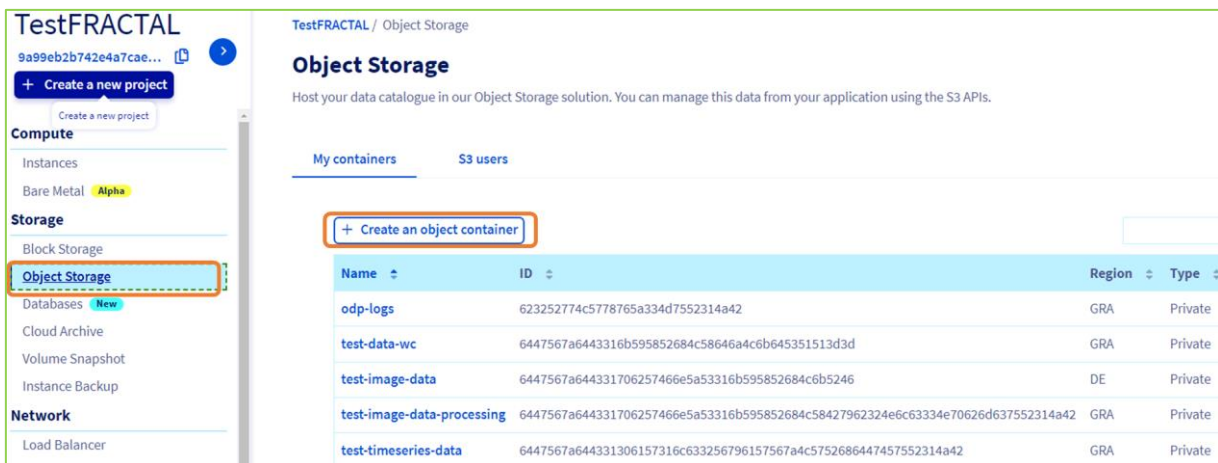


Figure 19: Creating Object container in the cloud

An object container is created by following the steps shown in Figure 20

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

TestFRACTAL / Create an object container

Create an object container

Host your data catalogue in our Object Storage solution. You can manage this data from your application using the S3 APIs. [View prices](#)

✓ Select your solution

1 **Standard (Swift)**
A space accessible in HTTPS via the OpenStack Swift API or the S3 API

✓ Select a region

2 **Frankfurt**
Frankfurt (DE)

✓ Select a type of container

3 **Public**
Multimedia, binaries, e-commerce. Store an enormous amount of data.

4 **Container name**

testMultimediaFracta|

Figure 20: Steps to create an object container

This creates the object container as shown in Figure 21:

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

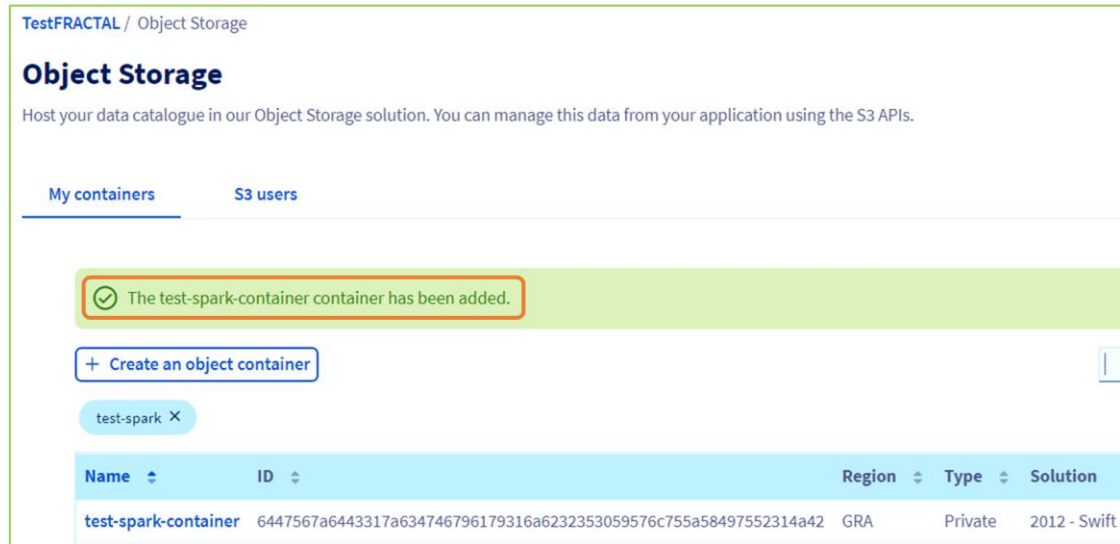


Figure 21: Final status after object container creation

5.4 Data preprocessing and feature extraction

Data preprocessing is one component of data preparation, where raw data is processed by correcting, manipulating, dropping or re-arranging the data before it is being used in the data mining process. This step is performed to ensure and enhance performance of various machine learning algorithms which consume this data for training.

Feature extraction is the process of transforming pre-processed data into numerical features that can be processed while preserving the information in the original raw data. Feature extraction can be performed either by manual definitions or through algorithms which can do automatic feature extractions.

In OVH cloud the data pre-processing and feature extraction can be done by using the "Data Processing" service. This service uses Apache Spark¹⁹, processing engine to process large amounts of data parallelly. To use Apache Spark, there is an opensource library in python called PySpark²⁰ which offers interface to interact with Spark

5.4.1.1 Managed data processing - Spark Service

To perform data pre-processing a python spark job has to be programmed and uploaded to OVH cloud Object Storage. To setup the job in the Data processing Manager, certain requirements have to be met.

¹⁹ <https://spark.apache.org/>

²⁰ <https://spark.apache.org/docs/latest/api/python/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

Requirements:

- Access to OVH cloud account with a project
- An Openstack user in the cloud project and access to Openstack Horizon Dashboard (can be created as shown here)
- Application code as python files
- An environment.yml in Conda standard

To create a simple spark job a starter script has been prepared which can be accessed in this link <https://github.com/harisymmnnv/data-transformation-spark-sample>. The script has to be uploaded in an object storage as shown in Figure 22:

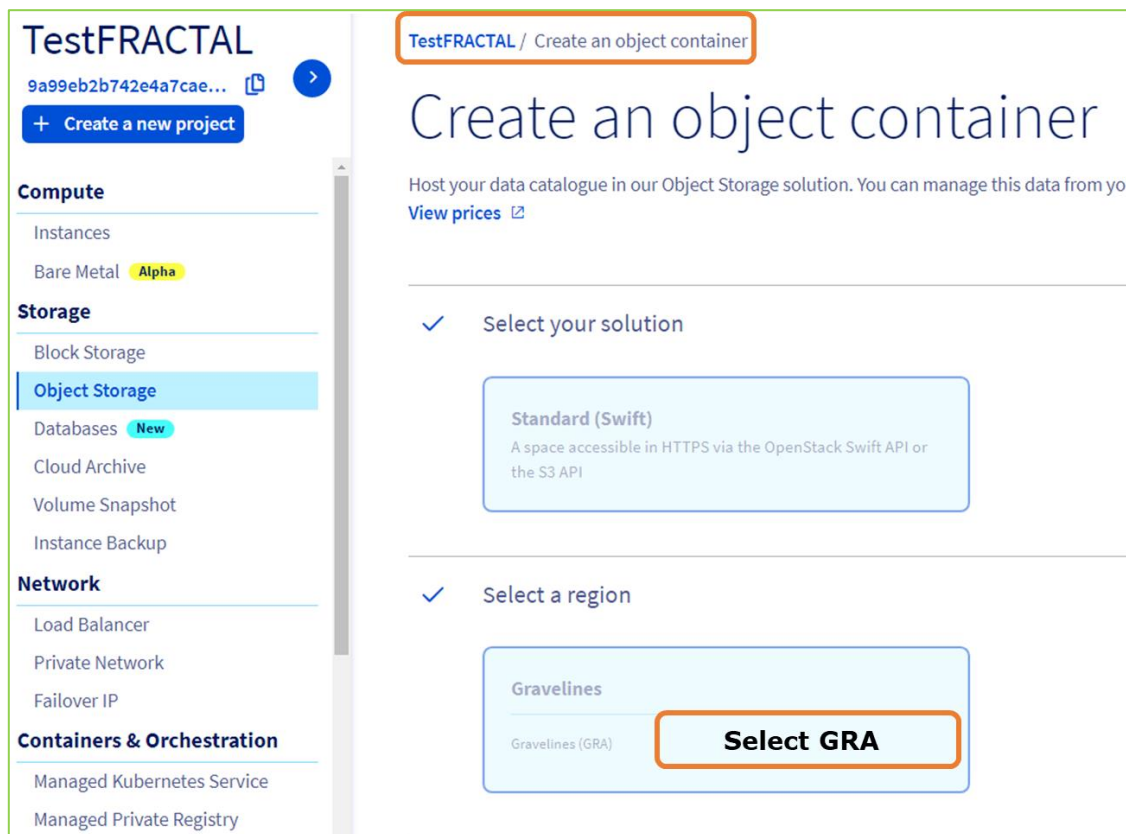
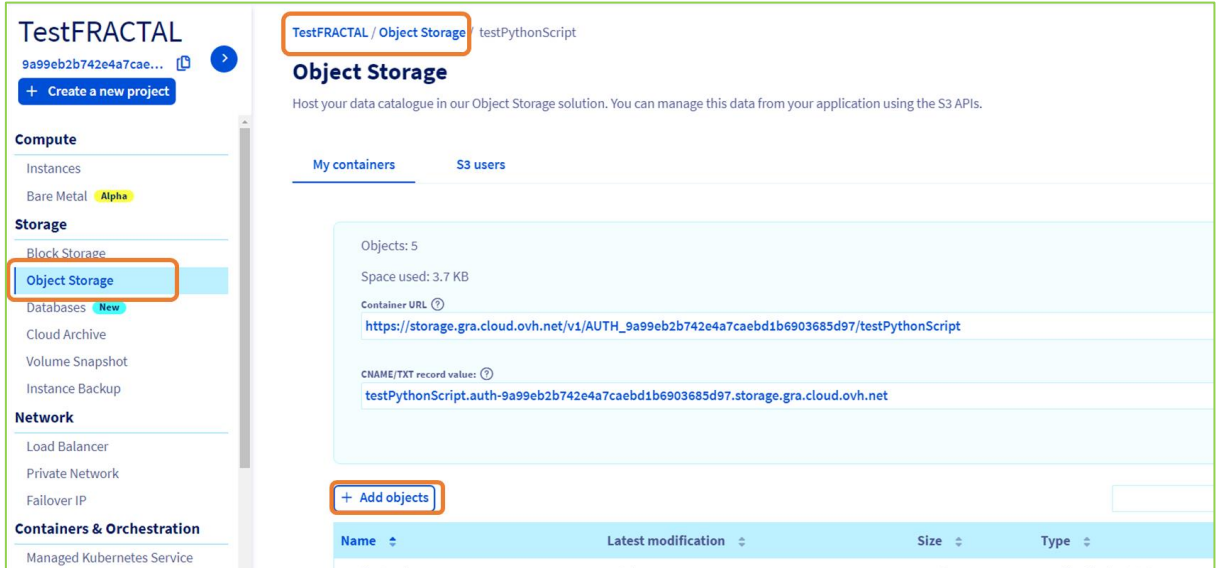


Figure 22: Object Container creation

First create a container as shown above in the figure. Once the container is created upload the python scripts and the environment YAML file to the container as shown in Figure 23:

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

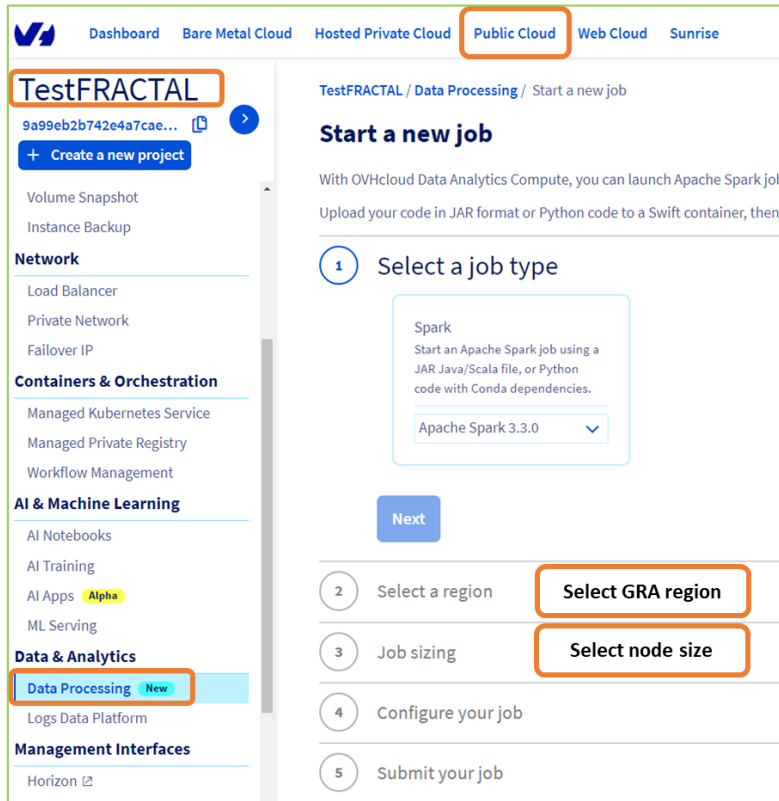


The screenshot shows the OVH cloud manager interface for a project named 'TestFRACRAL'. The left sidebar contains navigation menus for Compute, Storage, Network, and Containers & Orchestration. The 'Object Storage' option is highlighted. The main panel displays the configuration for a container named 'testPythonScript'. It shows the container URL as `https://storage.gra.cloud.ovh.net/v1/AUTH_9a99eb2b742e4a7cae1b6903685d97/testPythonScript` and the CNAME/TXT record value as `testPythonScript.auth-9a99eb2b742e4a7cae1b6903685d97.storage.gra.cloud.ovh.net`. A '+ Add objects' button is visible at the bottom of the configuration area.

Figure 23: Uploading objects to the container

To create the spark job, proceed to the Data processing tab in the OVH cloud Manager and then create the job with Spark version and region selected to the same one where the object container resides. These steps can be seen in Figure 24:

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		



The screenshot displays the OVHcloud console interface for setting up a Data processing job. The top navigation bar includes 'Public Cloud'. The main content area is titled 'Start a new job' and shows a 5-step process:

1. Select a job type: Spark (Start an Apache Spark job using a JAR Java/Scala file, or Python code with Conda dependencies). Apache Spark 3.3.0 is selected.
2. Select a region: Select GRA region
3. Job sizing: Select node size
4. Configure your job
5. Submit your job

Figure 24: Setting up a Data processing job

For configuring the job, the environment.yml file has to be present which can be created as shown at this link <https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#sharing-an-environment>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

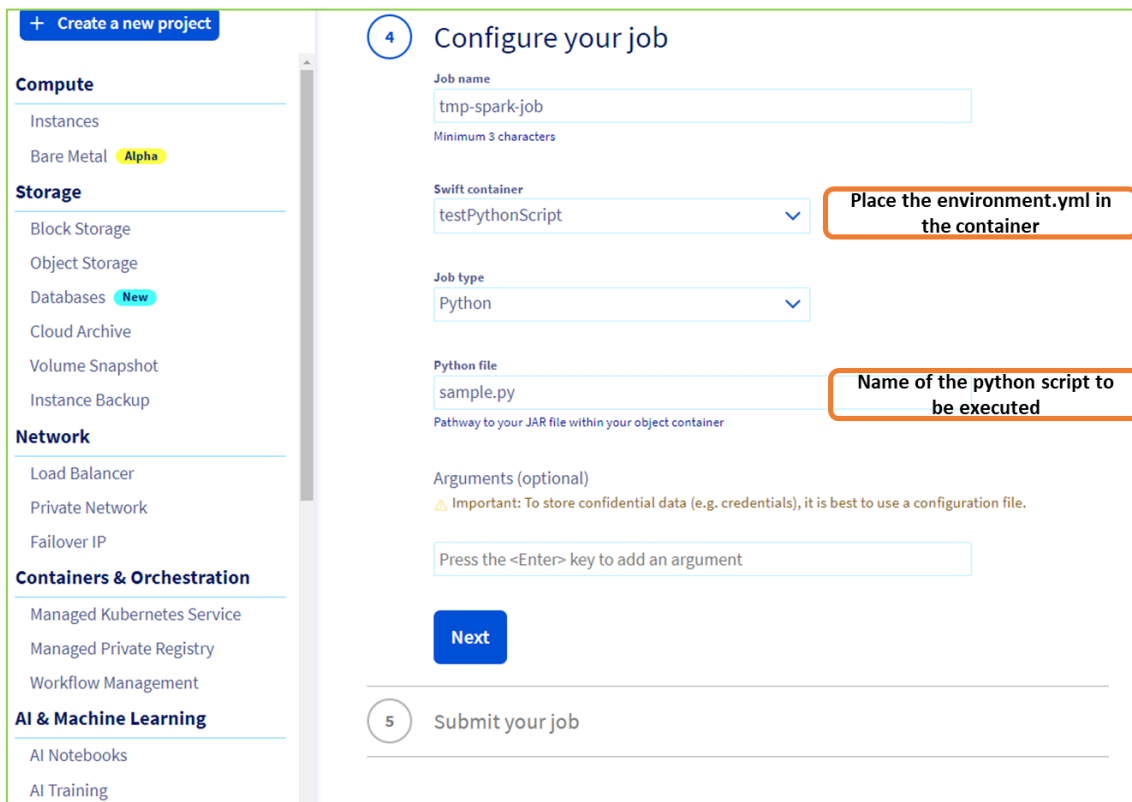


Figure 25: Selecting the container and python script in the data processing job

Then submit the job; the job logs and monitoring can be accessed from the job submission page. A Grafana dashboard can be used to monitor the progress of the job when it is running. The logs from the spark runtime are available in the logs tab

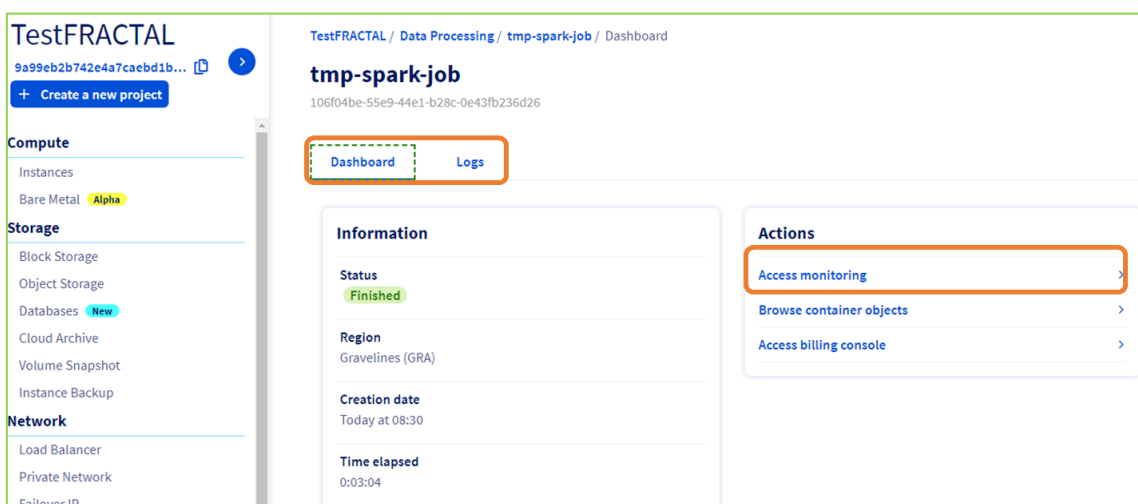


Figure 26: Job processing dashboard with logs and monitoring

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

5.5 Dataset repository and feature store

5.5.1 lakeFS

In the FRACTAL Cloud Platform multiple datasets of a variety of use cases will be stored. To provide this feature to the FRACTAL Cloud Platform, lakeFS will be deployed over the OVH Managed Kubernetes Service following the official documentation²¹ (using the official Helm chart²² stored on the Harbor Helm Charts Museum).

lakeFS²³ is an open-source platform that delivers resilience and manageability to existing object-based storage data lake. It enables building repeatable, atomic and versioned data lake operations from complex ETL jobs to data science and analytics.

The main advantage of lakeFS is that it provides a Git-like branching and committing model that scales to exabytes of data. This branching model makes data lakes ACID compliant by allowing changes to happen in isolated branches that can be created, merged, and rolled back atomically and instantly. Since lakeFS is compatible with the S3 API, all popular applications will work without modification.

As mentioned before, lakeFS is designed to be built on top of a conventional object storage service. As OVH offers an object storage service, it would be reasonable to use this service to build lakeFS on top of it. As lakeFS affirms that is compatible with any S3 compliant object storage service, an initial setup using the OVH object storage service was built. However, an error has been encountered during the installation of the lakeFS service when trying to build it on top of the OVH object storage service. After analyzing the error, an incompatibility has been found on the OVH object storage side, thus, an issue has been opened on GitHub (<https://github.com/treeverse/lakeFS/issues/2471>)

The issue has been fixed few days before writing this deliverable so although the error is currently fixed, the OVH object storage service was discarded because the issue was not fixed when the FRACTAL Cloud Platform was under development.

Due to this issue with OVH object storage, an alternative object storage solution was used. For further details on the selected object storage service, see Section 5.5.2.

5.5.1.1 Installation & Configuration

lakeFS can be installed on Kubernetes by using the official lakeFS Helm Chart. Some parameters will be customized by applying the lakefs.yml file when installing lakeFS using the chart.

²¹ https://docs.lakefs.io/quickstart/more_quickstart_options.html#on-kubernetes-with-helm

²² <https://artifacthub.io/packages/helm/lakefs/lakefs>

²³ <https://lakefs.io/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
helm repo add lakefs https://charts.lakefs.io
```

```
helm install -f lakefs.yml lakefs lakefs/lakefs --namespace fractal
```

Once lakeFS has been deployed it can be checked by using the following command:

```
kubectl get all -n fractal
```

```

NAME                                READY   STATUS    RESTARTS   AGE
pod/fractal-inferer-deployment-6969789cf4-k24dn  1/1     Running   0           5d18h
pod/lakefs-768cb7b7b7-br75g             1/1     Running   0           18h
pod/minio-5bddc45756-64jcg             1/1     Running   0           22h
pod/model-host-deployment-74b84986cb-b7mpc     1/1     Running   0           8d
pod/nginx-deployment-65cb5d769-k2zz4         1/1     Running   0           8d
pod/postgresql-postgresql-0              1/1     Running   0           19h
pod/prometheus-deployment-6d944bf45-68gqd     1/1     Running   0           8d

NAME                                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/fractal-inferer             NodePort     10.3.25.172  <none>        8000:30004/TCP  5d18h
service/lakefs                      ClusterIP    10.3.2.161   <none>        5434/TCP         18h
service/minio                       ClusterIP    10.3.68.133  <none>        9000/TCP,9001/TCP  22h
service/model-host                  NodePort     10.3.18.148  <none>        8000:30003/TCP  8d
service/nginx                       NodePort     10.3.7.172   <none>        80:30002/TCP    8d
service/postgresql                  ClusterIP    10.3.221.236 <none>        5432/TCP         19h
service/postgresql-headless         ClusterIP    None          <none>        5432/TCP         19h
service/prometheus                  NodePort     10.3.216.153 <none>        9090:30001/TCP  8d

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/fractal-inferer-deployment  1/1     1             1           5d18h
deployment.apps/lakefs                    1/1     1             1           18h
deployment.apps/minio                      1/1     1             1           22h
deployment.apps/model-host-deployment       1/1     1             1           8d
deployment.apps/nginx-deployment           1/1     1             1           8d
deployment.apps/prometheus-deployment       1/1     1             1           8d

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/fractal-inferer-deployment-6969789cf4  1         1         1       5d18h
replicaset.apps/lakefs-768cb7b7b7                    1         1         1       18h
replicaset.apps/minio-5bddc45756                     1         1         1       22h
replicaset.apps/model-host-deployment-74b84986cb     1         1         1       8d
replicaset.apps/nginx-deployment-65cb5d769           1         1         1       8d
replicaset.apps/prometheus-deployment-6d944bf45     1         1         1       8d

NAME                                READY   AGE
statefulset.apps/postgresql-postgresql  1/1     19h

```

Figure 27. lakeFS Service and Deployment

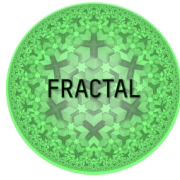
5.5.1.2 Accessing lakeFS

lakeFS can be exposed by using Ingress (or a Load Balancer) by applying the ingress-lakefs.yml YAML file:

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: lakefs-endpoints

```



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

```
namespace: fractal
annotations:
  kubernetes.io/ingress.class: "nginx"
  nginx.ingress.kubernetes.io/proxy-body-size: "0"
  nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  ingressClassName: nginx
  rules:
  - host: fractal.ik-europe.eu
    http:
      paths:
      - path: /?(.*)
        pathType: Prefix
        backend:
          service:
            name: lakefs
            port:
              number: 5434
```

```
kubectl apply -f ./LakeFs/ingress-lakefs.yml
```

This will expose the dashboard at <http://fractal.ik-europe.eu/>.

When accessing lakeFS for the first time, an admin user must be configured. After creating the admin user, the created credentials must be downloaded to access the lakeFS Portal.

5.5.2 MinIO

MinIO²⁴ offers high-performance, S3 compatible object storage. Native to Kubernetes, MinIO is available on every public cloud, every Kubernetes distribution, the private cloud and the edge. MinIO is software-defined and is 100% open source under GNU AGPL v3.

MinIO has been chosen as the object storage provider because it suits all the requirements and does not have any incompatibility issue with lakeFS. To provide a lakeFS compatible object storage to the FRACTAL Cloud Platform, MinIO will be

²⁴ <https://min.io/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

deployed over the OVH Managed Kubernetes Service following the official documentation (using the official Helm chart²⁵ stored on the Harbor Helm Charts Museum).

5.5.2.1 Installation & Configuration

MinIO can be installed on Kubernetes by using the Bitnami Object Storage Helm Chart based on MinIO. Some parameters will be customized by applying the minio.yml file when installing MinIO using the chart.

```
helm repo add bitnami https://charts.bitnami.com/bitnamihelm install -f
./Minio/minio.yml minio bitnami/minio --namespace fractal
```

Once MinIO has been deployed, the status can be checked by using the following command:

```
kubect1 get all -n fractal
```

²⁵ <https://artifacthub.io/packages/helm/bitnami/minio>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```

NAME                                READY   STATUS    RESTARTS   AGE
pod/fractal-inferer-deployment-6969789cf4-k24dn  1/1     Running   0           5d2h
pod/lakefs-768cb7b7b7-br75g                1/1     Running   0           3h14m
pod/minio-5bddc45756-64jcg                 1/1     Running   0           6h29m
pod/model-host-deployment-74b84986cb-b7mpc     1/1     Running   0           8d
pod/nginx-deployment-65cb5d769-k2zz4          1/1     Running   0           8d
pod/postgresql-postgresql-0                 1/1     Running   0           3h29m
pod/prometheus-deployment-6d944bf45-68gqd     1/1     Running   0           8d

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
service/fractal-inferer              NodePort      10.3.25.172   <none>        8000:30004/TCP   5d2h
service/lakefs                       ClusterIP     10.3.2.161    <none>        5434/TCP         3h14m
service/minio                        ClusterIP     10.3.68.133   <none>        9000/TCP,9001/TCP 6h29m
service/model-host                   NodePort      10.3.18.148   <none>        8000:30003/TCP   8d
service/nginx                        NodePort      10.3.7.172    <none>        80:30002/TCP     8d
service/postgresql                   ClusterIP     10.3.221.236  <none>        5432/TCP         3h29m
service/postgresql-headless          ClusterIP     None          <none>        5432/TCP         3h29m
service/prometheus                   NodePort      10.3.216.153  <none>        9090:30001/TCP   8d

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/fractal-inferer-deployment  1/1     1             1           5d2h
deployment.apps/lakefs                    1/1     1             1           3h14m
deployment.apps/minio                     1/1     1             1           6h29m
deployment.apps/model-host-deployment      1/1     1             1           8d
deployment.apps/nginx-deployment           1/1     1             1           8d
deployment.apps/prometheus-deployment      1/1     1             1           8d

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/fractal-inferer-deployment-6969789cf4  1         1         1       5d2h
replicaset.apps/lakefs-768cb7b7b7                    1         1         1       3h14m
replicaset.apps/minio-5bddc45756                     1         1         1       6h29m
replicaset.apps/model-host-deployment-74b84986cb     1         1         1       8d
replicaset.apps/nginx-deployment-65cb5d769           1         1         1       8d
replicaset.apps/prometheus-deployment-6d944bf45      1         1         1       8d

NAME                                READY   AGE
statefulset.apps/postgresql-postgresql  1/1     3h29m

```

Figure 28: MinIO Service and Deployment

5.5.2.2 Accessing MinIO

The Helm chart used to deploy MinIO allows to configure it to be exposed with an Ingress service. For that, in the MinIO configuration files used for the deployment (minio.yml) the hostname on which MinIO will be exposed must be set.

When accessing MinIO portal, it will ask for credentials. Access credentials can be obtained from the deployment of MinIO by using the following commands:

```

export SECRET_KEY=$(kubectl get secret --namespace default minio -o
jsonpath="{.data.secret-key}" | base64 --decode)

export ACCESS_KEY=$(kubectl get secret --namespace default minio -o
jsonpath="{.data.access-key}" | base64 --decode)

```

Those credentials will grant access to the MinIO Console.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

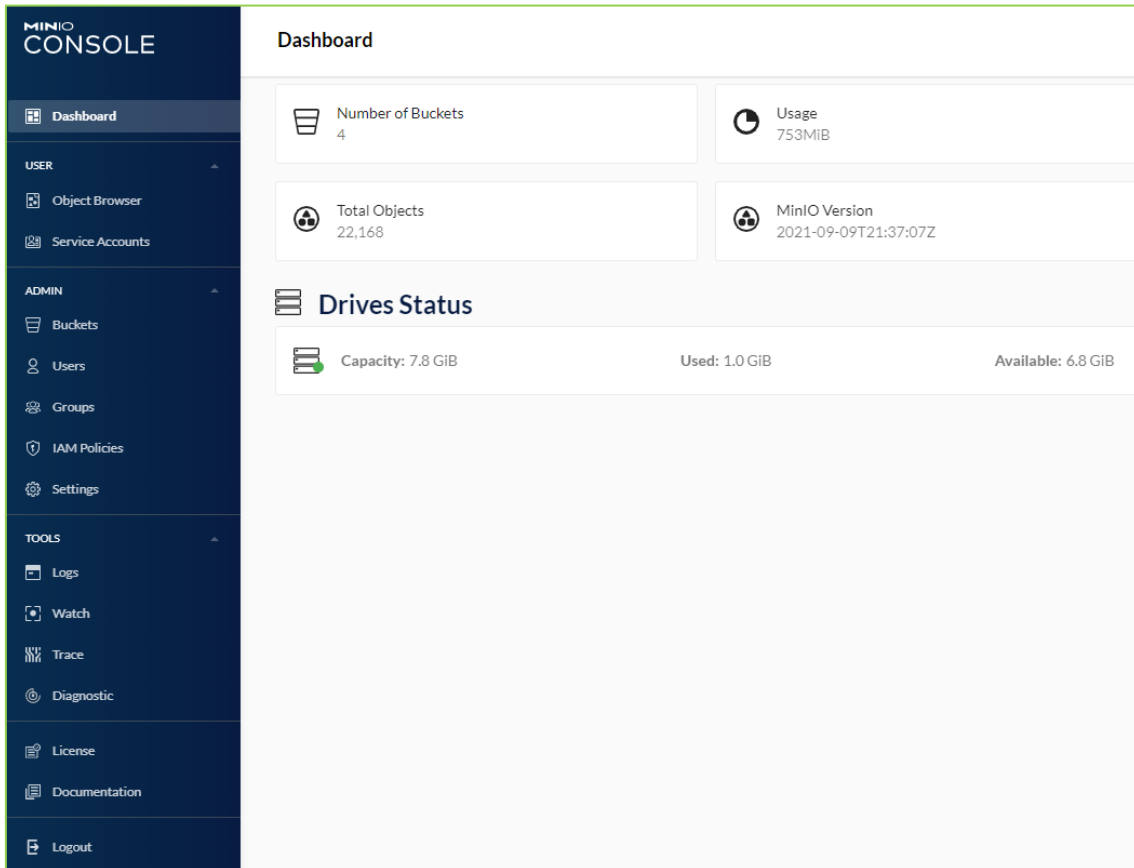


Figure 29: MinIO Console

5.5.3 Feast

Feast²⁶ (whose name is derived from Feature store) is an open-source data system for managing and storing the input features for models in production. It can serve the features to a model from both an offline (typically used in training) or an online store (typically used for real-time prediction). In the workflow, new features will be continuously materialized to update and refresh the online store after being added to the offline store.

A Feast feature store needs the following supporting infrastructure: a registry, in which metadata will be stored, an offline store to store features and an online store to serve the latest features with low latency. This infrastructure and the features stored inside it are defined in a feature repository. The versioning capabilities of lakeFS can be leveraged to keep track of any changes made to it.

The registry can be stored in an S3 bucket using MinIO.

²⁶ <https://feast.dev/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

The online store will be deployed as a Redis database. Multiple stores can use the same Redis instance.

The offline store can be a data warehouse such as BigQuery²⁷ or Redshift²⁸. Due to the deployment on OVH Cloud, the "File" offline store will have to be used. This means that when fetching features from the offline store, they will be read from the defined data sources (parquet files) and joined using Python. If this does not scale well enough, Apache Hive²⁹ could be integrated to Spark and MinIO in Kubernetes and be used as the offline store.

5.5.3.1 Installation

The configuration file (YAML) of Feast is the following:

```

apiVersion: v1
kind: Namespace
metadata:
  name: feast-fractal
---
apiVersion: batch/v1
kind: Job
metadata:
  name: feast
  namespace: feast-fractal
spec:
  template:
    spec:
      containers:
        - name: feast-image
          image: img/feast:redis-s3-dependencies
          ports:
            - containerPort: 6566
          command: ["bin/bash", "-c", "feast"]
      restartPolicy: Never

```

²⁷ <https://cloud.google.com/bigquery>

²⁸ <https://docs.aws.amazon.com/redshift/latest/mgmt/welcome.html>

²⁹ <https://hive.apache.org/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

Feast has no long running process. It interacts with the various parts of the infrastructure mentioned above and with the development environment only when invoked through the CLI or Python SDK. Thus, there is no need to deploy a Feast service. However, components wishing to update or access the feature store need to access the SDK or CLI. Feast could be added to these components as a dependency. If this is not possible or practical, a “feast-job.yaml” file has been provided. This Kubernetes job can run CLI commands or Python scripts. Feast can be used in the FRACTAL Cloud Platform through this job.

Feast can be installed easily via a pip command and be added to other components this way. To use Redis³⁰ and access S3 storage, it needs two dependencies.

```
pip install 'feast[redis,aws]'
```

Redis can be deployed on Kubernetes using a Helm Chart provided by bitnami.

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install my-release bitnami/redis
```

The feast-job.yaml “command” field can be modified to execute various actions using feast, or to run a python script.

The Feast job can be run using the following command:

```
kubectl apply -f feast-job.yaml
```

To check the actions executed by the job:

```
kubectl -n feast-fractal logs job/feast
```

5.6 Model repository

One of the FRACTAL Cloud Platform’s features is the AI model repository, where AI models are stored to make them available for the rest of the services. There are several options to choose from as a model repository, DVC³¹, MLflow³², and MLBuffet, however, some of them cover the FRACTAL Cloud Platform’s requirements better than others. MLBuffet is a lightweight distributed AI model server which is highly edge-oriented, and although it could be used for model storage in the cloud, it could result in a worse performance than other model repositories specifically developed for cloud instances. Lastly, DVC stands as a good option for model storing and version controlling. A complete installation procedure is given for DVC, MLBuffet and MLflow in the following subsections:

³⁰ <https://redis.io/>

³¹ <https://dvc.org/>

³² <https://mlflow.org/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

5.6.1 DVC

DVC is a framework created to manage the version control of big files, which includes AI models.

DVC (Data Version Control) can be installed in several ways as a pip python package. Here are described two installation methods, the first one being the bare metal installation as a pip Python package, and the second one, as a containerized service, which will be the preferred method for the OVH Cloud.

- Python package: This is the most common way to install DVC. This requires pip as the Python packages manager:

```
$ pip install dvc
```

- Local storage type: This is the default version of DVC. This mode only allows DVC to use local directories as repositories of the model versions.
- Remote storage type: DVC has extensions for the most common external storage: AWS (S3), Google Drive (gdrive), Google Storage Cloud (gs), Microsoft Azure (azure), Aliyun OSS (oss), SSH connection (ssh), HDFS (hdfs), so, to make available the use of that extension, this must be added to the package installation. For instance, if the AWS extension is wanted to be installed, the following command should be written: `pip install "dvc[s3]"`.
- Snap: DVC is available in the snap application repositories. To install it in Linux-based distributions:

```
$ snap install dvc --classic
```

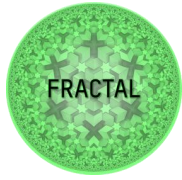
- Official repository: DVC can be installed in apt-managed distributions by adding the official resources to the apt repository:

```
$ sudo wget https://dvc.org/deb/dvc.list -O /etc/apt/sources.list.d/dvc.list
$ wget -qO - https://dvc.org/deb/iterative.asc | sudo apt-key add -
$ sudo apt update
$ sudo apt install dvc
```

- Package: The last way to install DVC is downloading the binary package from the home page (<https://dvc.org>) or the release page on GitHub (<https://github.com/iterative/dvc/releases/>), and executing the file:

```
$ sudo apt install ./dvc_<dvc-release-version>_amd64.deb
```

When the installation ends, to start using the tool on the directory containing the files to be tracked by DVC (usually the root directory of a Git repository), DVC must be started:



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

```
$ dvc init -<flag>
```

For the OVH Cloud instance, a containerized service to be deployed with Kubernetes has been developed. This service utilizes the DVC S3 storage plugin which is also compatible with the MinIO object storage deployed in the FRACTAL Cloud Platform. The container image must be built first, and the Dockerfile is as follows:

```
# syntax=docker/dockerfile:1.2
ARG PYTHON_VERSION=3.8.1
FROM python:${PYTHON_VERSION}
# shows secret from secret location:
RUN mkdir /run/secrets
RUN mkdir /run/secrets/dvc
RUN mkdir /usr/flask_app
RUN --
mount=type=secret,id=secretaccesskey,dst=/run/secrets/dvc/secretaccesskey.txt
cat /run/secrets/dvc/secretaccesskey.txt
RUN --mount=type=secret,id=accesskeyid,dst=/run/secrets/dvc/accesskeyid.txt
cat /run/secrets/dvc/accesskeyid.txt
RUN --mount=type=secret,id=login,dst=/run/secrets/dvc/login.txt cat
/run/secrets/dvc/login.txt
RUN --mount=type=secret,id=password,dst=/run/secrets/dvc/password.txt cat
/run/secrets/dvc/password.txt
RUN pip install --upgrade pip
RUN pip install "dvc[s3]"
COPY main.py /usr/src/flask_app/main.py
RUN pip install flask
# If DVC will not be used inside a Git repository, add the flag --no-scm to
'dvc init' command
RUN dvc init --no-scm
# https://techinplanet.com/installation-dvc-on-minio-storage/
# setup default remote (change "bucket-name" to your minio bucket name)
RUN dvc remote add -d minio s3://bucket-name -f
# add information about storage url (where "https://minio.mysite.com" your
url)
RUN dvc remote modify minio endpointurl https://minio.mysite.com
# add info about login and password
RUN dvc remote modify minio access_key_id my_login
```

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
RUN dvc remote modify minio secret_access_key my_password
ENTRYPOINT FLASK_APP=/usr/src/flask_app/main.py flask run --host=0.0.0.0
```

With this Dockerfile, the secret passing capabilities into the container of the docker build command are used. This is a way to build images with secrets inside which are not stored in the layer history or leave any trace during building. For these secrets to be passed to the docker build utility, they must be given as a parameter and then will be read by the Dockerfile in the RUN instructions, each identified by a unique id.

Docker build command:

```
DOCKER_BUILDKIT=1 docker build --no-cache --secret
id=secretaccesskey,src=secretaccesskey.txt --secret
id=accesskeyid,src=accesskeyid.txt --secret id=login,src=login.txt --secret
id=password,src=password.txt -t dvc .
```

Finally, the container is deployed as a Flask API which executes DVC commands with the DVC installed in the Python base image.

5.6.2 MLBuffet

MLBuffet is a ML orchestration tool which includes model storage features. MLBuffet's source code is available on GitHub (<https://github.com/zyklab/mlbuffet>) and can be downloaded locally with:

```
$ git clone https://github.com/zyklab/mlbuffet.git
```

MLBuffet is a containerized application, and a container orchestrator is required for its deployment. Docker Swarm³³ and Kubernetes are supported as orchestrators and deployment scripts are available for each of these.

Firstly, the container images must be built, and for this purpose, Dockerfiles for each of the microservices are provided to be built with Docker Engine. Also, for automated building of the images, a build.sh script is also provided that builds all the images simultaneously.

```
$ source mlbuffet/deploy/swarm/build.sh
```

Once the images are built, they must be deployed with a container orchestrator. For Docker Swarm deployments, a deploy script deploy.sh can be executed:

```
$ source mlbuffet/deploy/swarm/deploy.sh
```

Then, the script will prompt how many modelhost instances the user wants to deploy. After an integer has been given to the script via terminal, the swarm.yaml configuration file (also provided) will be deployed. This swarm.yaml deployment file

³³ <https://docs.docker.com/engine/swarm/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

can be changed in case the user has any requirements, for example, modifying the volumes to be used to target any other path in the local filesystem.

For Kubernetes deployments, the `deploy/kubernetes` has all the configuration YAML files required to do an automated K8S (Kubernetes) deployment on a dedicated namespace. A deploy script is also provided, and will sequentially apply the configuration files in the `mlbuffet/deploy/kubernetes/autodeploy` directory:

```
$ source mlbuffet/deploy/kubernetes/deploy.sh
```

There is a single configuration file for each of the services, so take this into consideration while deploying, because the Image names must be changed to fit the image name in the local image repository. Since images are built from source and not publicly available on Docker repositories, they must be provided to the local Kubernetes cluster, for example, by uploading them to the FRACTAL Cloud Platform's Harbor repository and then referencing them adequately in the YAML files.

5.6.3 MLflow

MLflow is an open-source platform to manage the ML lifecycle, including:

- Experimentation and reproducibility: allow users to track experiments to record and compare parameters and results.
- A central model registry: allow users to manage models with capabilities for versioning and annotating.
- Deployment: allow users to host models as REST endpoints.

There are two ways to access the model repository, either a UI or an API. The API allows users to integrate and run MLflow in other applications to store models.

MLflow can be installed using pip:

```
$ pip install mlflow
```

Once it is installed, the user can run MLflow's UI with the following instruction:

```
$ mlflow ui
```

The necessary files and instructions to deploy MLflow on Kubernetes for the FRACTAL Cloud Platform are available in this repository on GitHub (uploaded by UOULU): <https://github.com/vahidmohsseni/k8s-mlflow>.

MLflow needs two storage spaces to function.

The backend store is where MLflow Tracking Server stores experiment and runs metadata, as well as params, metrics, and tags for runs. MLflow supports two types of backend stores: *file store* and *database-backed store*.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

The artifact store is a location suitable for large data (such as an S3 bucket or shared NFS file system) and is where clients log their artifact output (for example, models).

Here are the different options for both stores:

<https://mlflow.org/docs/latest/tracking.html#backend-stores>

<https://mlflow.org/docs/latest/tracking.html#artifact-stores>

Once those have been set up, their location must be referenced in the Dockerfile, or in the "mlflow-deployment" YAML file for the Kubernetes deployment.

The full details of the deployment are as follows.

General Instruction:

```
$ git clone https://github.com/vahidmohsseni/k8s-mlflow
$ cd k8s-mlflow
```

To install only in docker:

```
$ cd deploy/docker
$ docker build -t mlflow .
$ docker run -p 8001:8001 mlflow
```

To deploy on Kubernetes:

```
$ chmod +x build.sh
$ ./build.sh
```

5.7 Image repository

The FRACTAL Cloud Platform is composed of multiple components that are deployed in the form of containerized services and microservices on a Kubernetes cluster. These services are developed in the form of container images that are stored on a common repository which Kubernetes can access to deploy the containerized services. Along with these images, different Helm Charts are used to deploy the required Kubernetes resources and specify their configuration. In FRACTAL, the OVH Managed Private Registry³⁴ has been used for the storage of the container images and Helm Charts that will be used for deploying the components that make up the cloud platform.

The OVH Managed Private Registry is a container registry for Docker images built upon Harbor: an open source, cloud native container registry and Helm chart

³⁴ <https://docs.ovh.com/gb/en/private-registry/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

museum that allows to securely store, share, and manage Docker images and Helm charts (a set of manifests that allow to define the required Kubernetes resources and deployments along with their configuration). FRACTAL uses Harbor to store and manage the different Docker images and Helm charts that will be used to deploy the services of the FRACTAL Cloud Platform. In the following sections, details about Harbor and how to deploy the OVH Private Managed Registry are provided.

5.7.1 Harbor

Harbor³⁵ is a cloud native registry with support for both container images and Helm charts. It serves as registry for cloud native environments like container runtimes and orchestration platforms. It also supports role-based access control where users access different repositories through projects and each user can have different permission for images or Helm charts under a project. Harbor also scans images regularly for vulnerabilities and has policy checks to prevent vulnerable images from being deployed. Moreover, Harbor enables a graphical user portal where users can easily browse and search repositories and manage projects.

5.7.2 Deploying OVH Managed Private Registry

The OVH private registry, has been deployed through the OVH Cloud Control Panel³⁶ following the official documentation³⁷. On the OVH Control Panel, once the FRACTAL Project has been selected, in *Containers and orchestration* section *Managed Private Registry* must be selected (see 1 in Figure 30). Then, the desired location for the registry must be chosen (see 2 in Figure 30) as well as the desired name for the registry (see 3 in Figure 30) and the desired size and billing plan (see 4 in Figure 30). Lastly, the *Create* button must be clicked (see 5 in Figure 30) and the OVH Managed Private Registry is created (see the created registry in Figure 31).

³⁵ <https://goharbor.io/>

³⁶ <https://www.ovh.com/manager/hub/#/>

³⁷ <https://docs.ovh.com/gb/en/private-registry/creating-a-private-registry/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

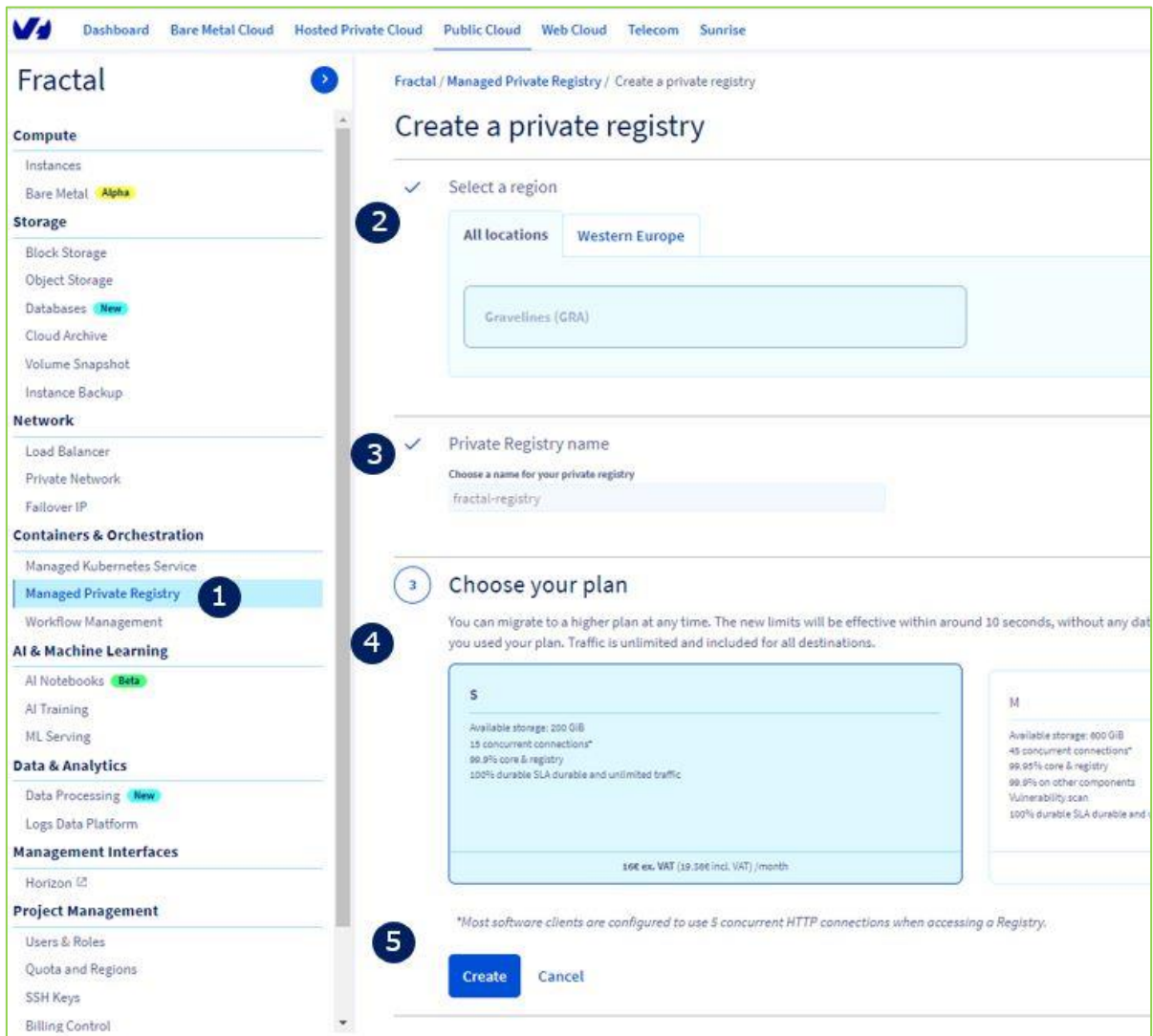


Figure 30: Deploying OVH Managed Private Registry through OVH Cloud Control Panel.

5.7.3 Configuring OVH Managed Private Registry

For using the OVH Managed Private Registry to store container images and Helm charts, first the access to Harbor must be configured, then a project must be created, and users must be added to the project as members.

5.7.3.1 Accessing the OVH Managed Private Registry

When accessing the OVH Managed Private Registry for the first time, the access credentials must be created. For that, after creating the OVH Managed Private registry, the *Generate Identification Details* (see Figure 31) option must be

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

selected³⁸. After generating the access credentials, the registry can be accessed by clicking on *Harbor user interface*³⁹ (see Figure 31).



Figure 31: Created OVH Managed Private Registry

5.7.3.2 Creating a Project in Harbor

Before using the image repository and the Helm Chart Museum of the Harbor Private registry, first a project must be created. Following the official documentation⁴⁰, a new project can be created with *New Project* option and fulfilling the required information in the form shown in Figure 32. In FRACTAL, the *Access Level* has not been set to *public* to restrict repository access only to FRACTAL partners.

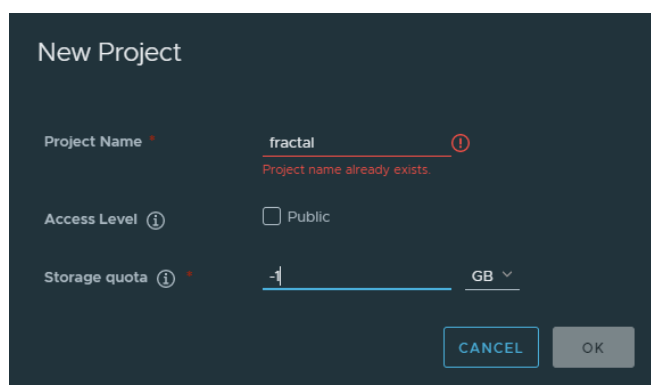


Figure 32: Creating a New Project in Harbor.

Inside the created project, the different container repositories and related resources can be found (see FRACTAL registry in Figure 33).

³⁸ <https://docs.ovh.com/gb/en/private-registry/creating-a-private-registry/>

³⁹ <https://docs.ovh.com/gb/en/private-registry/connecting-to-the-ui/>

⁴⁰ <https://docs.ovh.com/gb/en/private-registry/managing-users-and-projects/#creating-a-new-project>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

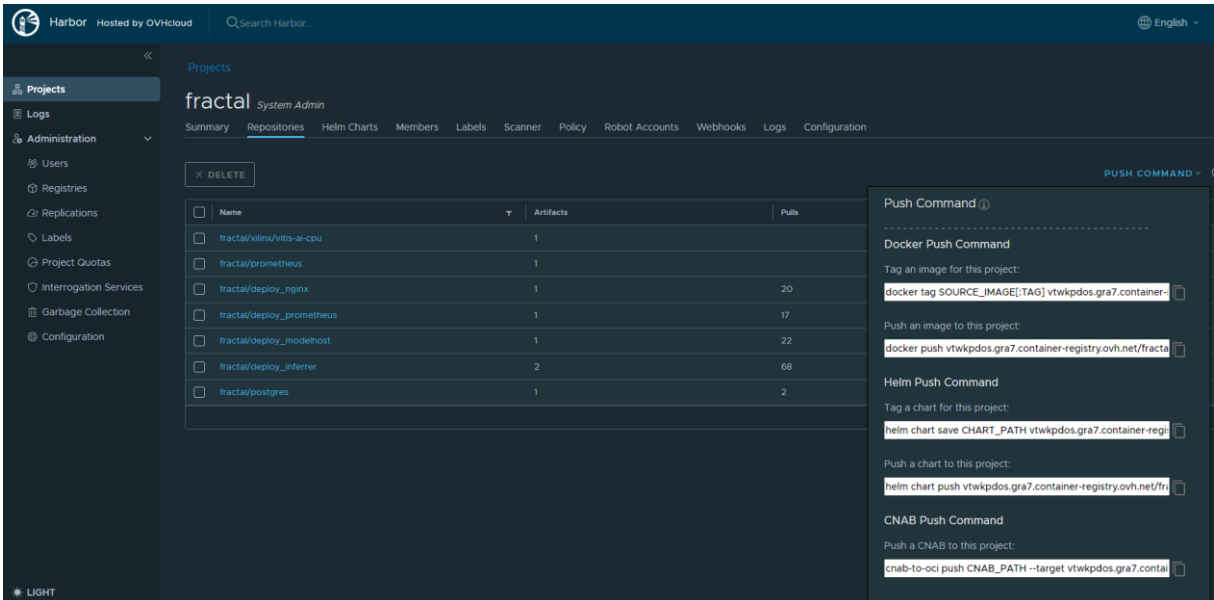


Figure 33: FRACTAL Harbor Private Registry.

5.7.3.3 Adding Users in Harbor

To add a user to Harbor, under *Administration* tab, *Users* option must be selected (Figure 33). A new window will be opened to enter the information of the user to be added (see Figure 34). Once the user has been created, different user management actions can be performed (grant administrator rights, delete user or reset password) from the *Users* control panel (see the official documentation for more details⁴¹). Lastly, in order to add a created user to a project, it must be added from *Members* tab in the Project Control panel (see Figure 33) by choosing the *Add User* option. Once a user has been added to a project, it would be able to pull and push Docker images and Helm Charts.

⁴¹ <https://docs.ovh.com/gb/en/private-registry/managing-users-and-projects/#creating-a-new-user-and-giving-it-rights-on-the-private-project>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

New User

Username *

Email *

First and last name *

Password *

Confirm Password *

Comments

Figure 34: Add a User to Harbor.

5.7.3.4 Creating a Robot Account

The previous section shows how to create an account for a user in Harbor and grant it access to a project. However, for accessing Harbor from some services, a *Service Account* (i.e., *Robot Account*) must be created. This kind of service accounts will be used to grant different services, such as Kubernetes, access the resources in the Harbor repository. For creating a service account, the *Robot Accounts* tab in the project control panel must be selected. A new window will be opened to enter the information of the service account to be added (see Figure 35). Once the service account has been created, the service account credentials (service account name and secret token) will be displayed, and an option to export them to a file will be shown. This credentials, will be the ones used by the different services, such as Kubernetes, to access the private Harbor registry.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

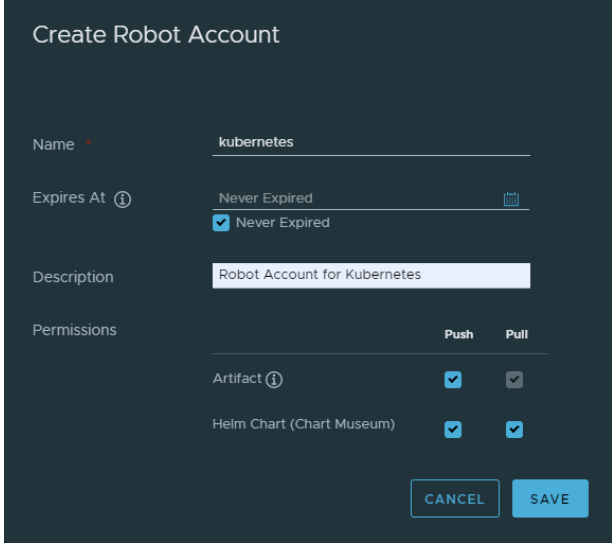


Figure 35: Creating a Robot Account for Kubernetes

5.8 ML orchestration

Orchestration refers to the process of managing resources, and in the case of ML processes, every step of the ML models lifecycle can be orchestrated, from dataset treatment and versioning, to training, storage and inference. These processes are managed by high-level tools which don't get actively involved in the processes themselves but are in charge of the execution and allocation of resources and tasks, which will then be executed by lower-level applications or libraries (ML training libraries, inference runtimes, or model deployment frameworks). There are several tools that can manage the orchestration of ML artifacts, and the installation steps of the ones selected for the FRACTAL Cloud Platform are described in following sections:

5.8.1 MLBuffet

MLBuffet is an open-source application that was developed during the FRACTAL Project from tasks in WP5. Initially, it was thought to be a model server to perform inference on ONNX models, but it evolved to become a multi-library training for models, model storage and inference framework.

MLBuffet installation steps were already described in Section 5.6.2, and its implementation details can be found in the GitHub's README (<https://github.com/zyklab/mlbuffet/blob/master/README.md>).

During the FRACTAL Project, the MLBuffet tool has been evolving and adding new functionalities in terms of artifact orchestration, including model version control, model description, containerized training capabilities and prediction caching to achieve the best performance for distributed Edge deployments. Although these

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

features may not be completely appropriate for a cloud environment, where computational resources are not restrained, MLBuffet can also be deployed on cloud for the developers to have a testing environment and make sure that their deployments will behave as expected once deployed at the edge. Its containerized nature ensures that the artifacts being orchestrated will behave the same on any hardware they are deployed on.

5.8.2 MLflow

MLflow is an open-source platform to manage the ML lifecycle. Installation steps for its setup can be found in Section 5.6.3.

This framework can be used in the machine learning orchestration to train, store and serve models. MLflow is composed of a model repository that can store several experiences and versions of ML models. Its tracking feature allows us to store several data about the training task such as parameters, artifacts (e.g., files of different formats), but also metrics (e.g., accuracy) to compare experiments and versions of a model.

5.8.3 Kubeflow

Kubeflow⁴² is a tool to help users to smoothly implement machine learning workflows on Kubernetes. Since it is based on Kubernetes architecture, the first prerequisite is setting up a K8S (Kubernetes) cluster and the command-line tool kubectl.

Kubeflow allows, through user interface or through scripts, the user to create Pipelines, running hyperparameter optimization (using Katib⁴³) and launching Jupyter Notebook⁴⁴ serves.

Pipelines in Kubeflow are directed acyclic graphs (DAG), including all the components to be computed in the workflow. Each component is a Docker image containing the user code and dependencies to run on Kubernetes.

The Pipelines SDK includes support for constructing and running pipelines from a Jupyter notebook—including the ability to build and compile a pipeline directly from Python code that specifies its functionality, without leaving the notebook. Building a pipeline for creating a machine learning model from data consists of these steps: (i) download data (ii) creating the pipeline (iii) include cloud platform (iv) submit the job for execution.

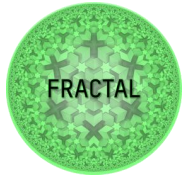
This is the YAML file that can be used for Kubeflow deployment.

```
apiVersion: kfdef.apps.kubeflow.org/v1
```

⁴² <https://www.kubeflow.org/>

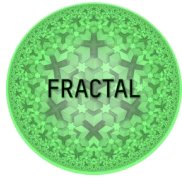
⁴³ <https://github.com/kubeflow/katib>

⁴⁴ <https://jupyter.org/>



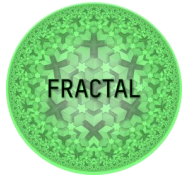
Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

```
kind: KfDef
metadata:
  namespace: kubeflow
spec:
  applications:
  - kustomizeConfig:
    repoRef:
      name: manifests
      path: namespaces/base
    name: namespaces
  - kustomizeConfig:
    repoRef:
      name: manifests
      path: application/v3
    name: application
  - kustomizeConfig:
    repoRef:
      name: manifests
      path: stacks/kubernetes/application/istio-1-3-1-stack
    name: istio-stack
  - kustomizeConfig:
    repoRef:
      name: manifests
      path: stacks/kubernetes/application/cluster-local-gateway-1-3-1
    name: cluster-local-gateway
  - kustomizeConfig:
    repoRef:
      name: manifests
      path: istio/istio/base
    name: istio
  - kustomizeConfig:
    repoRef:
      name: manifests
      path: stacks/kubernetes/application/cert-manager-crds
```



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

```
name: cert-manager-crds
- kustomizeConfig:
  repoRef:
    name: manifests
    path: stacks/kubernetes/application/cert-manager-kube-system-
resources
name: cert-manager-kube-system-resources
- kustomizeConfig:
  repoRef:
    name: manifests
    path: stacks/kubernetes/application/cert-manager
name: cert-manager
- kustomizeConfig:
  repoRef:
    name: manifests
    path: stacks/kubernetes/application/add-anonymous-user-filter
name: add-anonymous-user-filter
- kustomizeConfig:
  repoRef:
    name: manifests
    path: metacontroller/base
name: metacontroller
- kustomizeConfig:
  repoRef:
    name: manifests
    path: admission-webhook/bootstrap/overlays/application
name: bootstrap
- kustomizeConfig:
  repoRef:
    name: manifests
    path: stacks/kubernetes/application/spark-operator
name: spark-operator
- kustomizeConfig:
  repoRef:
```



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

```
    name: manifests
    path: stacks/kubernetes
name: kubeflow-apps
- kustomizeConfig:
  repoRef:
    name: manifests
    path: knative/installs/generic
name: knative
- kustomizeConfig:
  repoRef:
    name: manifests
    path: kfserving/installs/generic
name: kfserving
# Spartakus is a separate applications so that kfctl can remove it
# to disable usage reporting
- kustomizeConfig:
  repoRef:
    name: manifests
    path: stacks/kubernetes/application/spartakus
name: spartakus
repos:
- name: manifests
  uri: https://github.com/kubeflow/manifests/archive/v1.2.0.tar.gz
version: v1.2-branch
```

Note that the instructions given in this chapter are compatible until the 1.21 version of Kubernetes. If a newer version of Kubernetes is used, the installation process would be different, right now, there is no Kubeflow version compatible with Kubernetes 1.22 and above so it has been impossible to put those instructions in this deliverable.

5.9 Workflow management

A workflow is a sequence of operations (tasks) in the FRACTAL Cloud Platform that are executed according to a schedule or triggered by an event. The goal of the workflow management component consists in ensuring that all the different

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

executions are harmonized to avoid conflicts between them and repercussions of possible failures.

In the FRACTAL Cloud Platform, the Airflow⁴⁵ service will be used to provide this functionality.

5.9.1 Airflow

5.9.1.1 Enable nfs file provisioning on Kubernetes

Airflow consists of several pods that are deployed (worker, scheduler, flower...), therefore each pod needs storage. Furthermore, in order to perform correctly the deployment, this storage must be shared among pods. If the cloud provider does not support volumes that are mounted as readWriteMany, a solution for this is to deploy a NFS provisioner which will provide Kubernetes with NFS persistent volumes shared by different pods.

Following are some basic instructions given to deploy an NFS file provisioner in Kubernetes

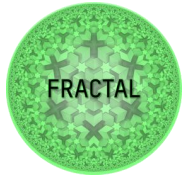
- Add helm chart repository

```
helm repo add kvaps https://kvaps.github.io/charts && helm repo update
```

- Setup the persistence that will back the NFS service by creating a file named `nfs_persistence.yaml`. This file specifies the persistence of the NFS provisioner. The given size here will be the size of the NFS volume.

```
---
# Nfs-provisioner params for Helm install
#
# For more details and possible options please see the table at:
# - <https://github.com/helm/charts/tree/master/stable/nfs-server-provisioner>
persistence:
  # Enables persistence of config values
  # Including the provisioner ID
  # -> Crucial so that the provisioner recognise itself after restarting
  enabled: true
  # Note that if storageClass is not defined,
```

⁴⁵ <https://airflow.apache.org/>



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

```
# Then this parameters defaults to the default storage class in the
cluster,
# which, as of today, in GCP, uses kubernetes.io/gce-pd provisioner
# Careful: Don't use EmptyDir, as it will not survive the pod's death
# -> Config will therefore be lost anyway
#Set OVH StorageClass. We have csi-cinder-classic and csi-cinder-high-speed
storageClass: csi-cinder-high-speed
size: "20Gi"
storageClass:
# Name of the storage class that will be managed by the provisioner
defaultClass: true
```

- Create the nfs namespace

```
kubectl create namespace nfs
```

- Install the helm chart and provide the newly created file that will override the persistence settings.

```
helm install nfs-provisioner kvaps/nfs-server-provisioner --namespace nfs--
version 1.3.1 -f nfs_persistence.yaml
```

- Check that the following resources have been created
 - o NFS provisioner pod named nfs-provisioner-nfs-server-provisioner-0
 - o storageClass named nfs
 - o PersistentVolume of 20GB
 - o PersistentVolumeClaim named data-nfs-provisioner-nfs-server-provisioner-0

5.9.1.2 Install Airflow

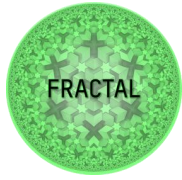
Airflow will be installed through a helm chart provided by the community, which is branched from the official Airflow chart, but with extended support and documentation. Before proceeding with the installation, two PersistentVolumeClaims need to be created, one for the log persistence and another for Airflow's DAG storage.

- Create airflow namespace

```
kubectl create namespace airflow
```

- Create a file named PersistentVolumeClaim.yaml with the following content, specifying each claim size and storageClass.

```
#PVC for airflow logs.
```



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim-airflow-logs
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: nfs
  resources:
    requests:
      storage: 15Gi
---
#PVC for airflow dags.
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-claim-airflow-dags
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: nfs
  resources:
    requests:
      storage: 2Gi
```

- Apply file to create the resources

```
kubectl create -f persistentVolumeClaim.yaml -n airflow
```

- Add airflow repository to helm

```
helm repo add airflow-stable https://airflow-helm.github.io/chartshelm
repo update
```

- Install helm chart specifying the persistence

```
helm install fractal-airflow airflow-stable/airflow --namespace airflow --
version 8.4.1 --values ./airflow_log_dag_persistence.yaml
```

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

- Expose airflow web as a LoadBalancer service in order to access it from the internet

```
kubectl expose service/fractal-airflow-web --type=LoadBalancer --port=8080 --name fractal-airflow-web-external -n airflow
```

- Check the external IP address to access Airflow web using the following command

```
kubectl get service fractal-airflow-web-external -n airflow -o jsonpath="{.status.loadBalancer.ingress[0].ip}"
```

- Web UI can be accessed through the returned IP on the port 8080

5.10 Model preparation for FRACTAL Edge

For a generic path to deploy models from the framework description to an actual physical acceleration target, a target-specific implementation must be available. The FRACTAL cloud services can serve to augment the framework-based model with such target-specific artifacts, thus providing model preparation support. From the elements presented in Sections 5.6, 5.7, 5.8 and 5.9 the chosen approach is to:

1. Utilize MLflow to store framework-based models in the MLflow repository.
2. Save these models into volumes that are handed into the processing steps.
3. Trigger an Airflow DAG execution to do the actual preparation steps.
4. Load the augmented model back into the MLflow repository.

These processing steps are collected in an Airflow DAG that binds the data volumes to target specific services. The target-specific services are made available as a single container for the given implementation for the FRACTAL platforms running on Versal VCK190. The container is hosted in Harbor as a prebuilt docker image. This scenario can support other target preparation steps by exchanging the proper combination of the Airflow DAG and the docker image. Any preprocessing of the model data that needs to be accounted for the DAG can add tasks to handle the specifics for the docker image to operate correctly.

To obtain a build of the image for the Xilinx Versal VCK190 development kit the complete base Vitis AI repository can be obtained from GitHub:

```
git clone https://github.com/Xilinx/Vitis-AI.git
```

This checkout provides a wrapper script for the docker environment based on Ubuntu 20.04 with a Kernel rev. 5.13. Versions of git checkout above and subsequent image build must match the versions of any earlier model verification.

```
docker_run.sh xilinx/vitis-ai-cpu:2.0
```

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

The docker image setup, parameters, and mount points are collected and used in the Airflow DAG, locally using a *DockerOperator*, and in the Cloud using a *KubernetesPodOperator* to start the container. The dependencies are shown here for local operation:

```

t1 = DockerOperator(
    task_id = 'vitis_ai_quantizer',
    # image = 'vtwkpdos.gra7.container-
registry.ovh.net/fractal/xilinx/vitis-ai-cpu:latest',
    image = 'xilinx/vitis-ai-cpu:latest',
    api_version='auto',
    auto_remove=False,
    #command = ["bash /workspace/source_bashrc.sh ", "bash
/workspace/docker_check.sh ", ],
    command = ' bash /workspace/prepare_model_full.sh',
    #command = "bash /workspace/docker_check.sh ",
    docker_url="unix://var/run/docker.sock",
    network_mode="host",
    environment = {'USER':os.environ.get('USER'),'UID':1001,'GID':1001},
    # user = None,
    mounts = [
        Mount(source='/opt/xilinx/dsa',target='/opt/xilinx/dsa:rw',
type='bind'),
        Mount(source = "/opt/xilinx/overlaybins",target=
"/opt/xilinx/overlaybins", type='bind'),
        Mount(source = "/etc/xbutler",target = "/etc/xbutler", type='bind'
),
        Mount(source="/dev/shm",target="/dev/shm", type='bind'),
        Mount(source = "/home/clouduser/vitis-workspace", target =
"/workspace" , type = 'bind')],
    working_dir = '/workspace'
)

```

These and similar tasks may be called either as a single DAG element to process the full model compilation at once or split into multiple subtasks to give a finer granularity control to the Airflow scheduler.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

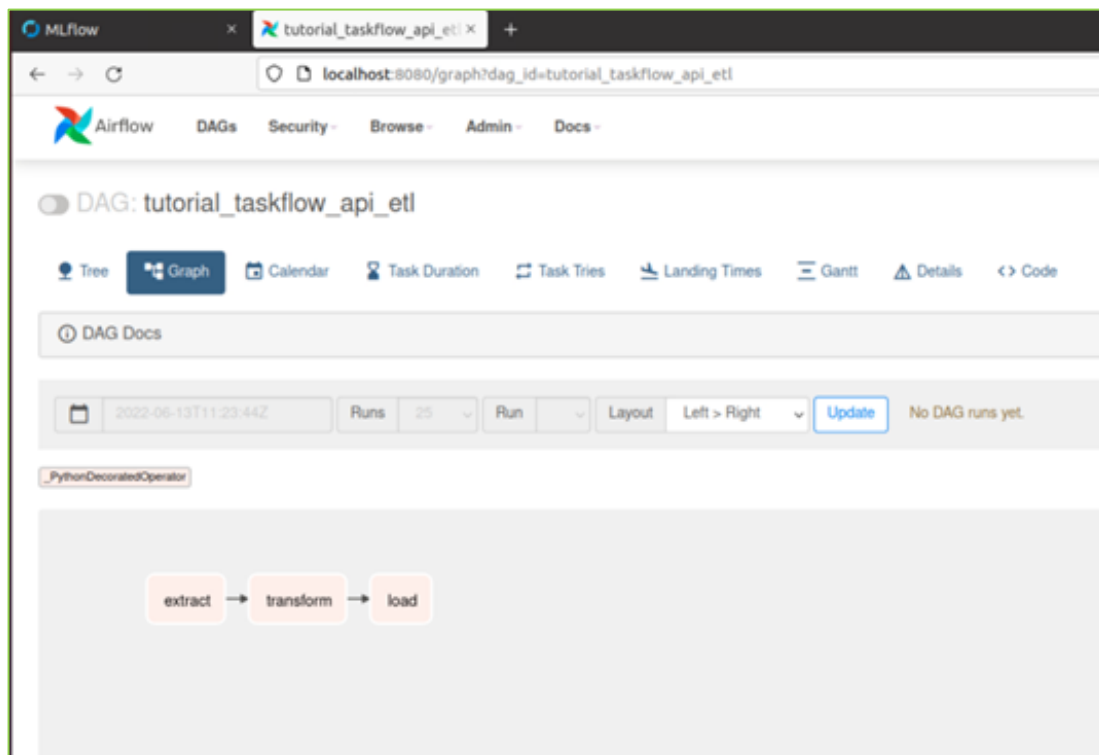


Figure 36: Airflow DAG model preparation

5.11 Platform infrastructure

The FRACTAL Cloud Platform is deployed over the infrastructure of a public cloud services provider, which offers different services such as storage space, computing power, networks, and other fundamental computing resources.

5.11.1 Configuring the OVH Managed Kubernetes

FRACTAL will use a Kubernetes cluster to deploy, manage and execute the different services developed alongside the different partners of the FRACTAL Project.

The OVH Managed Kubernetes service⁴⁶ has been deployed through the OVH Cloud Control Panel following the official documentation⁴⁷. On the OVH Control Panel, once the FRACTAL Project has been selected, in *Containers and orchestration* section *Managed Kubernetes Service* must be selected (see Figure 37). Then, the following steps must be accomplished:

1. The desired location for the cluster must be chosen.
2. Select the desired version of Kubernetes.

⁴⁶ <https://www.ovhcloud.com/es-es/public-cloud/kubernetes/>

⁴⁷ <https://docs.ovh.com/gb/en/kubernetes/creating-a-cluster/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

3. Select a private network for the cluster (or *None* for using public IPs). In this case a private network created beforehand has been selected.
4. Select the desired instance type for the nodes that conforms the Kubernetes cluster. In this case *B2-15*⁴⁸ instance type has been selected.
5. Configure the desired node pool size (i.e., the number of instances that compose the cluster). In this case a 3-nodes cluster size has been selected with the *autoscaling* option disabled by default.
6. Choose the desired billing type.
7. Select a name for the cluster (*fractal-kubernetes-cluster* in this case).

Lastly, the *Create* button must be clicked and the OVH Managed Kubernetes service is created (see the created registry in Figure 37).

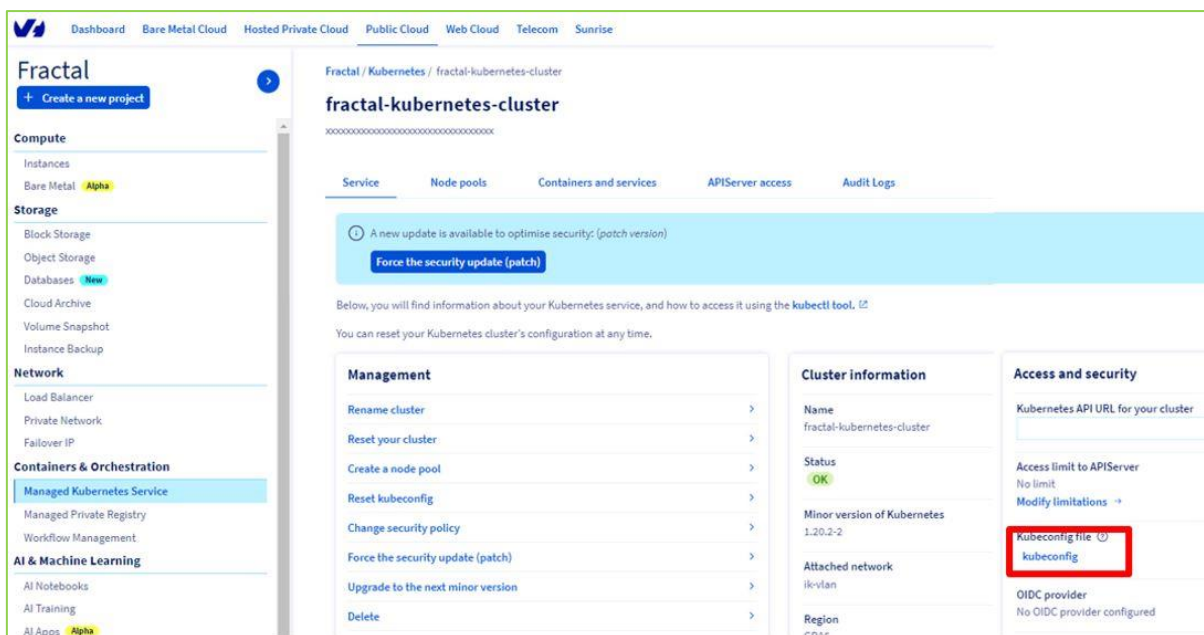


Figure 37: Created fractal-kubernetes-cluster.

5.11.2 Accessing the OVH Managed Kubernetes Service

In order to be able to access the OVH Managed Kubernetes Service, a 'kubeconfig' file will be used. This file can be downloaded from the Kubernetes cluster configuration pane (see the highlighted section in Figure 37). Then 'kubectl' tool must be installed⁴⁹ and configured⁵⁰. On Linux systems this can be achieved with the following commands:

⁴⁸ <https://us.ovhcloud.com/public-cloud/prices/>

⁴⁹ <https://kubernetes.io/docs/tasks/tools/>

⁵⁰ <https://docs.ovh.com/gb/en/kubernetes/configuring-kubectl/#step-1-configure-the-default-settings-for-kubectl>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

1. Update the apt package index and install packages needed to use the Kubernetes apt repository:

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
```

2. Download the Google Cloud public signing key:

```
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

3. Add the Kubernetes apt repository:

```
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
```

4. Update apt package index with the new repository and install kubectl:

```
sudo apt-get update
sudo apt-get install -y kubectl
```

5. Load the downloaded kubeconfig file:

```
export KUBECONFIG=/Users/myuser/.kube/my-test-cluster.yml
```

5.11.3 Deploy and access the Kubernetes Dashboard

The Kubernetes Dashboard⁵¹ is a web-based Kubernetes user interface, that can be used to deploy containerized applications to a Kubernetes cluster, troubleshoot containerized applications, and manage the cluster resources. It also allows to get an overview of applications running on a cluster, as well as for creating or modifying individual Kubernetes resources (such as Deployments, Jobs, Daemon Sets, etc.).

For installing the Kubernetes Dashboard on OVH Cloud Managed Kubernetes Service, the official documentation has been⁵² followed. Check the following subsections for more details.

5.11.3.1 Install & Configure Kubernetes Dashboard

Kubernetes Dashboard can be installed using the recommended YAML file⁵³ and the following command:

⁵¹ <https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/>

⁵² <https://docs.ovh.com/gb/en/kubernetes/installing-kubernetes-dashboard/>

⁵³ <https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/#deploying-the-dashboard-ui>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.3.1/aio/deploy/recommended.yaml
```

With the following command, it can be checked that the Kubernetes Dashboard is deployed and running (see Figure 38).

```
kubectl get all -n kubernetes-dashboard
```

```

NAME                                READY   STATUS    RESTARTS   AGE
pod/dashboard-metrics-scraper-79c5968bdc-njjtc  1/1     Running   0           27h
pod/kubernetes-dashboard-658485d5c7-r59l7       1/1     Running   0           27h

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
service/dashboard-metrics-scraper  ClusterIP    10.3.127.254  <none>        8000/TCP   27h
service/kubernetes-dashboard        ClusterIP    10.3.52.134   <none>        443/TCP    27h

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/dashboard-metrics-scraper  1/1     1             1           27h
deployment.apps/kubernetes-dashboard        1/1     1             1           27h

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/dashboard-metrics-scraper-79c5968bdc  1         1         1       27h
replicaset.apps/kubernetes-dashboard-658485d5c7      1         1         1       27h

```

Figure 38: Kubernetes Dashboard Service.

5.11.3.2 Accessing the Kubernetes Dashboard

5.11.3.2.1 Create Service Account

In order to access the Dashboard, a new user must be created with the service account mechanism in Kubernetes and granted this user admin permissions. For that, the service-account.yml will be used (see Figure 39). When applying with kubectl, this YAML file creates a service account (admin-user) in Kubernetes.

```
kubectl apply -f ./Kubernetes-Dashboard/service-account.yml
```

```

service-account.yml 104 Bytes
1 ---
2 apiVersion: v1
3 kind: ServiceAccount
4 metadata:
5   name: admin-user
6   namespace: kubernetes-dashboard
7

```

Figure 39: service-account.yml file content.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

5.11.3.2.2 Create a Role Binding

Then a RoleBinding will be created for binding the cluster-admin role in the Kubernetes cluster to the created ServiceAccount. For this, the cluster-role-binding.yml will be used (see Figure 40).

```
kubectl apply -f ./Kubernetes-Dashboard/cluster-role-binding.yml
```

```

cluster-role-binding.yml 275 Bytes
1 ---
2 apiVersion: rbac.authorization.k8s.io/v1
3 kind: ClusterRoleBinding
4 metadata:
5   name: admin-user
6 roleRef:
7   apiGroup: rbac.authorization.k8s.io
8   kind: ClusterRole
9   name: cluster-admin
10 subjects:
11 - kind: ServiceAccount
12   name: admin-user
13   namespace: kubernetes-dashboard
14

```

Figure 40: cluster-role-binding.yml

5.11.3.2.3 Get Bearer Token

The created service account and role binding will allow to get an access token for the Kubernetes dashboard. The token can be retrieved with the following command:

```
kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get secret | grep admin-user-token | awk '{print $1}')
```

5.11.3.2.4 Exposing the Dashboard with Ingress

The Kubernetes dashboard can be accessed by first exposing it using Ingress or a Load Balancer. For that, the Ingress service should be installed and configured (check Section 6.10). Kubernetes Dashboard can be exposed with Ingress by applying the ingress-kubernetes-dashboard.yml YAML file (see Figure 41).

```
kubectl apply -f ./Kubernetes-Dashboard/ingress-kubernetes-dashboard.yml
```

This will expose the dashboard at [https://\[INGRESS_IP_OR_DOMAIN\]/dashboard/](https://[INGRESS_IP_OR_DOMAIN]/dashboard/).

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```

ingress-kubernetes-dashboard.yml 526 Bytes
1 ---
2 apiVersion: networking.k8s.io/v1
3 kind: Ingress
4 metadata:
5   name: kubernetes-dashboard-endpoint
6   namespace: kubernetes-dashboard
7   annotations:
8     kubernetes.io/ingress.class: "nginx"
9     nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
10    nginx.ingress.kubernetes.io/rewrite-target: /$1
11 spec:
12   rules:
13   - host:
14     http:
15       paths:
16       - backend:
17         service:
18           name: kubernetes-dashboard
19           port:
20             number: 443
21         path: /dashboard/(?.*)
22         pathType: Prefix
23

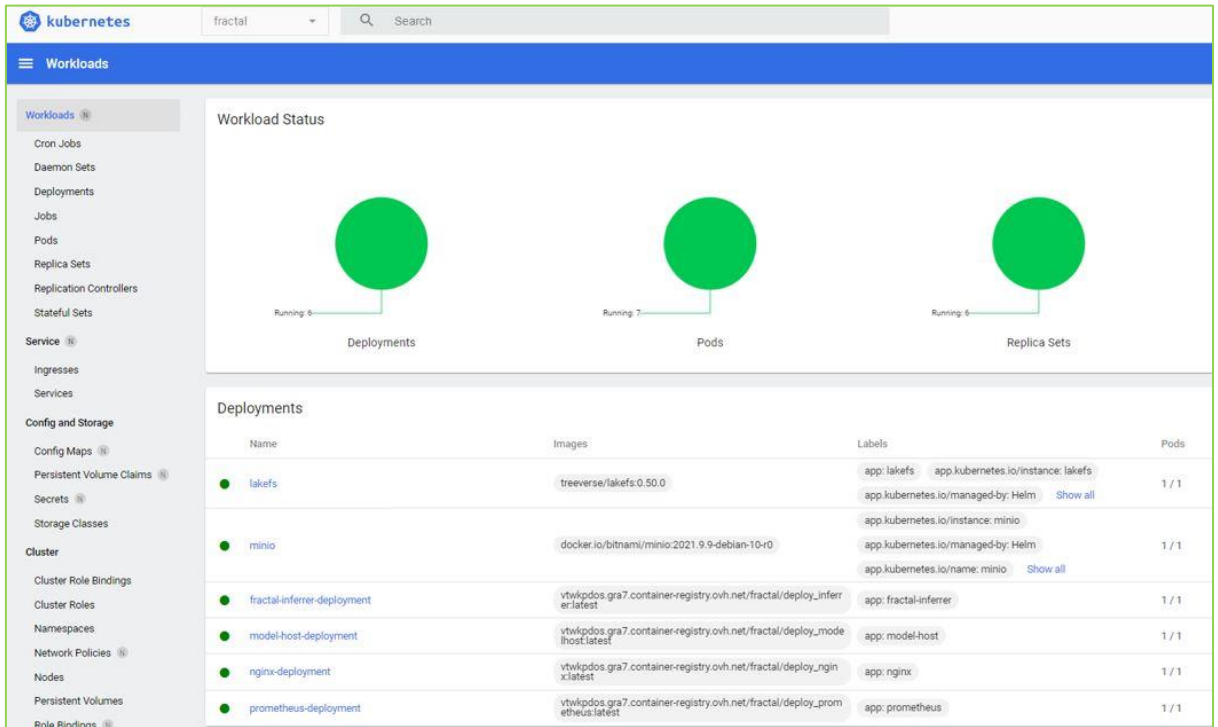
```

Figure 41: ingress-kubernetes-dashboard.yml file contents.

5.11.3.2.5 Accessing the Kubernetes-dashboard

When accessing the dashboard, it will redirect to the login endpoint on which an access token must be provided in order to access the dashboard. By using the obtained token in the previous section, the Kubernetes Dashboard can be accessed (see Figure 42).

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		



The screenshot shows the Kubernetes dashboard for the 'fractal' namespace. The 'Workload Status' section displays three green circles representing the status of Deployments, Pods, and Replica Sets. Below this, the 'Deployments' table lists several deployments with their respective images, labels, and pod counts.

Name	Images	Labels	Pods
lakefs	treeverse/lakefs:0.50.0	app: lakefs app.kubernetes.io/instance: lakefs app.kubernetes.io/managed-by: Helm	1 / 1
minio	docker.io/bitnami/minio:2021.9.9-debian-10-r0	app.kubernetes.io/instance: minio app.kubernetes.io/managed-by: Helm	1 / 1
fractal-inferer-deployment	vtwkpodos.gra7.container-registry.ovh.net/fractal/deplo_inferer:latest	app: fractal-inferer	1 / 1
model-host-deployment	vtwkpodos.gra7.container-registry.ovh.net/fractal/deplo_mode:latest	app: model-host	1 / 1
nginx-deployment	vtwkpodos.gra7.container-registry.ovh.net/fractal/deplo_nginx:latest	app: nginx	1 / 1
prometheus-deployment	vtwkpodos.gra7.container-registry.ovh.net/fractal/deplo_prometheus:latest	app: prometheus	1 / 1

Figure 42: Kubernetes dashboard.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

6. Cloud platform use guidelines

This section presents the usage guidelines of the cloud components developed and deployed in the OVH cloud platform. These components will compose the FRACTAL Cloud Platform and will enable the implementation of the functionalities required from the Use Cases. The main goal of this section is to give some hints on the use of each component so the use cases are aware and familiar with the usage of same.

6.1 Data ingestion

In FRACTAL Cloud Platform, data ingestion service is provided using Kafka platform. Typically, the data is sent from the IoT devices (in this case by the edge nodes), using the MQTT protocol. Kafka platform offers a distributed event streaming platform for high performance data pipelines, streaming analytics, data integration and mission critical applications

6.1.1 Kafka platform

The Kafka cluster can be accessed using the links provided Table 1 in Section 5.2.1.3. In order to send data to Kafka, the corresponding topics have to be created in the Kafka cluster. This can be done using the YAML definitions which have to be applied to the cluster as shown below

For examples referring to the usage of Kafka cluster, check the GitHub repo in the following link: <https://github.com/harisyammnv/kafka-stream-ovh-fractal/tree/master/Data-Streaming>

6.1.1.1 Creating Topics

To create Topics in the Kafka cluster, a YAML file has to be defined with the topic name and the cluster identifier as shown

```

1  apiVersion: kafka.strimzi.io/v1beta1
2  kind: KafkaTopic
3  metadata:
4    name: <topic name here>
5    labels:
6      strimzi.io/cluster: <cluster identifier here>
7  spec:
8    replicas: 3
9    partitions: 10

```

Figure 43: Kafka topic YAML

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

Once the YAML is ready with new values, it has to be applied against the kubectl cluster using the following command

```
kubectl apply -f kafkatopic.yml -n <namespace>
```

To check if the topic has been created on the cluster, use the following command

```
kubectl get kafkatopics.kafka.strimzi.io -n <namespace>
```

6.1.1.2 Sending Messages to Topics

To send the messages to the topics using the topics created in the previous step, a python module has to be used. The GitHub repo in the following link: <https://github.com/harisyamnv/kafka-stream-ovh-fractal/tree/master/Data-Streaming>, shows the steps to install the python-Kafka library. The following Python file illustrates the steps to connect to the bootstrap Kafka cluster which enables the Kafka Producer module to send or stream messages to the topic.

```

1  from kafka import KafkaProducer
2  from confluent_kafka.avro import AvroProducer
3  from datetime import datetime
4  import time
5  from json import dumps
6  import random
7
8  KAFKA_TOPIC_NAME_CONS = "<topic-name>"
9  KAFKA_BOOTSTRAP_SERVER_CONS = "kafka-bs.fractal-kafka.ovh:9092" #
10
11 if __name__ == "__main__":
12     print("Simple Kafka Producer Application Started ...!!!")
13
14     kafka_producer_obj = KafkaProducer(bootstrap_servers = KAFKA_BOOTSTRAP_SERVER_CONS,
15                                     value_serializer = lambda x: dumps(x).encode('utf-8'),
16                                     acks="all")
17     transaction_card_type = ["Visa", "MasterCard", "Maestro"]
18
19     key = "test-key".encode("utf-8")
20
21     message = None
22
23     for i in range(25):
24         i = i+1
25         message = {}
26         print(f"Sending message {i} to the topic {KAFKA_TOPIC_NAME_CONS}")
27         event_datetime = datetime.now()
28
29         message["transaction_id"] = str(i)
30         message["transaction_card_type"] = random.choice(transaction_card_type)
31         message["transaction_amount"] = round(random.uniform(5.5, 555.5), 2)
32         message["transaction_datetime"] = event_datetime.strftime("%Y-%m-%d %H-%M-%S")
33         print(f"Message to be sent: {message}")
34
35         kafka_producer_obj.send(topic=KAFKA_TOPIC_NAME_CONS, key=key,
36                                value=message)
37         time.sleep(1)

```

Figure 44: Streaming messages to Topics

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

Executing the simple producer script from the command line with python would send the messages to the Kafka Topics

```
$ python3 simpler_producer.py
```

```

nlsbhabh 43430@INGURWIN120054:~$ python3 simpler_producer.py
Simple Kafka Producer Application Started ...!!!
Sending message 1 to the topic my-topic-1
Message to be sent: {'transaction_id': '1', 'transaction_card_type': 'Visa', 'transaction_amount': 163.09, 'transaction_datetime': '2022-07-17 15-37-35'}
Sending message 2 to the topic my-topic-1
Message to be sent: {'transaction_id': '2', 'transaction_card_type': 'MasterCard', 'transaction_amount': 491.19, 'transaction_datetime': '2022-07-17 15-37-37'}
Sending message 3 to the topic my-topic-1
Message to be sent: {'transaction_id': '3', 'transaction_card_type': 'Maestro', 'transaction_amount': 508.89, 'transaction_datetime': '2022-07-17 15-37-38'}
Sending message 4 to the topic my-topic-1
Message to be sent: {'transaction_id': '4', 'transaction_card_type': 'MasterCard', 'transaction_amount': 519.51, 'transaction_datetime': '2022-07-17 15-37-40'}
Sending message 5 to the topic my-topic-1
Message to be sent: {'transaction_id': '5', 'transaction_card_type': 'Maestro', 'transaction_amount': 293.9, 'transaction_datetime': '2022-07-17 15-37-41'}
Sending message 6 to the topic my-topic-1
Message to be sent: {'transaction_id': '6', 'transaction_card_type': 'Maestro', 'transaction_amount': 317.39, 'transaction_datetime': '2022-07-17 15-37-42'}
Sending message 7 to the topic my-topic-1
Message to be sent: {'transaction_id': '7', 'transaction_card_type': 'Maestro', 'transaction_amount': 414.07, 'transaction_datetime': '2022-07-17 15-37-44'}

```

Figure 45: Executing the simple_producer.py

6.1.1.3 Consuming Messages to Topics

For consuming the messages produced in the previous section, the Python file for the Kafka Consumer i.e., `simple_consumer.py` from the GitHub repo <https://github.com/harisymmny/kafka-stream-ovh-fractal/tree/master/Data-Streaming> is used. The python file is run from the command line to extract messages from the topic as shown below:

```
$ python3 simpler_consumer.py
```

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```

from kafka import KafkaConsumer
from confluent_kafka.avro import AvroProducer
from datetime import datetime
import time
from json import loads
import random
KAFKA_TOPIC_NAME_CONS = "my-topic-1"
KAFKA_BOOTSTRAP_SERVER_CONS =
" 141.95.96.119:9094"
# kafka-bs.fractal-kafka.ovh:9092

if __name__ == "__main__":
    print(
"Simple Kafka Consumerr Application Started
...!!!"
)
    kafka_consumer_obj = KafkaConsumer(
KAFKA_TOPIC_NAME_CONS,
bootstrap_servers =
KAFKA_BOOTSTRAP_SERVER_CONS,
auto_offset_reset='earliest',
api_version=(0, 10, 1),enable_auto_commit=True)

for message in kafka_consumer_obj:
    print(message.value)

```

Figure 46: Using Kafka Consumer to extract messages

6.2 Raw data storage

For connecting to OVH object storage from Python, the module from Amazon SDK, i.e. the boto3 library, has to be installed in the Python environment.

```
pip install boto3
```

The following snippet of Python code will send messages to the object storage. The `s3_region` is the region where the storage was created, and `access_key` and `secret_key` were obtained in the previous chapter:

```

session = boto3.Session(aws_access_key_id=access_key,
aws_secret_access_key=secret_key)
s3_client = session.client('s3',
endpoint_url=f"https://s3.{s3_region}.cloud.ovh.net/",
region_name=s3_region)

```

The following line will create new bucket on the OVH Object Storage

```
s3_client.create_bucket(Bucket=bucket_name,
CreateBucketConfiguration={'LocationConstraint': s3_region})
```

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

To upload some data to the OVH Object Storage, the following line has to be executed. Beware: filename and bucket name are regular Python strings, but data has to be encoded into the byte array format.

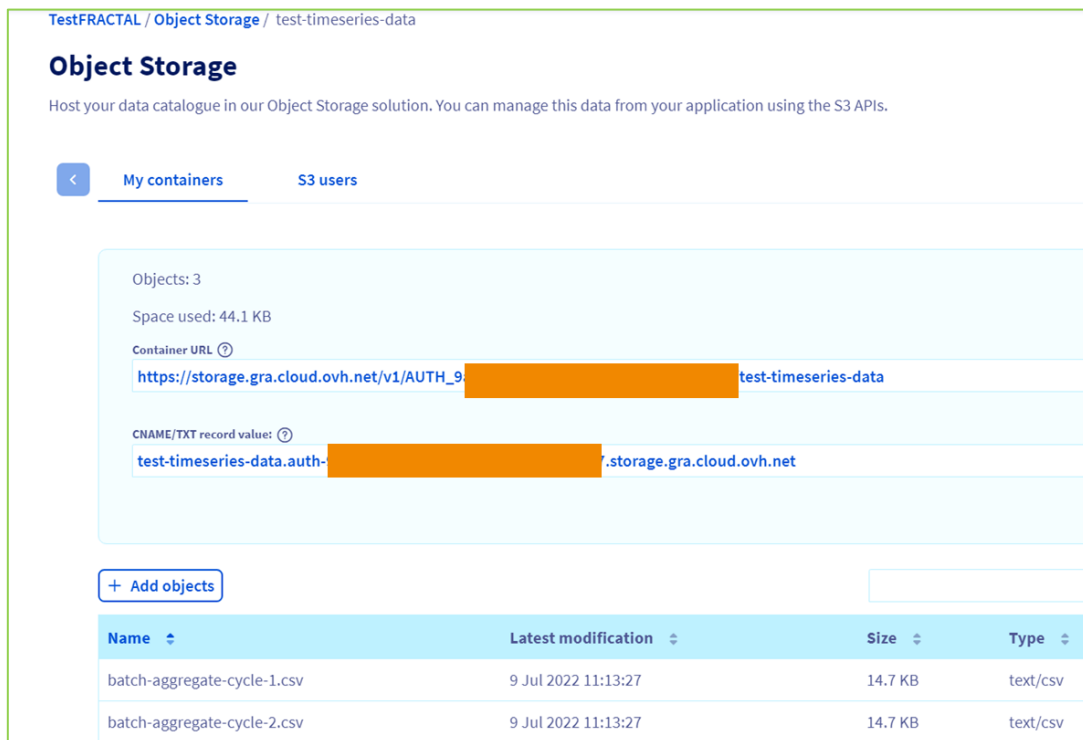
```
s3_client.upload_fileobj(Fileobj=io.BytesIO(data), Bucket=bucket_name, Key=filename)
```

Other sample scripts can be accessed in the following link: <https://github.com/harisymmnb/kafka-stream-ovh-fractal/blob/master/Data-Processing/s3-boto-sample.py>

6.3 Data preprocessing & feature extraction

6.3.1.1 Timeseries data processing

Typically, the data coming from IoT devices are sensor measurements which generally have associated timestamps with the readings. The readings are sent generally at a fixed sampling rate or on value change to MQTT brokers. Kafka connect, which accesses the MQTT brokers, forwards the sensor data as JSON objects to Kafka cluster. A python job generally aggregates the raw data and stores them into batch collected CSV object in the object storage. For example, the object storage could contain batch files as shown in Figure 47



TestFRACTAL / Object Storage / test-timeseries-data

Object Storage

Host your data catalogue in our Object Storage solution. You can manage this data from your application using the S3 APIs.

My containers S3 users

Objects: 3
Space used: 44.1 KB

Container URL [https://storage.gra.cloud.ovh.net/v1/AUTH_9\[redacted\]test-timeseries-data](https://storage.gra.cloud.ovh.net/v1/AUTH_9[redacted]test-timeseries-data)

CNAME/TXT record value: [test-timeseries-data.auth-\[redacted\].storage.gra.cloud.ovh.net](test-timeseries-data.auth-[redacted].storage.gra.cloud.ovh.net)

+ Add objects

Name	Latest modification	Size	Type
batch-aggregate-cycle-1.csv	9 Jul 2022 11:13:27	14.7 KB	text/csv
batch-aggregate-cycle-2.csv	9 Jul 2022 11:13:27	14.7 KB	text/csv

Figure 47: Batch aggregates of raw drive cycles

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

Using the CSV files in the object container, the data preprocessing can be executed using the following code snippet, which in this case as an example, combines the drive cycles and calculates the distance travelled using vehicle speed and time.

```

1  from __future__ import print_function
2  from operator import add
3  from pyspark.sql import SparkSession
4  from pyspark.sql.functions import col
5  import toml
6
7  if __name__ == "__main__":
8
9      config = toml.load("config.toml")
10     # create a SparkSession
11     # We want to use the Swift S3 API. So we have to provide some attributes
12     spark = SparkSession\
13         .builder\
14         .appName("PythonWordCount") \
15         .config("spark.hadoop.fs.s3a.access.key", config["S3"].get('s3_access_key')) \
16         .config("spark.hadoop.fs.s3a.secret.key", config["S3"].get('s3_secret_key')) \
17         .config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")\
18         .config("spark.hadoop.fs.s3a.path.style.access", "true")\
19         .config("spark.hadoop.fs.s3a.endpoint", "s3.gra.cloud.ovh.net")\
20         .getOrCreate()
21
22     s3_folder_path = "s3a://test-timeseries-data/"
23     df = spark.read.csv(s3_folder_path)
24     # to calculate distance = speed [km/hr] * (time[s]/3600)
25     df.withColumn("distance_travelled", col("Speed") * (col("time")/3600))
26
27     df.write.csv("s3a://test-timeseries-data/csv/final-drive-cycle.csv")

```

Figure 48: Timeseries data processing script

This code can be uploaded in the data processing section to create a spark job which is shown in the next sub-section. More complex data processing scripts are available in the GitHub repo, in the following link <https://github.com/harisymm/nv/kafka-stream-ovh-fractal/tree/master/Data-Processing>.

6.3.1.2 Multimedia data processing

For Computer Vision applications, the raw data is in the form of images various sizes. To use the images for training a ML model, the images have to be converted into tensors according to the model architecture. When there are thousands of images this process takes time and spark can be used to speed up the process, using snippets as the one shown in Figure 49.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```

1 import os
2 from pyspark.sql import SparkSession
3 import pyspark.sql.functions as F
4 from pyspark.ml.image import ImageSchema
5 from pyspark.ml.linalg import DenseVector, VectorUDT
6 import toml
7
8 # S3 credentials are stored in Config TOML
9 config = toml.load("config.toml")
10 # create a SparkSession
11 # We want to use the Swift S3 API. So we have to provide some attributes
12 spark = SparkSession\
13     .builder\
14     .appName("PythonWordCount") \
15     .config("spark.hadoop.fs.s3a.access.key", config["S3"].get('s3_access_key')) \
16     .config("spark.hadoop.fs.s3a.secret.key", config["S3"].get('s3_secret_key')) \
17     .config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")\
18     .config("spark.hadoop.fs.s3a.path.style.access", "true")\
19     .config("spark.hadoop.fs.s3a.endpoint", "s3.gra.cloud.ovh.net")\
20     .getOrCreate()
21
22 # Object Storage bucket where images are present
23 s3_url = "s3a://test-image-data-processing/*"
24 df = spark.read.format("image").load(s3_url)
25
26 img2vec = F.udf(lambda x: DenseVector(ImageSchema.toNDArray(x).flatten()),
27                 VectorUDT())
28
29 df = df.withColumn('vecs', img2vec("image"))
30 # object storage bucket where the output is saved
31 df.write.csv('s3a://test-data-wc/images_info.csv')

```

Figure 49: Image processing snippet

The environment YAML file is as shown below

```

1 name: SparkEnv
2 dependencies:
3   - python=3.9
4   - numpy=1.21.*
5   - pip:
6     - Flask
7     - toml

```

Figure 50: Sample environment file

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

The data processing job can be setup from the OVH cloud terminal as shown in Figure 51.

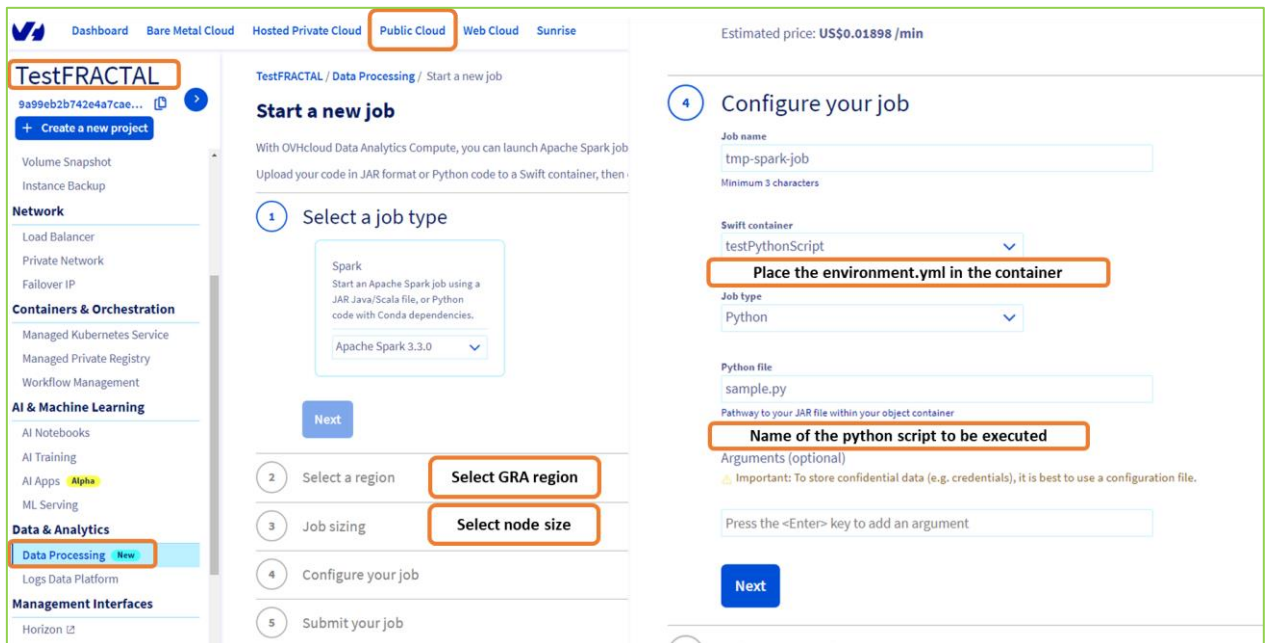


Figure 51: Setting up the spark job

Similar to the image processing script, for more reference, there are text processing script samples provided in the GitHub repo, in the following link <https://github.com/harisymmnnv/kafka-stream-ovh-fractal/blob/master/Data-Processing/word-count-s3.py>

6.4 Dataset repository and feature store

6.4.1 lakeFS

6.4.1.1 Creating users & permissions

To add a new user, it will be necessary to go to the administration panel and add the corresponding user. It is worth mentioning that once the user is created, it will be necessary to assign the new user to a user group. User groups are used to assign specific policies (permissions) to the members of each group.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

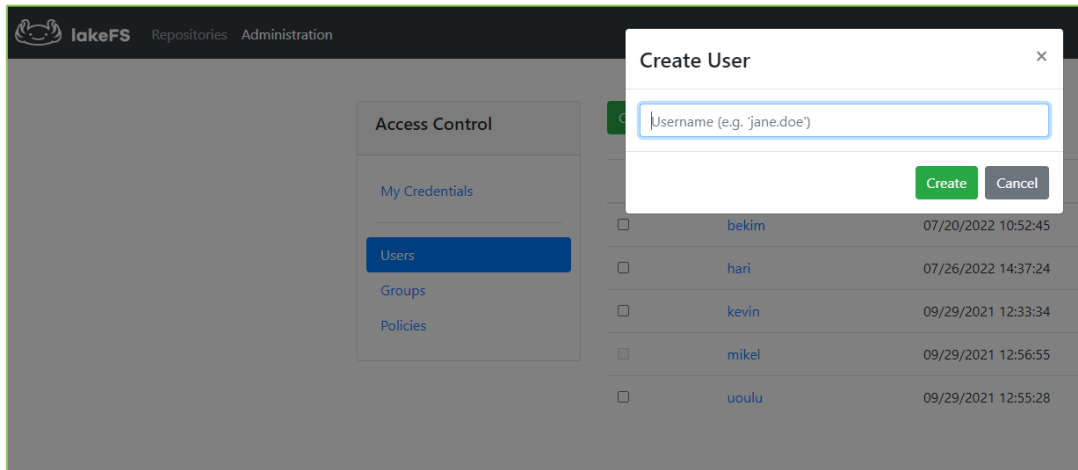


Figure 52: Creating users in lakeFS

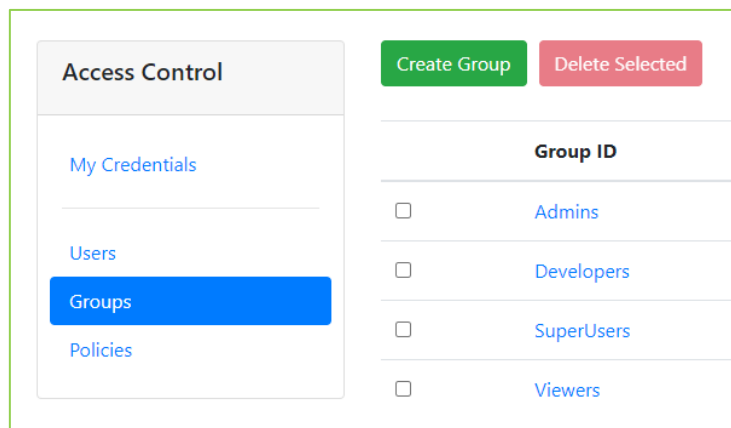


Figure 53: User group administration in lakeFS

6.4.1.2 Creating a repository & branches

Repositories and branches are used inside lakeFS to organize and structure the different data sources, use cases, and stages of the data. lakeFS provides those features on a GIT like way and they can be created using the UI.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

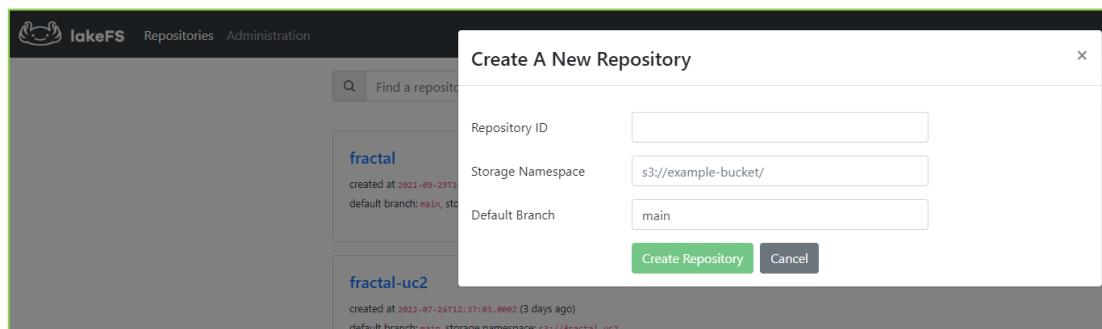


Figure 54: Creating repository in lakeFS

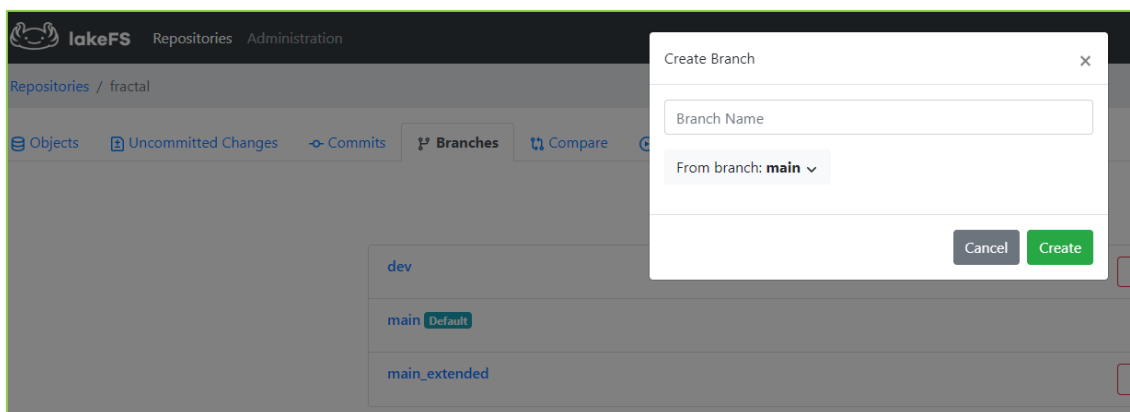


Figure 55: Creating a branch in lakeFS

6.4.1.3 Uploading & committing

In the same way, lakeFS offers the versioning of datasets. To do so, datasets can be uploaded and committed to specific branches.

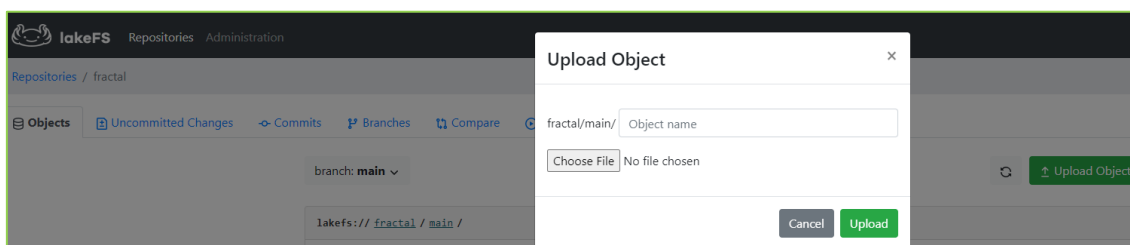


Figure 56: Uploading objects in lakeFS

6.4.1.4 Other interactions

lakeFS also supports integrations with other programming languages and platforms. There are many different possibilities, the main interactions guidelines being as follows.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

For example, if lakeFS is used on an Airflow DAG, the Airflow integration could be used to perform the required tasks.

<https://docs.lakefs.io/integrations/airflow.html>

LakeFS could also be used by Kubeflow, so in this case, the Kubeflow integration could be used.

<https://docs.lakefs.io/integrations/kubeflow.html>

In a more generic use case, the python integration could be used. For example, the edge node could be using the FRACTAL Cloud Platform lakeFS service using the python integration.

<https://docs.lakefs.io/integrations/python.html>

6.4.2 Feast

There are 5 main steps to set up a Feast repository, including:

1. Create feast repo
2. Register for feature definition
3. Generate features data
4. Load features to feast online store
5. Retrieve online-stored features and use

The first step is creating a Feast directory. It can be achieved with the following instruction:

```
$ feast init -m repo_name
```

The repository's name is the place where Feast will store features. This command will also auto-generate the YAML file inside the repo_name folder. It would look like this

```
# feature_store.yaml
project: repo_name
registry: data/registry.db
provider: local
online_store:
path: data/online_store.db
```

where:

- *project* allows to isolate feast repositories using the same infrastructure.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

- *registry* is the path of the registry file where Feast will store the feature definitions
- *provider* is the target environment where features are stored. Providers allow for the use of custom logic tailored to a specific infrastructure. Feast has 3 basic providers: AWS (Amazon Web Services), GCP (Google Cloud Platform), and local, which does not have any cloud-specific logic. A custom provider could be created to associate custom logic to feast actions.
- *online_store* is the path of the environment that Feast uses to store features for low-latency inference.

A feature repository using the infrastructure defined in Section 5.5.3 would look like this:

```
project: fractal-test
provider: local
registry: s3://feast-test/registry.db
online_store:
  type: redis
  connection_string: localhost:6379,password=REDIS_PASSWORD
offline_store:
  type: file
```

Figure 57: Feature repository definition

To follow the feast repo hierarchy of the first example above, the *data/* directory must be created, and move all raw data into it.

```
# go to the feast directory
$ cd repo_name
# create data folder
$ mkdir data/
# move all raw data into data folder
$ mv [raw_data_file_] data/
```

In the second step, a python file to define the features must be created, called here *def.py*.

```
$ nano def.py
```

Inside this document, Entity, FileSource, FeatureView, etc., are created. The documentation⁵⁴ shows a complete example of how to write this document and what fields the user should mention. One important parameter is the *ttl* which is related to the time that the Feature View will be cached for. Feast uses this to make sure that only new features are served to the model. Also, because of this, a

⁵⁴ <https://docs.feast.dev/getting-started/concepts>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

timestamp column needs to be created for the raw data, so feast can find out the proper features to serve.

Once the document is created, the user should run the apply command to register all the entities and feature views defined.

```
$ feast apply
```

Features can be saved in a dataset through this method:

```
dataset = store.create_saved_dataset(
    from_=data_retrieval,
    name="name_dataset",
    storage=SavedDatasetFileStorage("/path/to/save/name.parquet")
)
```

Last step to use Feast is to be able to load features data to online store. It can be achieved by running the same *materialize_incremental* command such as:

```
CURRENT_TIME=$(date -u +"%Y-%m-%dT%H:%M:%S")
feast materialize-incremental $CURRENT_TIME
```

Finally, after pushing features to Feast stores, features from both online or offline stores can be fetched and used for training/inferencing. For example, to get the data from the saved offline store, the following python code can be used:

```
training_df = store.get_saved_dataset(name="name_dataset").to_df()
```

To retrieve the low-latency data from online store, the function *get_online_features* should be called:

```
test_data = store.get_online_features(
    features=infer_features,
    entity_rows=[{"row_id": 568}]
).to_dict()
```

Full code that integrates Feast to run on Kubeflow is available on the following GitHub repository <https://github.com/Nannakaroliina/kubeflow-pipeline-demo/tree/kubeflow-feast>.

6.5 Model repository

One of the FRACTAL Cloud Platform's features is the AI model repository, where AI models are stored to make them available for the rest of the services. There are several options to choose from as a model repository, DVC, MLflow, and MLBuffet,

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

however, some of them cover the FRACTAL Cloud Platform’s requirements better than others. MLBuffet is a lightweight distributed AI model server which is highly edge-oriented, and although it could be used for model storage in the cloud, it could result in a worse performance than other model repositories specifically developed for Cloud instances. Lastly, DVC stands as a good option for model storing and version controlling. Complete use steps are given for DVC, MLBuffet, and MLflow in the following subsections:

6.5.1 DVC

Once DVC is started on the directory or parent directory, every file or directory of the parent directory can be tracked by DVC with Git-like syntaxis:

Adding files or directories to a repository

To add a file or folder to the repository:

```
$ dvc add (-<flags>) target (target2 ...)
```

This command generates a file (or a pair of files, if being used in a Git repository) in the tracked directory.

- .gitignore file. This file disables Git tracking from the file being now managed by DVC. This file is generated only if DVC is being used as file-tracking system inside a Git repository.
- <filename/directory>.dvc file. This file has all the information that DVC needs to track the file on the DVC repository (both on the cloud and locally).

While adding directories to the repository, some files can be ignored adding the path to the .dvcignore file (like Git ignored files) placed on the directory where DVC was initialized.

Upload (push) files/directories to the repository

After adding the files/directories to be tracked by DVC, those must be pushed to the repository:

```
$ dvc push (-<flags> <options>)
```

This command makes a copy of the target files/directories added to DVC on the file repository with codified names using the hash numbers written on the .dvc files.

Download (pull) files/directories from the repository

Once the target files/directories are pushed to the repository, those are available to be pulled as many times as required. Having the corresponding .dvc files:

```
$ dvc pull (-<flags> <options>)
```

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

All the tracked files will be downloaded to the folder, or the directory with its own directory tree and files.

More information on how to use DVC can be found on the official documentation (<https://dvc.org/doc>)

Set local (or remote) storage

A custom and generic cloud can be linked to a DVC repository to be used as a data registry (<https://dvc.org/doc/command-reference/remote>):

```
$ dvc remote add -d myremote /path/to/remote
```

The project's config file can also be modified locally:

```
['remote "myremote"']
url = /path/to/remote
[core]
remote = myremote
```

6.5.2 MLBuffet

MLBuffet provides a set of instructions to interact with its main API, and it can be used to store models locally or on the cloud. These models could be uploaded and downloaded from the edge to the cloud (and vice-versa), and managed in multiple ways, updating the model versions, deleting no longer needed models, uploading new models and providing information about each of their versions.

A summary of all the model-handling related actions and example curl HTTP requests for each of them is given below. A more detailed description and updated information can be found on the GitHub's README (<https://github.com/zyklab/mlbuffet/#readme>):

Note: The URIs to access the services should be substituted with the Ingress service or the endpoint for each of the deployments, either on Kubernetes, Docker Swarm or Docker standalone deployments.

Test the API:

```
curl inferrer:8000/
```

Access help:

```
curl inferrer:8000/help
```

Model handling:

- Display the full list of available models:

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
curl -X GET inferrer:8000/api/v1/models
```

- Display a model provided information:

```
curl -X GET inferrer:8000/api/v1/models/<tag>/information
```

- Update a model's information:

```
curl -X PUT -H "Content-Type: application/json" --data
'{"model_description": "<model_description>"}'
inferrer:8000/api/v1/models/<tag>/information
```

- Upload a new model:

```
curl -X POST -F "path=@/path/to/local/model"
inferrer:8000/api/v1/models/<tag>
```

- Download a stored model locally:

```
wget inferrer:8000/api/v1/models/<tag>/download --content-disposition
```

- Delete a stored model:

```
curl -X DELETE inferrer:8000/api/v1/models/<tag>
```

- Set a model as default:

```
curl -X POST -H "Content-Type: application/json" --data '{"default": <new
default version>}' inferrer:8000/api/v1/models/<tag>/default
```

6.5.3 Kubeflow and MLflow

Kubeflow will be used with MLflow in order to train and store models. When a new model is sent to the cloud, Kubeflow trains it by using a preprocessed dataset. The new model is then evaluated and sent to the model repository powered by MLflow. MLflow will save the model storing along its version and several parameters that can be tracked and compared. Any model stored by MLflow can then be loaded using its name.

Once Kubeflow and MLflow services are up and running in the cluster, the APIs and user interface can be accessed by their associated IP addresses. The guidelines to check their readiness and usage are introduced below.

There are a couple of essential points for the usage regarding exposing the Kubeflow interface to be accessed from outside of the Kubernetes cluster and creating the user accounts.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

Kubeflow uses the NodePort type to expose its core service using the Istio ingress-gateway. In order to make the Kubeflow accessible from the outside, first, a mechanism is needed to assign an IP address to the LoadBalancer type in the Kubernetes; this mechanism should be already available by the cloud provider, or services such as MetalLB⁵⁵ can do the same.

Second, the host of HTTPS must be added to the Istio service; otherwise, some services, such as notebook, will not work properly. The instructions for this part are available in the following link <https://v0-7.kubeflow.org/docs/started/k8s/kfctl-existing-arrikto/#secure-with-https>. Although it is from one old version, it is the same procedure as the latest version of Kubeflow.

Another point worth mentioning here is that Kubeflow uses the Dex authentication service to manage the users. By default, it has the user `user@example.com` with password `12341234`. It is required to remove any default users and create other users to eliminate the security risks⁵⁶.

Check the status of all the pods in kubeflow namespace are ready

```
$ kubectl get pod -n kubeflow
```

Get the IP address and access the UI

```
$ export KUBEFLOW_IP=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[].ip}')
$ echo $KUBEFLOW_IP
```

The default username and password to access the UI is the same as mentioned above, unless change the user authentication in Dex is changed.

Besides Kubeflow, MLflow will be used as another (or alternative) model repository. The configuration of the MLflow is pretty simple for the cloud operator for the deployment. All necessary configuration variables are available in the YAML files in the repository mentioned in the Section 5.6.3. It is worth noting that the only Tracking Server of MLflow is being used to store the model, and it is compatible with multiple storages such as S3, NFS (Network File System), and HDFS (Hadoop Distributed File System). The default value in configuration is to use a file in the container.

The configuration can be changed as desired in `deploy/kubernetes/mlflow-deployment.yaml`. There are environment variables to be set. As an example, `ARTIFACT_ROOT` refers to the `--default-artifact-root` which can be set to one of the mentioned storage classes that are available. More information about configuring the

⁵⁵ <https://metallb.universe.tf/installation/>

⁵⁶ <https://v0-7.kubeflow.org/docs/started/k8s/kfctl-existing-arrikto/#add-static-users-for-basic-auth>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

storage can be found at <https://mlflow.org/docs/latest/tracking.html#id14>. Below, some basic commands for MLflow are included, to make sure that it is functioning correctly. Since Kubeflow is exposed to the user and will interact with MLflow, the MLflow is only accessible within the Kubernetes cluster. Exposing the MLflow outside the cluster without an authentication mechanism such as Nginx or Dex will compromise the security.

Get the IP address and Port of the MLflow

```
$ export MLFLOW_IP=$(kubectl -n mlflow-k8s get service mlflow -o
jsonpath='{.spec.clusterIP}')
$ export MLFLOW_PORT=$(kubectl -n mlflow-k8s get service mlflow -o
jsonpath='{.spec.ports[.].port}')
```

Check that the service is working

```
$ curl -X GET http://$MLFLOW_IP:$MLFLOW_PORT
```

Note that the IP address and port number of MLflow should be given correctly to pipeline, so the pipeline stores the generated model in the repository.

More information about the Kubeflow and MLflow usage is included in Section 6.7.2

6.6 Harbor Private Registry

FRACTAL will use Kubernetes to deploy, manage and execute the different containerized services developed by the different partners of the FRACTAL Project. These services have been developed in the form of container images, along with different Helm charts specifying their deployment and configuration that are stored and managed by the Harbor private registry. During the deployment, Kubernetes access the registry to get the Docker images and Helm charts to configure and deploy the containerized services in different pods of the Kubernetes cluster. In the following sections, how the Harbor registry is used in the FRACTAL Cloud Platform is detailed.

6.6.1 Using Harbor in FRACTAL Cloud Platform

During the deployment of the different services that compose the FRACTAL Cloud Platform, Kubernetes access the registry to get the Docker images and Helm charts to configure and deploy the containerized services in different pods of the Kubernetes cluster (see Figure 58).

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

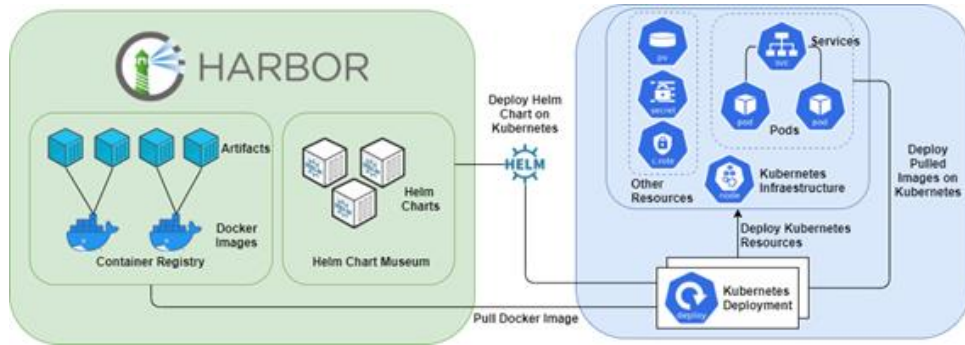


Figure 58: Interaction between Harbor (Image repository) and the Kubernetes cluster.⁵⁷

6.6.1.1 Accessing Harbor Container Registry from Kubernetes

For the deployment of a Docker Container in the Kubernetes platform, a YAML file must be created specifying the deployment for the container⁵⁸. In this file, the Docker image that will be used to deploy the container must be specified. As an example, in Figure 59, the YAML file (*inferred.yml*) used for the deployment of the FRACTAL inferer is shown. In the section of code shown in this figure, it can be seen the Docker image used for the deployment of the container. During the deployment, Kubernetes pulls this image from the Harbor container registry and creates a containerized service in a pod (the Docker image must be previously uploaded to the container registry, as shown in Subsection 6.6.2.1).

For accessing the private registry, Kubernetes also uses a secret⁵⁹ (see the secret named *fractalregistry* in the *imagePullSecret* section of code in Figure 59) with the access credentials for the Harbor container registry (see Section 5.7.3 for how to configure and create a secret in Kubernetes).

⁵⁷<https://blog.ovhcloud.com/managing-harbor-at-cloud-scale-the-story-behind-harbor-kubernetes-operator/>

⁵⁸ <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

⁵⁹ <https://kubernetes.io/es/docs/concepts/configuration/secret/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```

36 spec:
37   containers:
38     - name: fractal-inferer
39       image: vtwkpdos.gra7.container-registry.ovh.net/fractal/deploy_inferer:latest
40     ports:
41     - containerPort: 8000
42     env:
43     ...
44   imagePullSecrets:
45   - name: fractalregistry

```

Figure 59: YAML File for the Deployment of FRACTAL Inferer Container.

6.6.1.2 Accessing Harbor Helm Chart Museum from Kubernetes

For the deployment of a service in the Kubernetes platform using a Helm Chart, Kubernetes accesses the Helm Chart Museum of the Harbor private registry. Helm uses the \$KUBECONFIG environment variable⁶⁰ used to specify the Kubernetes configuration file to deploy charts from the available repositories in Kubernetes. However, for accessing the Chart Museum of the Harbor Private Registry, it must be added as a Helm repository (see Section 6.6.3.1 for details of how to add a repository to Helm).

6.6.2 Using the Image Repository

To use the image repository, first users must authenticate by using the *docker login* command⁶¹. After introducing their access credentials, the user is authenticated and will be able to work with the image repository. See the following subsections for pushing and pulling images to the repository.

```
docker login https://vtwkpdos.gra7.container-registry.ovh.net/
```

6.6.2.1 Pushing a Docker Image to the Container Registry

In order to *push* an image to a repository, first, the image must be tagged with the desired name and tag. In the project control panel (see Figure 33) in *Push Command*, a command reference can be found for tagging, and pushing docker images:

```
docker tag SOURCE_IMAGE[:TAG] vtwkpdos.gra7.container-
registry.ovh.net/fractal/REPOSITORY[:TAG]
docker push vtwkpdos.gra7.container-registry.ovh.net/fractal/REPOSITORY[:TAG]
```

Following the command reference, an image can be tagged and pushed to the repository using the following commands:

⁶⁰ <https://helm.sh/docs/helm/helm/>

⁶¹ Docker must be installed: <https://docs.docker.com/desktop/windows/install/>.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
docker tag vitis-ai-cpu:latest vtwkpdos.gra7.container-registry.ovh.net/fractal/xilinx/vitis-ai-cpu:demo
docker push vtwkpdos.gra7.container-registry.ovh.net/fractal/xilinx/vitis-ai-cpu:demo
```

Figure 60 shows the pushed docker image into the Xilinx repository.

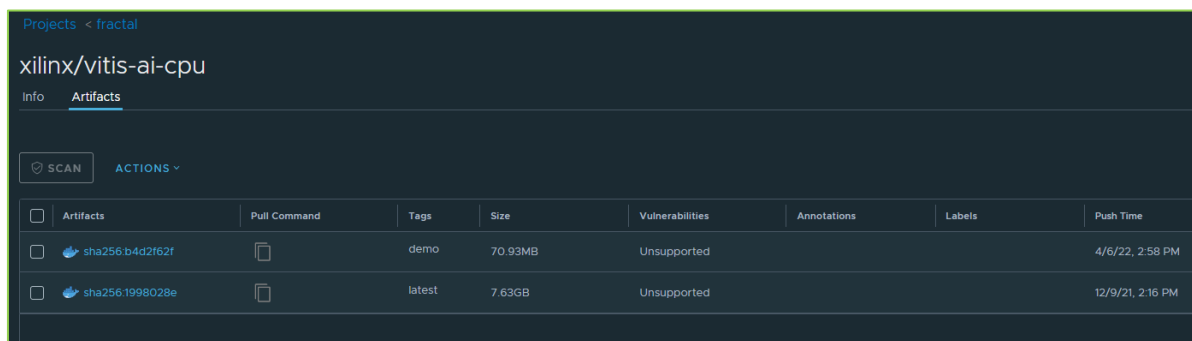


Figure 60: Pushed Docker Image to Xilinx Repository in FRACTAL Project

6.6.2.2 Pulling a Docker Image to the Container Registry

The images from the Harbor repositories can be pulled using the pull command available along with the artifact in the Harbor repository (see *Pull Command* section in Figure 60). This will copy to the clipboard the docker command for pulling the image. Next, the command for pulling the image pushed in the previous subsection is shown:

```
docker pull vtwkpdos.gra7.container-registry.ovh.net/fractal/xilinx/vitis-ai-cpu@sha256:b4d2f62f6411f88e3f6f5ce917332e7dc5569e5e3b4dd145d1baba8e4f3b5219
```

6.6.3 Using the Helm Chart Museum

For using the Helm Chart Museum⁶² to store and access Helm charts, first, a repository must be added in Helm⁶³. Helm will keep a list of the repositories where the different charts are stored for their usage on deployments. Next, the steps to add a repository to Helm and how to push and pull charts from it will be detailed.

6.6.3.1 Adding the FRACTAL Repository in Helm

For using the Harbor Helm Chart Museum with Helm, first, the repository must be added to the repository list⁶⁴. With the following command, the repository for the FRACTAL Project can be added.

⁶² <https://docs.ovh.com/sg/en/private-registry/using-helm-chart-museum/#instructions>

⁶³ Helm must be installed: <https://helm.sh/docs/intro/install/>

⁶⁴ <https://goharbor.io/docs/1.10/working-with-projects/working-with-images/managing-helm-charts/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
helm repo add --username=<USERNAME> --password=<PASSWORD> fractal
https://vtwkpdos.gra7.container-registry.ovh.net/chartrepo/fractal
```

To check the available repositories in helm the following command can be used:

```
helm repo ls
```

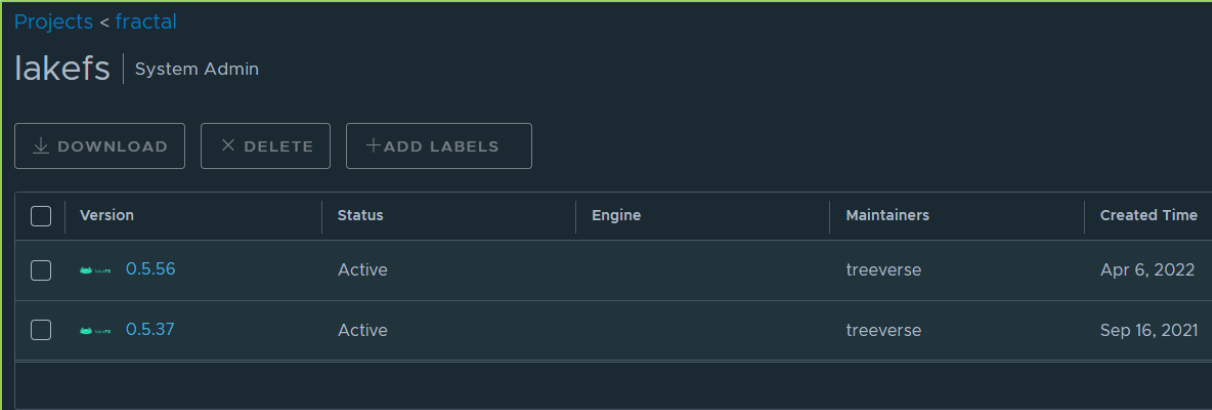
6.6.3.2 Pushing a Chart to the Helm Repository

Once the repository has been added to the Helm repository list, a chart can be pushed using the commands from the command reference available in the project control panel (see Figure 33).

In FRACTAL Project, different Helm Charts from most common repositories (e.g., Artifact Hub⁶⁵) will be used. Normally, these charts have been widely used and tested by the community and they offer a faster and simpler approach to deploying services, rather than creating custom charts⁶⁶. However, with the aim of keeping a copy of the used charts, these charts will be downloaded and pushed to the Helm Chart Museum in Harbor. Next, an example of how to pull the official Helm chart⁶⁷ for lakeFS and push⁶⁸ it to the Helm Chart Museum is shown.

```
helm pull lakefs/lakefs
helm plugin install https://github.com/chartmuseum/helm-push
helm cm-push -u=<USERNAME> -p=<PASSWORD> lakefs-0.5.56.tgz fractal
```

In Figure 61 the uploaded charts to the lakeFS repository can be seen:





<input type="checkbox"/>	Version	Status	Engine	Maintainers	Created Time
<input type="checkbox"/>	 0.5.56	Active		treeverse	Apr 6, 2022
<input type="checkbox"/>	 0.5.37	Active		treeverse	Sep 16, 2021

Figure 61: Helm Chart Museum.

⁶⁵ <https://artifacthub.io/>

⁶⁶ https://helm.sh/docs/helm/helm_create/

⁶⁷ <https://github.com/treeverse/charts/tree/master/charts/lakefs>

⁶⁸ For pushing the Helm chart, *chart museum push* plugin has been used: <https://github.com/chartmuseum/helm-push>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

6.6.3.3 Pulling a Chart to the Helm Repository

For pulling a chart from a Helm Repository, the pull command will be used:

```
helm pull lakefs/lakefs
```

However, in FRACTAL usually, helm charts will be directly accessed from the repository and installed⁶⁹:

```
helm install lakefs fractal/lakefs --version 0.5.56
```

6.7 ML orchestration

6.7.1 MLBuffet

MLBuffet acts mainly as a distributed ML Model server, where models are deployed and they can perform inference in an asynchronous manner. This is done through a modular architecture where the models are stored and distributed over a subnet of modelhost containers which hold the models and deploy them, so the APIs can communicate with each other, and the application runtime is not blocked by late inferences.

In addition, it can be provided with a train script, a dataset and a requirements file to perform training either in the cloud or the edge, on theoretically any ML training library. Two training mechanisms are provided: The first one is through Kubernetes clustering and is the recommended way of training since MLBuffet is recommended to be deployed over K8S. By providing all the necessary files for the training, the Trainer pod will be created and the model will be taken by the trainer and sent to the Inferer for automatic storage and deployment. The second mechanisms are through Docker sockets and the daemon API, which could be the way to go in RISC-V or not supporting K8S platforms. Instructions on how to set up these can be found in the README document and below.

In practice, all the ML lifecycle steps can be performed on MLBuffet, from model design, training, deploying, and inference, and all the orchestration processes which come after the models have been created can also be addressed, like model optimization, model updating, re-training of the models, and model version control. Its lightweight implementation makes it a flexible tool, being deployable on any machine which can perform containerization, either resource limited or not.

Also, it can be easily integrated with other ML tools (e.g., Kubeflow), just by sending an HTTP request to the main API, other tools can perform any of the actions in

⁶⁹ https://helm.sh/docs/helm/helm_install/

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

MLBuffet (model hosting, training, and inference) and include the results in their workflows and pipelines, by parsing the JSON response MLBuffet provides back.

Note: The URIs to access the services should be substituted with the Ingress service or the endpoint for each of the deployments, either on Kubernetes, Docker Swarm or Docker standalone deployments.

Model predictions:

- JSON array input:

```
curl -X POST -H "Content-Type: application/json" --data '{"values":[2, 5, 1, 4]}' inferrer:8000/api/v1/models/<tag>/prediction
```

- Images or files as input:

```
curl -X GET -F "file=@image.jpeg"
inferrer:8000/api/v1/models/<tag>/prediction | jq
```

Train a model:

```
curl -X POST inferrer:8000/api/v1/train/<tag>/<model_name> -F
"dataset=@/path/to/dataset.csv" -F "script=@/path/to/train.py" -F
"requirements=@/path/to/requirements.txt"
```

For model training, some considerations must be taken into account. This feature requires the Docker daemon host to be exposed securely, this means, providing MLBuffet with an accessible way to deploy Docker containers on the training machine. The TLS (Transport Layer Security) implementation is a good practice when a Docker daemon is exposed in a host's port because exposing the Docker daemon insecurely can lead to root access from attacking external users. However, the implementation details of this kind of communication are out of the scope of this document.

A detailed guide on how to expose the Docker daemon securely can be found on: <https://docs.docker.com/config/daemon/>

Another guide on implementing TLS secure communications for external applications with the exposed Docker daemon can be found in this link: <https://docs.docker.com/engine/security/protect-access/>

Once the Docker daemon is exposed and secured, the client certificates must be provided to the Inferrer container in the `mlbuffet/inferrer/flask_app/utils/client` directory, which will be used by a Python Client to schedule training containers on the host machine.

For Kubernetes deployments, training can also be done through Docker by accessing external daemons, however, the Kubernetes MLBuffet Trainer is recommended,

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

which takes care of managing certificates automatically and is more user-transparent.

6.7.2 Kubeflow and MLflow

MLflow can be used in Kubeflow to store models. To do this, both tools need to run in docker components to enable communication between each other. In this section, a simple pipeline composed of 3 steps will be used: the data preprocessing, the model training, and the model validating. Each step of the pipeline is a python file that runs the required code to achieve its goal.

MLflow's API provides a means to use MLflow in the pipeline components of Kubeflow. The main methods to do this are: *set_tracking_uri()* which allows to connect to MLflow from Kubeflow, *log_model()*, which registers a model in MLflow, and *get_registered_model()*, which loads a stored model. In the following example, our code allows us to store a model previously trained by Kubeflow, load it using its name, modify its description and update it in the model repository. This code can be written in the validating step of Kubeflow's pipeline. By doing this, the performance (e.g., accuracy) of the model can be stored in MLflow along the parameters.

```

mlflow.set_tracking_uri("http://172.17.0.2:5000")
with mlflow.start_run() as run:
    mlflow.sklearn.log_model(model, "sk_learn",
                             serialization_format="cloudpickle",
                             registered_model_name=model_name)

client = MlflowClient()
model = client.get_registered_model(model_name)

# It shows some parameters of the model
print_model_info(model)
desc = "This sentiment analysis model classifies tweets' tone: happy, sad, angry."
client.update_registered_model(model_name, desc)
model = client.get_registered_model(model_name)
# It shows the same model with an updated description
print_model_info(model)

```

Figure 62: Use of MLflow in the pipeline components of Kubeflow

Now it has been seen how to integrate MLflow in Kubeflow, the rest of this section will present how to run both frameworks. Firstly, the following instructions allow to build the images of each step that composes a pipeline:

```

$ docker build ./preprocess_data --tag
kubeflow_pipeline_mlflow_preprocessing:latest

```


	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```

$ docker push kubeflow_pipeline_mlflow_preprocessing:latest
$ docker build ./train_model --tag kubeflow_pipeline_mlflow_train:latest
$ docker push kubeflow_pipeline_mlflow_train:latest
$ docker build ./predict --tag milowb/kubeflow_pipeline_mlflow_predict:latest
$ docker push kubeflow_pipeline_mlflow_predict:latest

```

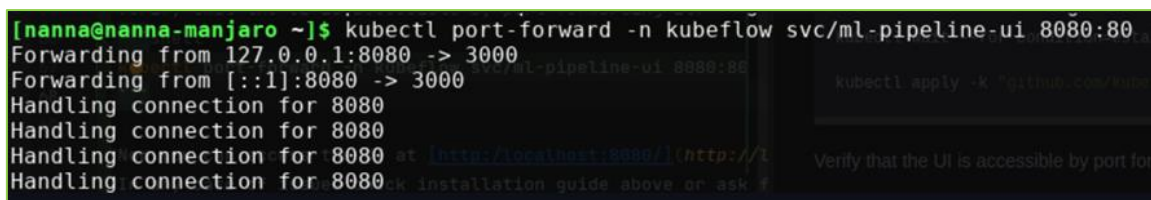
Once Kubeflow images are built, Kubeflow can be launched:

```

$ export PIPELINE_VERSION=1.8.1
$ kubectl apply -k
"github.com/kubeflow/pipelines/manifests/kustomize/cluster-scoped-
resources?ref=$PIPELINE_VERSION"
$ kubectl wait --for condition=established --timeout=60s
crd/applications.app.k8s.io
$ kubectl apply -k
"github.com/kubeflow/pipelines/manifests/kustomize/env/platform-agnostic-
pns?ref=$PIPELINE_VERSION"
$ kubectl port-forward -n kubeflow svc/ml-pipeline-ui 8080:80
MLFlow installation instructions are detailed in the section X.X. Then it can
be run with docker with the following instruction:
$ docker run -it --rm -p 5000:5000 -v /local/path:/mlflow --name mlflow-
server atcommons/mlflow-server

```

The expected result, after running Kubeflow, should be the following:



```

[nanna@nanna-manjaro ~]$ kubectl port-forward -n kubeflow svc/ml-pipeline-ui 8080:80
Forwarding from 127.0.0.1:8080 -> 3000
Forwarding from [::]:8080 -> 3000
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080 at http://localhost:8080/ (http://
Handling connection for 8080 k Installation guide above or ask ?

```

Figure 63: Result after running Kubeflow

Now UI can be accessed at <http://localhost:8080> to run experiments or use Kubeflow's SDK.

There are two Kubeflow concepts of interest in this section:

1. A **run**: it is a single execution of a pipeline. Runs comprise an immutable log of all experiments that the user attempts, and are designed to be self-contained to allow for reproducibility.
2. A **pipeline**: it is a workspace where the user can run several *runs* of one pipeline.

There are two ways of making new pipelines and runs:

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

1. Through the UI. The button Upload pipeline allows to add one.

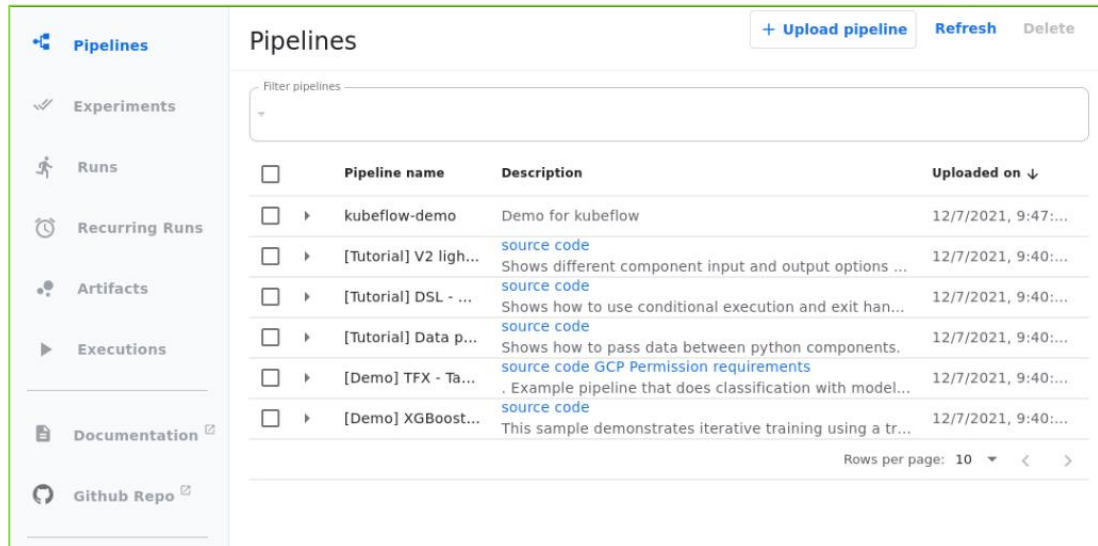


Figure 64: Kubeflow pipelines UI.

Adding a pipeline requires importing the *pipeline.yaml*. This file is a configuration file describing how the pipeline is built and where to find images of each step.

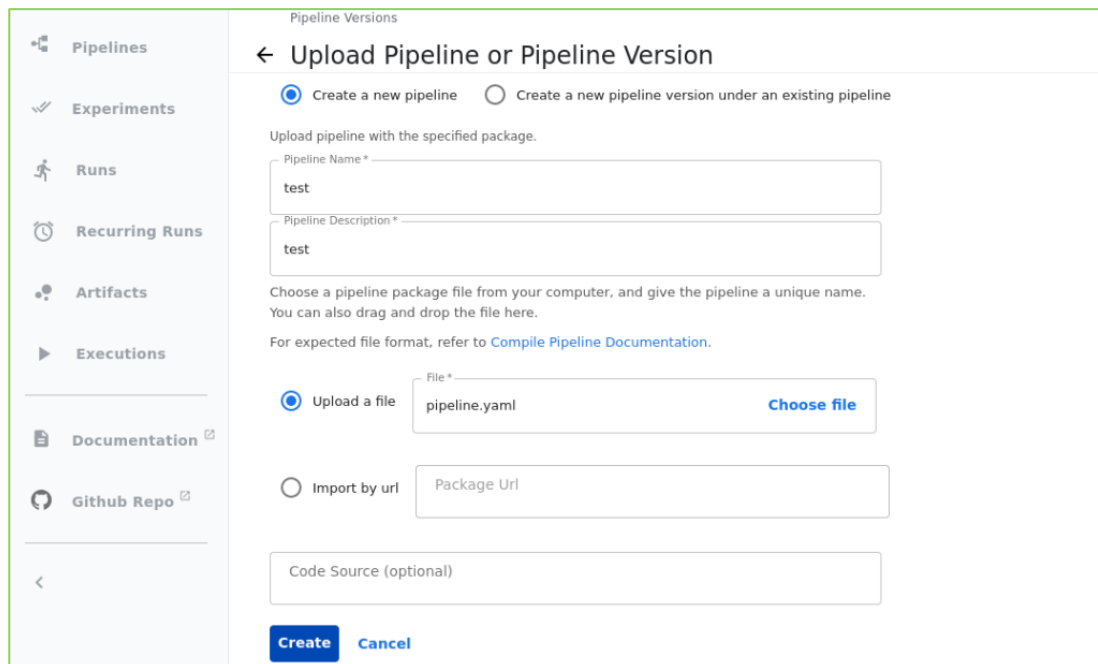


Figure 65: Adding a pipeline in Kubeflow.

Once the pipeline is created, the UI asks to the user to start a run.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

2. A second method is to use Kubeflow's SDK in Python with the following line of Python:

```
import kfp
client = kfp.Client(host='http://localhost:8080')
client.create_run_from_pipeline_func(
    my_pipeline,
    arguments={
        'url': 'https://storage.googleapis.com/ml-pipeline-playground
/iris-csv-files.tar.gz'
    })
```

This code allows the user to connect to Kubeflow and create a run for a pipeline (here called `my_pipeline`) without having to use the UI. It is possible to send a parameter to the pipeline through the last arguments of the method.

6.8 Workflow management

6.8.1 Airflow

Workflows are defined in Airflow by DAGs (Directed Acyclic Graphs). Those DAGs are python files with a specific structure. An example of that structure is shown below:

```
#Step 1
from airflow import DAG
from datetime import datetime, timedelta
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.python_operator import PythonOperator

#step 2
default_args = {
    'owner' : 'airflow',
    'depends_onpast' : False,
    'start_date' : datetime(2021,11,4),
    'retries':0
}

#step 3
dag = DAG(dag_id='DAG-1', default_args=default_args, catchup=False,
schedule_interval='@once')

#step 4
start = DummyOperator(task_id='start', dag=dag)
end = DummyOperator(task_id='end', dag=dag)
```

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
#step 5
start >> end
```

The example shown above explains how to structure, create and operate different tasks among a DAG in Airflow. It is worth mentioning that every task that Airflow has to execute is defined by an operator⁷⁰. There are three main types of operators:

- Action operators: they perform an action or tell another system to perform an action.
- Transfer operators: they move data from one system to another system.
- Sensor operators: they allow to check if a criterion is met to get completed.

The created DAGs can be managed in the DAG page of the Airflow service. In this page, an overview of each DAG execution can be seen, including the number of successful executions, the periodicity, the latest execution date, and more.

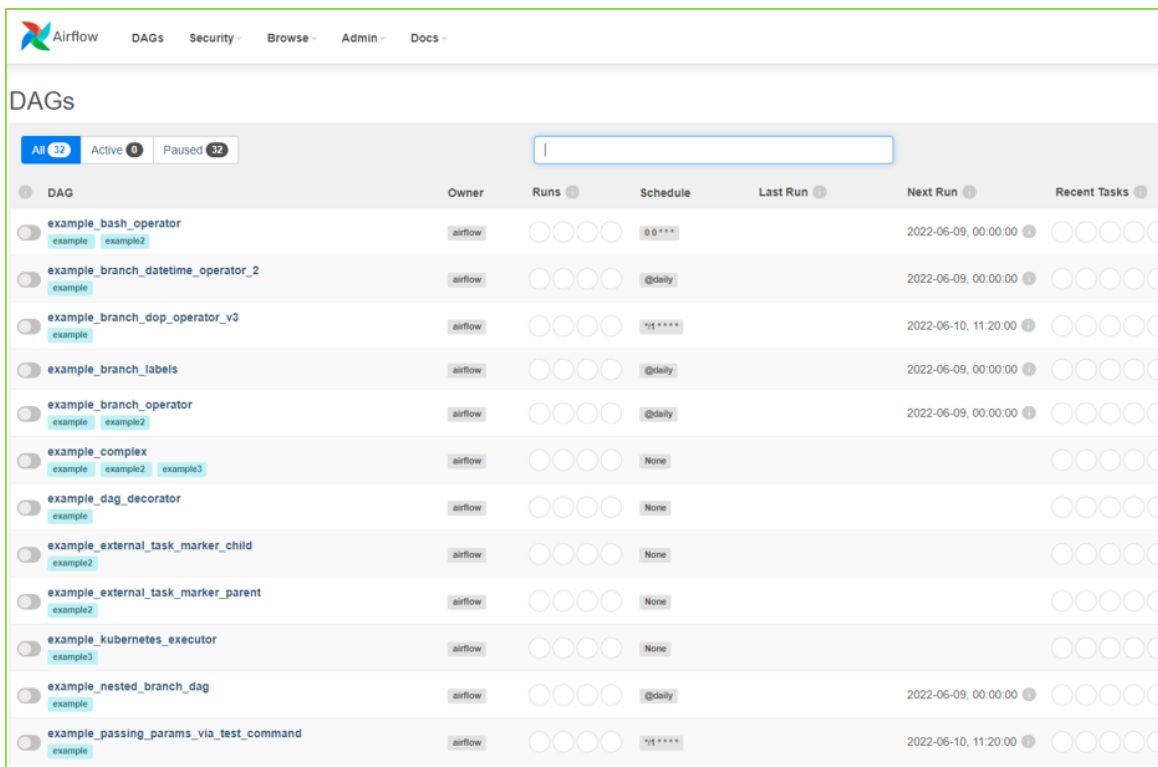


Figure 66: Airflow DAG list UI

⁷⁰ <https://airflow.apache.org/docs/apache-airflow/stable/python-api-ref.html#operators>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

6.9 Model preparation for FRACTAL Edge

6.9.1 Triggering a preparation run

There are multiple ways to drive a model preparation run for deployment.

With any new model being registered in the Model Repository, the next Airflow scheduled run can detect this and operate the preparation, the model is to be exposed to. Qualifiers added to each model guide the correct Airflow DAG to operate, as to which target to be built for, and the steps required to achieve the target.

If the orchestration does not schedule the Airflow operation regularly, a manual trigger can be issued by the uploading entity. This may be generated from a retraining step of a model that invalidates the model and explicitly triggers a preparation. These mechanics need to be covered on the orchestration layer.

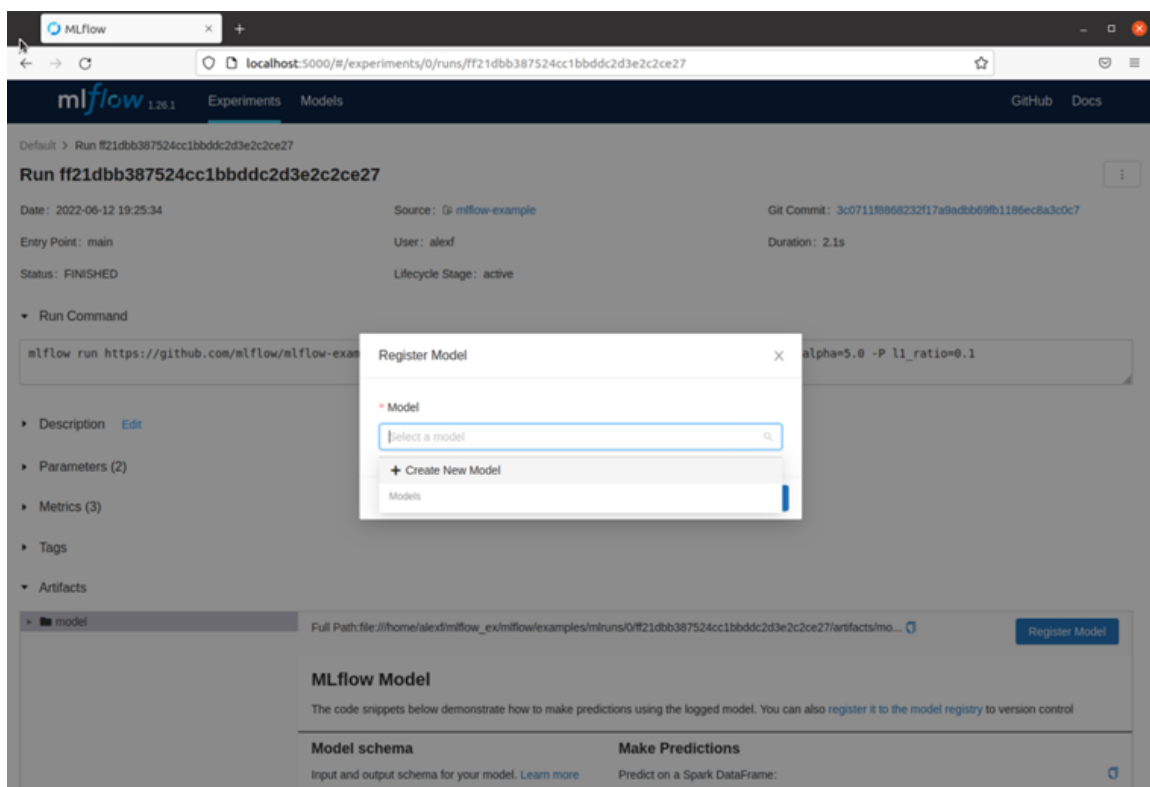


Figure 67: Model registration GUI in MLflow

6.9.2 Upload model into MLflow Repository within DAG

After the Airflow DAG tasks have generated the target implementation and other artifacts, the data structure of the model is augmented and the artifacts are uploaded into the MLflow model repository.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

6.10 Upload configuration scripts to FRACTAL Cloud Platform

As mentioned in Section 5.1 one possibility for partners to contribute to the FRACTAL Cloud Platform is directly providing the deployment and configuration YAML files or Helm Charts that will be used to deploy the different services that conform the FRACTAL Cloud Platform in Kubernetes. This section will show how the deployment of two different services of the FRACTAL Cloud Platform in the OVH Managed Kubernetes platform could be accomplished (one using a YAML file and another one using a Helm Chart).

6.10.1 Deployment of a service in Kubernetes with YAML file

Kubernetes uses some entities known as *Kubernetes objects*⁷¹ to represent the state of the cluster and the different services and applications that run on it. Usually, these objects are used to describe:

- What containerized applications are running (and on which nodes).
- The resources available to those applications.
- The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance.

For creating objects in a Kubernetes system, the Kubernetes API is used (either directly or via kubectl). When creating an object in Kubernetes, the object specification that describes its desired state, as well as some basic information about the object (such as a name) must be provided. This object specification must be included as JSON data in the API request body. However, most often, the information is provided to kubectl tool in a .yaml file that then, kubectl converts to JSON for the API request. The next subsection shows how to deploy an ingress service using a .yaml file.

6.10.1.1 Deploying an Ingress Controller in Kubernetes with a YAML File

Ingress is a specialized load balancer for Kubernetes. It accepts traffic from outside the Kubernetes platform, and load balances it to pods or containers running inside the platform (see Figure 68). Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource. For deploying an Ingress Controller in Kubernetes⁷² the .yaml file available at <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.44.0/deploy/static/provider/cloud/deploy.yaml> has been used. This service can be deployed to Kubernetes with the following command:

⁷¹ <https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>

⁷² <https://kubernetes.io/docs/concepts/services-networking/ingress/>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.44.0/deploy/static/provider/cloud/deploy.yaml
```

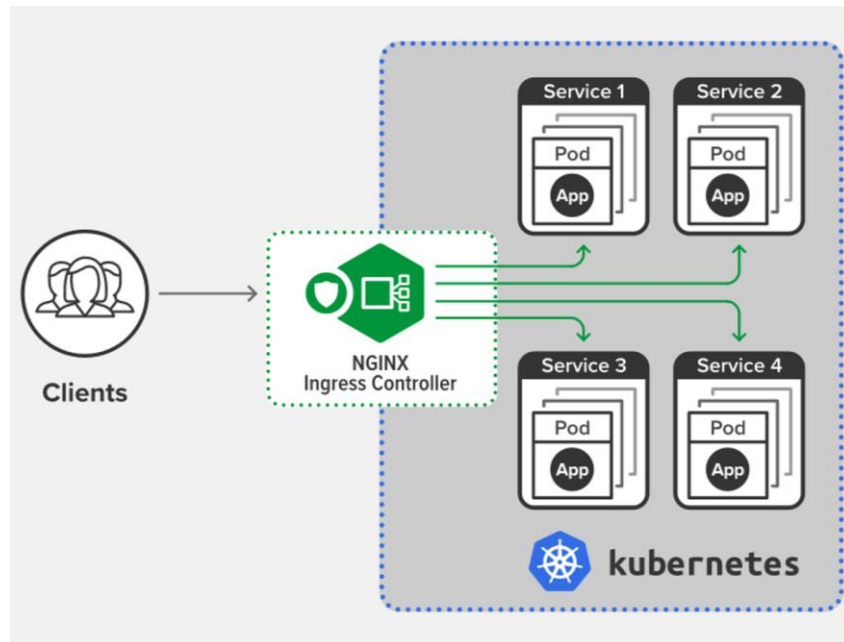


Figure 68: Ingress Load Balancer for Kubernetes.

6.10.2 Deployment of a service in Kubernetes using a Helm Chart

Helm charts are a set of manifests that allow to define the required Kubernetes resources and deployments along with their configuration. For the deployment of a service in the Kubernetes platform using a Helm Chart, Kubernetes accesses the Helm Chart Museum of the Harbor private registry (see Section 5.7). Helm uses the `$KUBECONFIG` environment variable used to specify the Kubernetes configuration file to deploy charts from the available repositories in Kubernetes. The next subsection shows how to deploy lakeFS dataset repository using a Helm chart (see also Section 5.7 for working with Helm charts in Harbor Helm chart museum).

6.10.2.1 Deploying lakeFS Dataset Repository with a Helm Chart

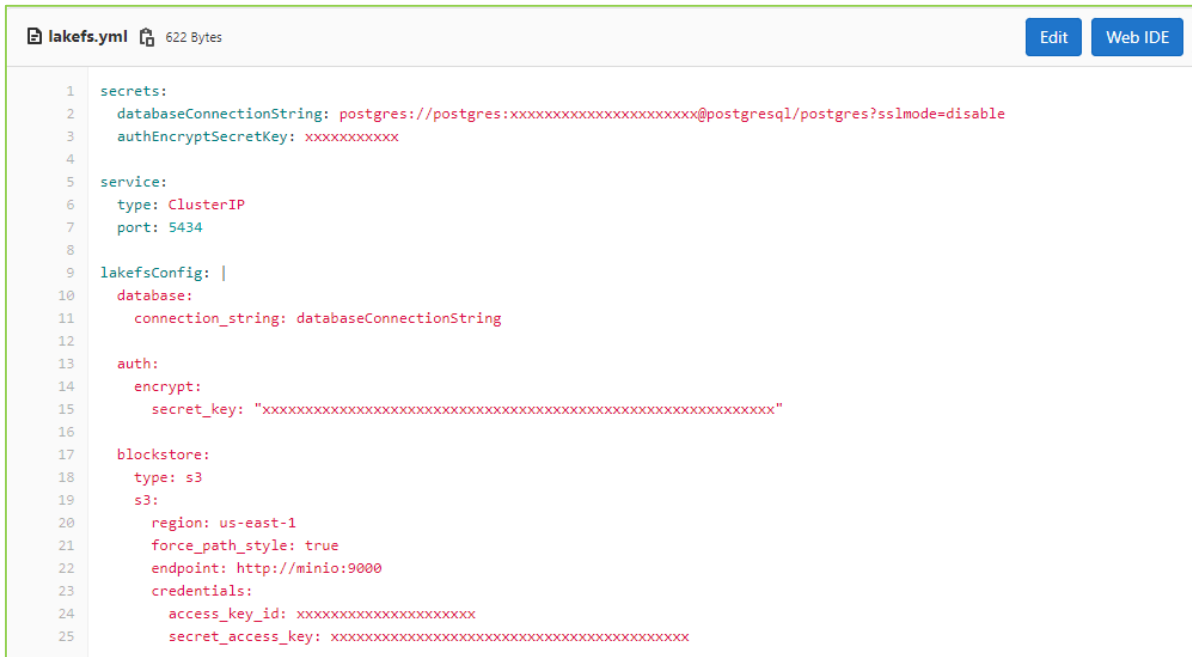
lakeFS can be installed on Kubernetes by using the Official lakeFS Helm Chart⁷³. Some parameters could be customized⁷⁴ by applying the lakefs.yml file (see Figure 69) when installing lakeFS using the Helm chart. To install lakeFS with the desired parameters customization, the following command can be used (please, note that this chart is being accessed from the FRACTAL repository in the Harbor Helm Chart museum as described in Section 5.7):

⁷³ <https://artifacthub.io/packages/helm/lakefs/lakefs>

⁷⁴ <https://docs.lakefs.io/reference/configuration.html#example-minio>

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

```
helm install -f lake.yml lakefs fractal/lakefs --version 0.5.56
```



```

1 secrets:
2   databaseConnectionString: postgres://postgres:xxxxxxxxxxxxxxxxxxxxxxxx@postgresql/postgres?sslmode=disable
3   authEncryptSecretKey: xxxxxxxxxxxx
4
5 service:
6   type: ClusterIP
7   port: 5434
8
9 lakefsConfig: |
10  database:
11    connection_string: databaseConnectionString
12
13  auth:
14    encrypt:
15      secret_key: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
16
17  blockstore:
18    type: s3
19    s3:
20      region: us-east-1
21      force_path_style: true
22      endpoint: http://minio:9000
23      credentials:
24        access_key_id: xxxxxxxxxxxxxxxxxxxx
25        secret_access_key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Figure 69: lakeFS Helm Charts parameters customization YAML file.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

7. Conclusions

This deliverable is part of the FRACTAL Task 5.2 developing a runtime platform to deploy, test and run the AI algorithms and it is a continuation of the previous deliverable D5.2 Intermediate Platform for Federated AI. In the previous deliverable, a list of requirements for the FRACTAL Cloud Platform were presented in addition to functional and technical descriptions of different technologies that could fit with those requirements.

In this deliverable, the list of the specific technologies selected for the FRACTAL Cloud Platform was presented, as well as a detailed procedure of the installation of each technology/component. On top of that, user guidelines were specified for each component on the FRACTAL Cloud Platform, which lead to detailed instructions for the Use Cases to use the components identified according to their needs.

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

8. List of figures

Figure 1. Learning Approaches in FRACTAL: (left) Centralized Learning; (centre) Decentralized Learning; (right) Federated Learning.....	9
Figure 2: Relationship between the cloud platform and the edge nodes.	12
Figure 3: FRACTAL Cloud Platform modules.	13
Figure 4: OVH Managed Kubernetes Service.	19
Figure 5: Data flow using MQTT broker with Connect	22
Figure 6: Schematic showing Kafka Connect bridging MQTT broker and Kafka broker	22
Figure 7: Communication from Edge to deployed Kafka service	22
Figure 8: Strimzi Operator and Kafka Architecture in K8s	23
Figure 9: Obtaining the kubeconfig file	24
Figure 10: Pods in the K8s cluster	25
Figure 11: User Access and Roles Dashboard	27
Figure 12: OpenStack RC file for accessing Horizon.....	27
Figure 13: Installing python openstack client and setup.....	28
Figure 14: Creating local credentials for object storage access	28
Figure 15: Editing credentials in awscli credentials file.....	29
Figure 16: Editing AWS config file to access OVH object storage	29
Figure 17: listing object storage containers in OVH cloud.....	29
Figure 18: Object storage container dashboard in OVH cloud	30
Figure 19: Creating Object container in the cloud	30
Figure 20: Steps to create an object container	31
Figure 21: Final status after object container creation	32
Figure 22: Object Container creation.....	33
Figure 23: Uploading objects to the container	34
Figure 24: Setting up a Data processing job.....	35
Figure 25: Selecting the container and python script in the data processing job....	36
Figure 26: Job processing dashboard with logs and monitoring.....	36
Figure 27. lakeFS Service and Deployment	38
Figure 28: MinIO Service and Deployment	41
Figure 29: MinIO Console	42
Figure 30: Deploying OVH Managed Private Registry through OVH Cloud Control Panel.	51
Figure 31: Created OVH Managed Private Registry	52
Figure 32: Creating a New Project in Harbor.	52
Figure 33: FRACTAL Harbor Private Registry.	53
Figure 34: Add a User to Harbor.	54
Figure 35: Creating a Robot Account for Kubernetes	55
Figure 36: Airflow DAG model preparation	65
Figure 37: Created fractal-kubernetes-cluster.	66

	Project	FRACTAL		
	Title	Platform and building blocks for Federated AI		
	Del. Code	D5.4		

Figure 38: Kubernetes Dashboard Service.68

Figure 39: service-account.yml file content.68

Figure 40: cluster-role-binding.yml69

Figure 41: ingress-kubernetes-dashboard.yml file contents.....70

Figure 42: Kubernetes dashboard.71

Figure 43: Kafka topic YAML72

Figure 44: Streaming messages to Topics.....73

Figure 45: Executing the simple_producer.py74

Figure 46: Using Kafka Consumer to extract messages75

Figure 47: Batch aggregates of raw drive cycles76

Figure 48: Timeseries data processing script.....77

Figure 49: Image processing snippet.....78

Figure 50: Sample environment file78

Figure 51: Setting up the spark job79

Figure 52: Creating users in lakeFS80

Figure 53: User group administration in lakeFS80

Figure 54: Creating repository in lakeFS81

Figure 55: Creating a branch in lakeFS.....81

Figure 56: Uploading objects in lakeFS81

Figure 57: Feature repository definition83

Figure 58: Interaction between Harbor (Image repository) and the Kubernetes cluster.....90

Figure 59: YAML File for the Deployment of FRACTAL Inferrer Container.....91

Figure 60: Pushed Docker Image to Xilinx Repository in FRACTAL Project92

Figure 61: Helm Chart Museum.93

Figure 62: Use of MLflow in the pipeline components of Kubeflow96

Figure 63: Result after running Kubeflow97

Figure 64: Kubeflow pipelines UI.98

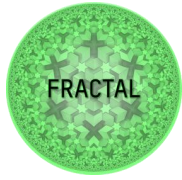
Figure 65: Adding a pipeline in Kubeflow.98

Figure 66: Airflow DAG list UI100

Figure 67: Model registration GUI in MLflow101

Figure 68: Ingress Load Balancer for Kubernetes.....103

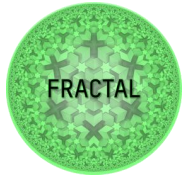
Figure 69: lakeFS Helm Charts parameters customization YAML file.104



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

9. List of tables

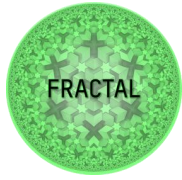
Table 1: Kafka platform access URLs.....25



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

10. List of abbreviations

ACID	Atomicity, Consistency, Isolation and Durability
AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
CLI	Command Line Interface
CSV	Comma Separated Values
DAG	Directed Acyclic Graph
DNS	Domain Name System
DVC	Data Version Control
ETL	Extract, Transform and Load
GCP	Google Cloud Platform
GB	GigaByte
GNU AGPL	GNU (GNU's Not Unix) Affero General Public License
HDFS	Hadoop Distributed File System
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IaaS	Infrastructure as a Service
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
ML	Machine Learning



Project	FRACTAL		
Title	Platform and building blocks for Federated AI		
Del. Code	D5.4		

MQTT	Message Queuing Telemetry Transport
NFS	Network File System
ONNX	Open Neural Network Exchange
PaaS	Platform as a Service
REST	Representational State Transfer
S3	Simple Storage Service
SaaS	Software as a Service
SDK	Software Development Kit
SSH	Secure SHell
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UI	User Interface
YAML	YAML Ain't Markup Language