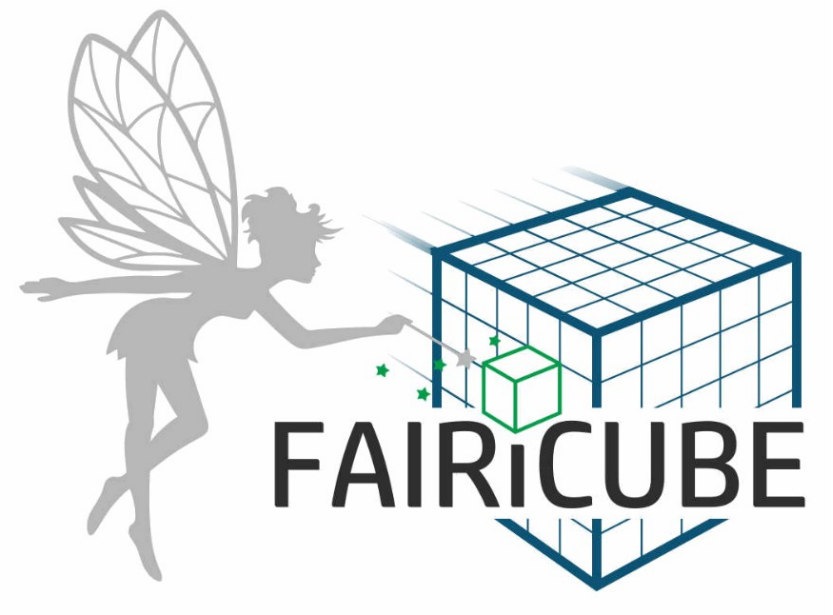

FAIRICUBE – F.A.I.R. INFORMATION CUBES

WP4 Share

D4.1 FAIRiCUBE Hub Architecture

Deliverable Lead: EOX

Deliverable due date: 28-02-2023

Version: 2.1

17-11-2023

Document Control Page

Document Control Page	
Title	D4.1 FAIRiCUBE Hub Architecture
Creator	EOX
Description	The FAIRiCUBE Hub Architecture details the functionalities available from the FAIRiCUBE Hub for users but also for providers of services, data, apps, notebooks, etc.
Publisher	"FAIRiCUBE – F.A.I.R. information cubes" Consortium
Contributors	All
Date of delivery	28-02-2023
Type	Text
Language	EN-GB
Rights	Copyright "FAIRiCUBE – F.A.I.R. information cubes"
Audience	✓ Public ☐ Confidential ☐ Classified
Status	☐ In Progress ☐ For Review ✓ For Approval ☐ Approved

Revision History			
Version	Date	Modified by	Comments
0.1	11-12-2022	Stephan Meißl, EOX	Initial draft
0.2	14-12-2022	Cristina Carnerero, 4sfera Jaume Targa, 4sfera	Review
0.3	12-02-2023	Kathi Schleidt, DataCove Peter Baumann, JacobsU Cristina Carnerero, 4sfera Jaume Targa, 4sfera Stefan Brand, EOX Stephan Meißl, EOX	Further input
0.4	26-02-2023	Kathi Schleidt, DataCove Mirko Gregor, space4environment Stephan Meißl, EOX	Restructuring and further input
1.0	27-02-2023	Kathi Schleidt, DataCove Peter Baumann, JacobsU Stephan Meißl, EOX	First complete release
1.1	07-03-2023	Jaume Targa, 4sfera Cristina Carnerero, 4sfera Eudard Lama, 4sfera	Final review
1.2	08-03-2023	Stephan Meißl, EOX	Final version for submission
1.3	16-06-2023	Christian Schiller, Stephan Meißl, EOX	changing: FAIRiCUBE Hub to FAIRiCUBE Lab FAIRiCUBE Platform to FAIRiCUBE Hub adding API and workflow descriptions adding FAIRiPATH – as new Fig .2
1.4	05-09-2023	Christian Schiller, Stephan	adding additional Details about

		Meißl, EOX	the FAIRiCUBE Lab setup of Apps, Control Plane and Worker Plane
1.5	02-10-2023	Antonio Cozzolino, Epsilon	adding Knowledge Base
2.0	13-10-2023	Jaume Targa, 4sfera	Final review
2.1	17-11-2023	Christian Schiller, Stephan Meißl, EOX	Updating Metadata ingestion procedure, replacing placeholder Figures, adding Community Collaboration Platform Chapter

Disclaimer

This document is issued within the frame and for the purpose of the FAIRICUBE project. This project has received funding from the Horizon Europe research and innovation programme under grant agreement No. 101059238. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Table of Contents

List of Figures

1 Introduction

This is the deliverable D4.1 "FAIRiCUBE Hub Architecture" of the FAIRiCUBE project, i.e., the FAIRiCUBE integrated datacube platform. It provides a detailed description of the FAIRiCUBE Hub architecture detailing its components, its deployment and operations strategy based on control and worker plane as well as on-boarding requirements and processes for service and application providers. The FAIRiCUBE Hub Architecture details the functionalities available from the FAIRiCUBE Hub for users but also for providers of services, data, apps, notebooks, etc.

D4.1 FAIRiCUBE Hub Architecture, this deliverable, is the output of the WP4 Share Task 4.1 "Define FAIRiCUBE Hub Architecture" prepared by EOX and JUB. As such, it relies on the further WPs WP5 "Ingest", WP4 "Share", and WP3 "Process" which each contribute a key facet to the overall FAIRiCUBE Hub.

The core objective of FAIRiCUBE is to enable users from beyond classic Earth Observation (EO) domains to provide, access, process, and share gridded data and algorithms in a FAIR and TRUSTable manner.

To reach the objective above, the FAIRiCUBE Hub is developed. This Hub is a cross-cutting platform and framework for data ingestion, provision, analysis, processing, and dissemination, to unleash the potential of environmental, biodiversity and climate data through dedicated European data spaces. Within the FAIRiCUBE project, TRL 7 will be attained for new components and the overall system, together with the necessary governance aspects to assure continued maintenance of the FAIRiCUBE Hub beyond the project lifespan (Figure 1). It will enable a broader range of stakeholders to focus on what they are supposed to do best: overcome technical barriers to make data-driven decisions and leverage state-of-the-art processing technologies, including Machine Learning (ML).

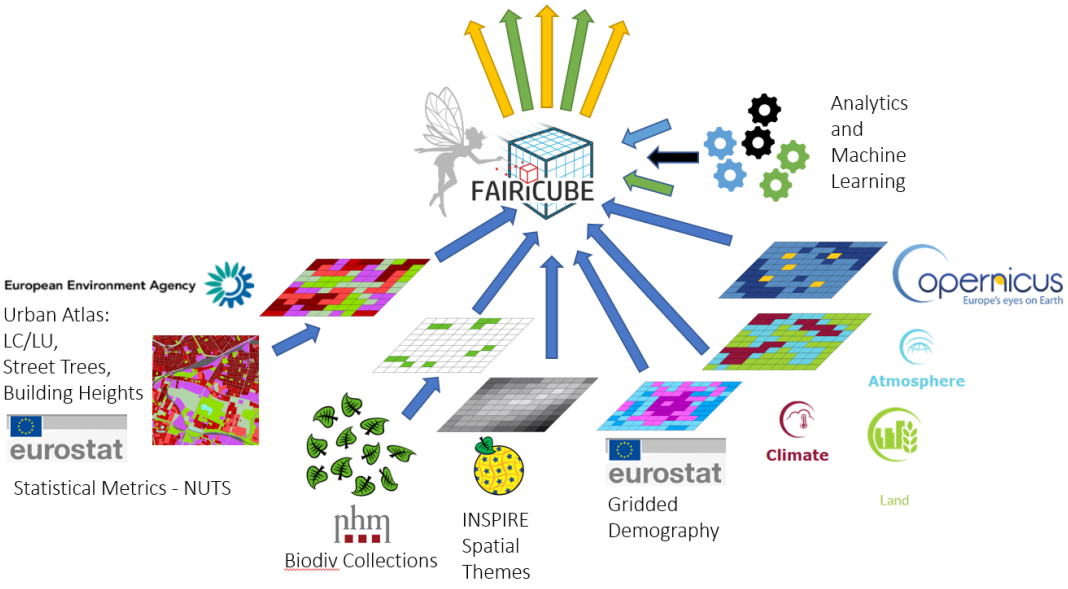

Figure 1: Schematic Project Overview

The FAIRiCUBE Hub is the central technology point of FAIRiCUBE. The Hub provides access to diverse data sources as well as processing and visualization tools. Users wishing to perform an analysis can bring their own data to FAIRiCUBE, whereby non-aligned sources (point, vector, but also gridded data not aligned to the European Grid) will be transformed as required before ingestion or during a request. Diverse analysis and ML tools are made available for users to tailor to their specific requirements and data sources. Finally, the FAIRiCUBE Dashboards will allow for interactive presentation of results.

In WP2, selected use cases illustrate how data-driven projects can benefit from cube formats, infrastructure, and computational benefits. They guide the project in creating a user-friendly FAIRiCUBE Hub providing relevant stakeholders an overview of both data and processing modules readily available to be applied to these data sources.

Tools enabling users not intimately familiar with the worlds of EO and ML to scope the requirements and costs of their desired analyses are implemented, easing uptake of these resources by a broader community. The FAIR sharing of results with the community is fostered by providing easy to use tools and workflows directly in the FAIRiCUBE Hub.

One of the expected results of the FAIRiCUBE Hub is that, in addition to a rich catalog of available gridded data resources, diverse processing modules utilizing diverse ML techniques (in some cases pre-trained for specific applications) are made available. The functionality and usability are trialled through the defined use cases to provide an advanced method/procedure to big data analysis with the help of ML, to improve decision making for a broad range of interested parties.

This document represents the first version, due at M8, i.e., by 28.02.2023, of the deliverable D4.1 "FAIRiCUBE Hub Architecture" of the FAIRiCUBE project. No further versions of D4.1 are due. This deliverable reports the activities carried out within the Task 4.1 "Define FAIRiCUBE Hub Architecture".

2 FAIRiCUBE Hub Overview

The FAIRiCUBE Hub is a fully managed cloud environment providing scalable compute and (co-located) storage resources for diverse stakeholders ranging from scientists to decision makers. The FAIRiCUBE Hub Architecture is based on EOxHub as well as rasdaman and other Open-Source components. The overall FAIRiCUBE Hub encompasses the FAIRiCUBE Catalog, FAIRiCUBE Services and Applications, as well as the FAIRiCUBE Lab (see Figures 18, 30, and 34). The user interaction with the FAIRiCUBE Hub is mainly by interactively using Jupyter Notebooks or other applications in the FAIRiCUBE Lab or directly by using the provided FAIRiCUBE Services and APIs.

Please find below the definitions of the components of the FAIRiCUBE Hub to avoid confusions and ambiguities.

FAIRiCUBE Hub: The overall FAIRiCUBE technical environment encompassing the FAIRiCUBE Catalog, FAIRiCUBE Services and Applications, as well as the FAIRiCUBE Lab.

FAIRiCUBE Catalog: The integrated catalog providing metadata and references to ingested datasets, processes, and models available from FAIRiCUBE.

FAIRiCUBE Services/Apps: Components providing various ways to access the data and processing facilities provided by FAIRiCUBE.

FAIRiCUBE Lab: A single container for all workspaces providing the interface to back-ends via various back-end protocols as well as an execution environment for user provided workloads.

FAIRiCUBE Workspace: The user area within the FAIRiCUBE Lab where users are able to collaborate and share the content of their workspaces.

FAIRiCUBE Knowledge Base: Provides a set of tools to enable appropriate knowledge of how to apply algorithms and ML techniques to solve similar demands.

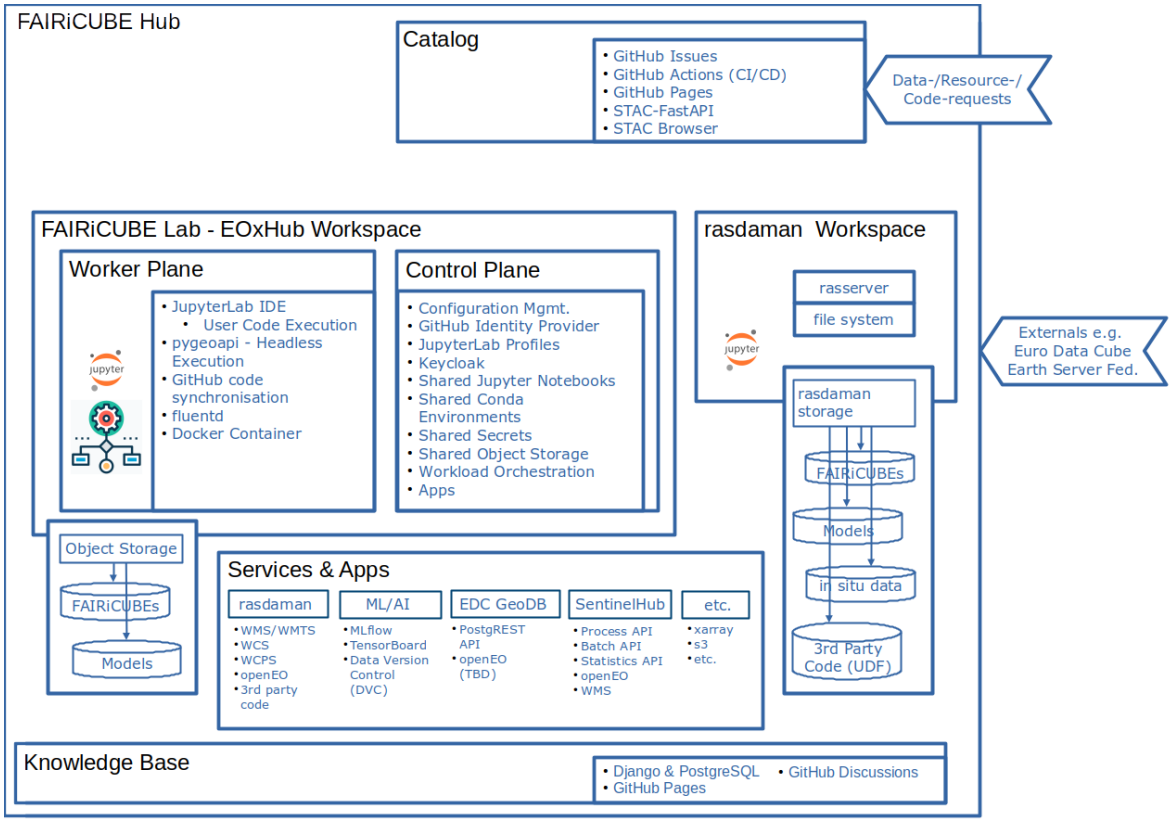

Figure 2: FAIRiCUBE Architecture Overview

The FAIRiCUBE Hub provides a complete working environment where users can access algorithms and data remotely to obtain computing resources and tools that they might not otherwise have and avoid the need to download and manage large volumes of data. This new approach removes the need to transfer/download large e.g., Earth Observation data sets around the world, while increasing the analytical power available research scientists, industry, operational service providers, regional authorities, and policy analysts.

FAIRiCUBE provides:

- Easy access to data and the tools to exploit these data.
- Use of online cloud computing resources which removes the need to download and store large volumes of data locally.
- Data, visualisation, and processing options that are tailored to the needs of science and operational users.
- Personalised and private accounts that can be accessed from any location through the Internet.
- A clear and intuitive user interface to access the platform functionality, including an interactive map portal for visualising data and outputs.
- Access to built-in processors or user-provided processors.
- Processor outputs that can be used in other processors, shared with other users, or download.
- Access to considerable processing capacity for analysis of large volumes of data.
- A customisable online development environment with all the necessary software tools and libraries to develop processors and optionally make them available to other users.
- A collaboration environment for groups to communicate via a platform forum, share software code using the platform code repository, and track problems with a built-in issue tracker.

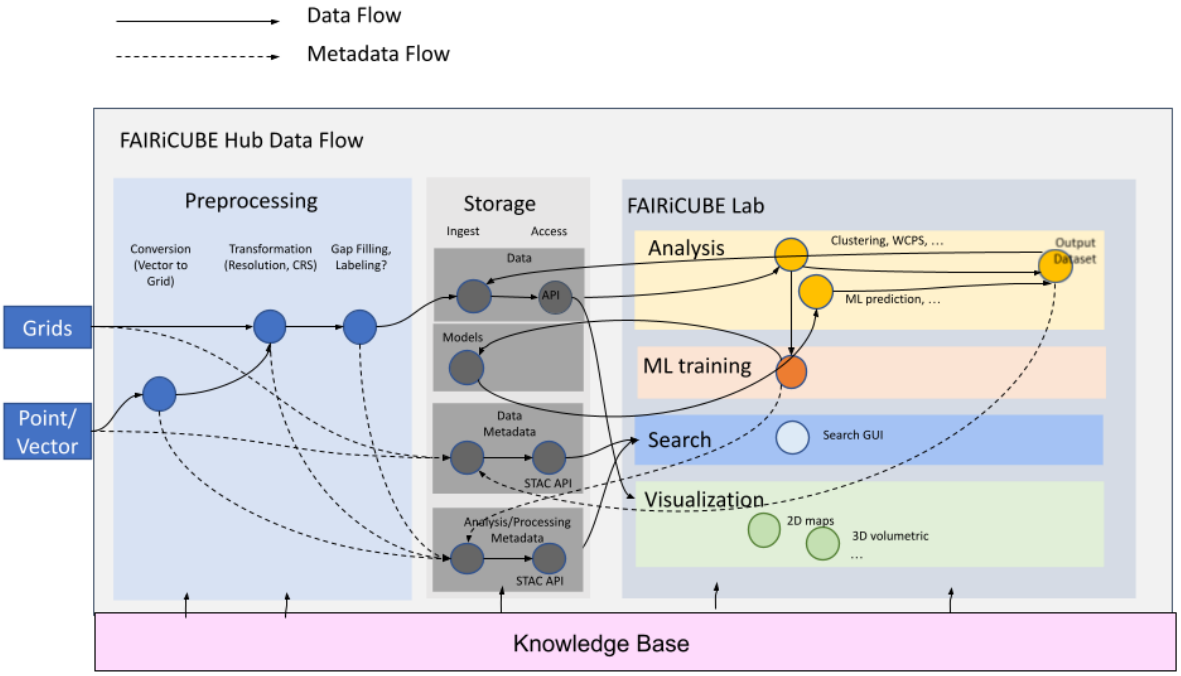

Figure 3: FAIRiPATH – Overview of the FAIRiCUBE Hub Data Flow

In Figure 3 an overview showing the flow of data and metadata within the FARiCUBE Hub system is given. Data provided will be submitted vi STAC-API to the STAC catalog holding the metadata. If required some preprocessing steps (e.g., conversions, geographic transformations and if necessary, some gap-filling procedures) might be applied to the respective dataset. Such preprocessing steps are especially useful to harmonize datasets (e.g., their resolution, projection, etc.) prior to ingestion.

Once preprocessing is finished data will be ingested into the data-store from where will be readily available to further actions applied by the user. These include further data analysis, visualization and ML-Training. All results can be submitted into the storage for further usage or fed into ML-Models.

3 FAIRiCUBE Lab (an EOxHub instance)

The EOxHub deployment, aka FAIRiCUBE Lab, is separated in two parts, the Control and the Worker Plane. The Control Plane offers the central Hub functionalities, data management and analysis, and needs to run continuously. User workloads are executed in the Worker Plane which is scaled as needed.

As a central component, the FAIRiCUBE Lab provides users a workspace where they can install apps like JupyterLab, manage service subscriptions, and administrate their data. The workspace provides a runtime for user-defined workloads. The workspace connects the control plane of the EOxHub with the worker plane.

Figure 4 shows the FAIRiCUBE Lab Architecture as Virtual Private Cloud (VPC) in a Kubernetes cluster split into control (bottom) and worker (top) plane providing the workspaces. The tooling deployed in the worker plane like JupyterLab, pygeoapi[1], or MLflow[2] is accessed by end users whereas the control plane is accessed by API or cluster management only.

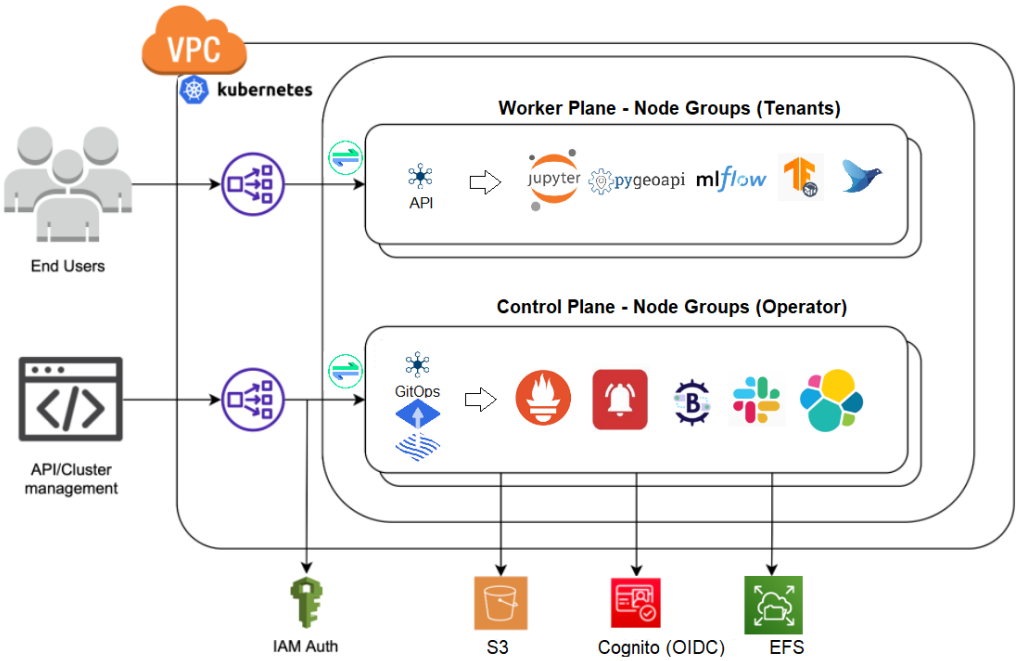

Figure 4: FAIRiCUBE Lab Architecture

3.1 Control Plane

The operator control plane provides tenant specific workspaces to individual science teams or other users. In order to do so, some stable workloads need to be always running in the Kubernetes cluster:

- User management & access control
- Metrics & monitoring
- Workload management & invocation
- Infrastructure provisioning

1 https://pygeoapi.io

2 https://mlflow.org

The control plane is declaratively deployed and operated through GitOps principles relying on the Flux CD tooling (see Section **Error! Reference source not found.** "**Error! Reference source not found.**". Important to mention is that the cluster management is nominally performed exclusively through GitOps, i.e., operators don't need to run any commands like kubectl directly on the cluster and thus don't need any access rights granted. In case of the necessity of troubleshooting or deeper debugging, operators are assuming roles via Identity and Access Management (IAM) but never via accounts directly.

The control plane relies on these managed services:

- Elastic Kubernetes Service[3] (EKS) to provide a kubernetes cluster
- S3 for object storage
- Cognito following the Open ID Connect[4] (OIDC) standard for user management
- Elastic File System[5] (EFS) for block storage

The main tools running in the control plane are

- Prometheus[6] to collect exported metrics
- Grafana[7] to provide metrics dashboards
- Alertmanager[8] to send notifications for example to dedicate operations Slack channels
- Brigade[9] for running scriptable, automated tasks
- Elastic stack of elasticsearch, fluent, and kibana[10] (EFK) to collect and present logs

3.1.1 Configuration Management

The configuration management of the FAIRiCUBE Lab is organized in a private GitHub repository (https://github.com/FAIRiCUBE/flux-config/) to manage the workspace profiles for JupyterLab sessions, installed applications for users of the FAIRiCUBE Lab teams, as well as all other team relevant configurations. Currently there are four teams (Use Cases) configured with different profiles, apps, secretes, buckets, etc. as required.

The configuration management relies on GitOps principles to deploy the desired configuration via the Flux CD operator. GitHub issues at https://github.com/FAIRiCUBE/flux-config/issues are used to track the status of various configuration requests. The sections below provide details of how the different aspects of the FAIRiCUBE configuration are managed in the central configuration management and which options are available.

3.1.2 GitHub as Identity Provider

The FAIRiCUBE Lab uses GitHub as Identity Provider. The YAML code below shows the configuration to grant an administrator and a user access to the resources of a specific FAIRiCUBE team via **userName** and **role**.

```yaml
allowedLogins:
- approvalTimestamp: "2023-06-07T00:00:00Z"
  creationTimestamp: "2023-06-07T00:00:00Z"
  email: achtsnits@eox.at
  userName: achtsnits
```

3 https://docs.aws.amazon.com/whitepapers/latest/overview-deployment-options/amazon-elastic-kubernetes-service.html

4 https://openid.net/connect/

5 https://aws.amazon.com/efs

6 https://prometheus.io

7 https://grafana.com/

8 https://prometheus.io/docs/alerting/latest/alertmanager/

9 https://v1.brigade.sh

10 https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html

```
    role: admin

  allowedLogins:
  - approvalTimestamp: "2023-06-07T00:00:00Z"
    creationTimestamp: "2023-06-07T00:00:00Z"
    email: stephan@meissl.name
    userName: schpidi
    role: user
```

3.1.3 JupyterLab Profiles

Different JupyterLab profiles can be configured and are available for authenticated and authorized users as shown in Figure 6.

The YAML code below shows the configuration for one JupyterLab profile of the Use Case 4 (UC4.

```
spec:
  k8s-namespace: faricubeuc4
  creationTimestamp: "2023-06-07T00:00:00Z"
  inventory:
  - creationTimestamp: "2023-06-07T00:00:00Z"
    entityId: eoxhub
    entityType: infra
    expirationDate: "2024-12-31"
    productKey: EOxHub - Default
  activations:
  - activationDate: "2023-06-07"
    creationTimestamp: "22023-06-07T00:00:00Z"
    data:
      profile_faricubeuc4: display_name=FAIRiCUBE-
UC4,node_purpose=useruc4,mem_guarantee=30064771072,cpu_guarantee=7,mem_limit=322122547
20,cpu_limit=7.5,s3_bucket_name=hub-faricubeuc4,secret_names=hub-faricubeuc4
      s3_bucket: "53687091200"
      storage: "53687091200"
      user:  "2592000" #30d
      useruc4: "2592000" #30d
      url: https://eoxhub.faricube.eu
    entityId: eoxhub
    entityType: infra
```

The main information is encoded in this line:

```
profile_faricubeuc4: display_name=FAIRiCUBE-
UC4,node_purpose=useruc4,mem_guarantee=30064771072,cpu_guarantee=7,mem_limit=322122547
20,cpu_limit=7.5,s3_bucket_name=hub-faricubeuc4,secret_names=hub-faricubeuc4
```

This profile, named “FAIRiCUBE-UC4”, provides a guaranteed minimum of 30 GB RAM with a maximal amount of 32GB of RAM, and a guaranteed minimum of 7.5 CPUs. It further injects a secret, as described below, and mounts the Use Case specific S3 bucket named "hub-faricubeuc4".

3.1.4 Keycloak

Keycloak is used as authorization system. Users can be granted various rights for example to get access to the shared folder to curate the shared Jupyter notebooks or to get access to conda-store to manage conda environments.

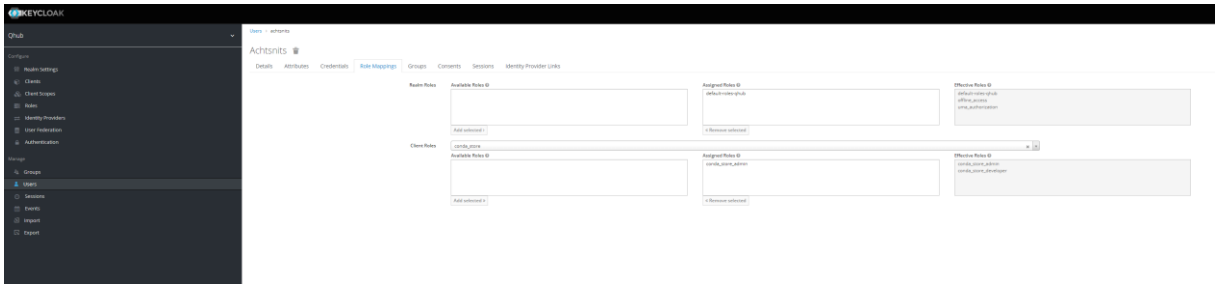

Figure 5: Keycloak for authorization

3.1.5 Shared Jupyter Notebooks

Created Jupyter notebooks can easily be shared with other FAIRiCUBE users within the same customer or team by making them available on a curated shared folder. Curation access to this shared folder is granted via Keycloak as described above. From there they are automatically picked by the FAIRiCUBE Lab and made available through the Notebook Catalog UI.

This UI is shown once a JupyterLab profile is started where the user sees the FAIRiCUBE Catalog tile on the launcher as shown in the bottom of Figure 6. Through this UI it is possible to browse, execute, and comment on Jupyter notebooks to steer interaction and foster collaboration. Also shown is the mounted S3 bucket already made available for direct access to the provided datasets and the available Machine Learning notebooks.

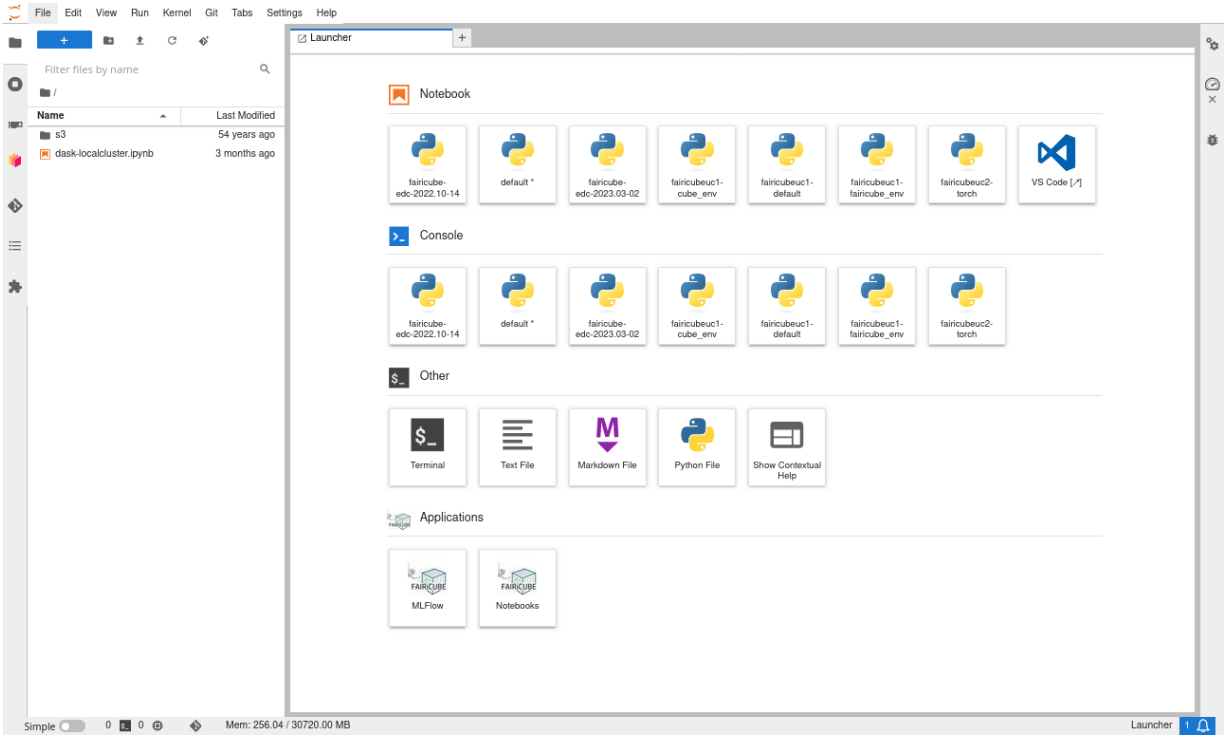

Figure 6: FAIRiCUBE Catalog launcher tile

At the final stage of expansion there will be various "Getting-started" and specific "Tutorial notebooks" available for execution in the FAIRiCUBE Catalog (as shown below).

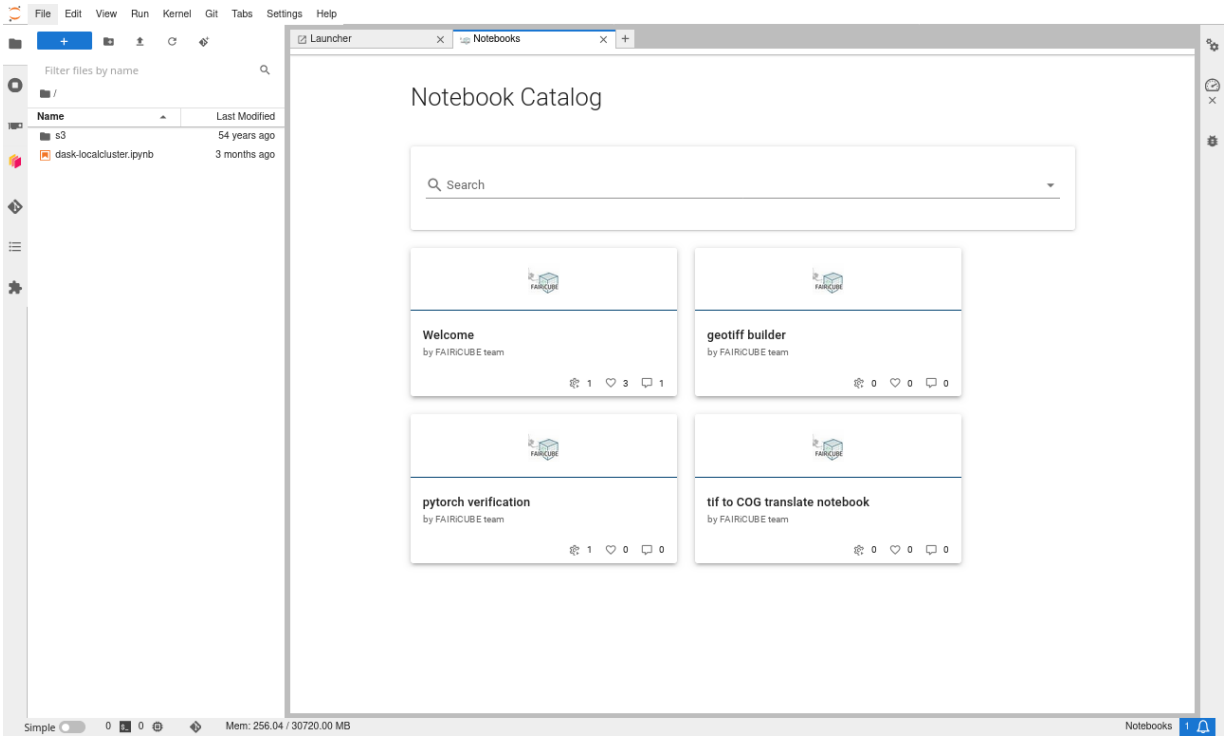

Figure 7: FAIRiCUBE Catalog Notebook Viewer

3.1.6 Shared Conda Environments

The FAIRiCUBE Lab bundles the open source conda-store tool (https://github.com/Quansight/conda-store) to provide the familiarity and flexibility of conda environments to FAIRiCUBE users at https://eoxhub.fairicube.eu/conda-store. The conda-store tool not only enables the usage of conda environments but also supports through its UI the initial environment creation as well as the sharing of created environment with other users. Figure 8 shows the conda environment management via an environment.yml specification files. Team members can be granted the permissions to curate the available environments for their team via Keycloak as described above.

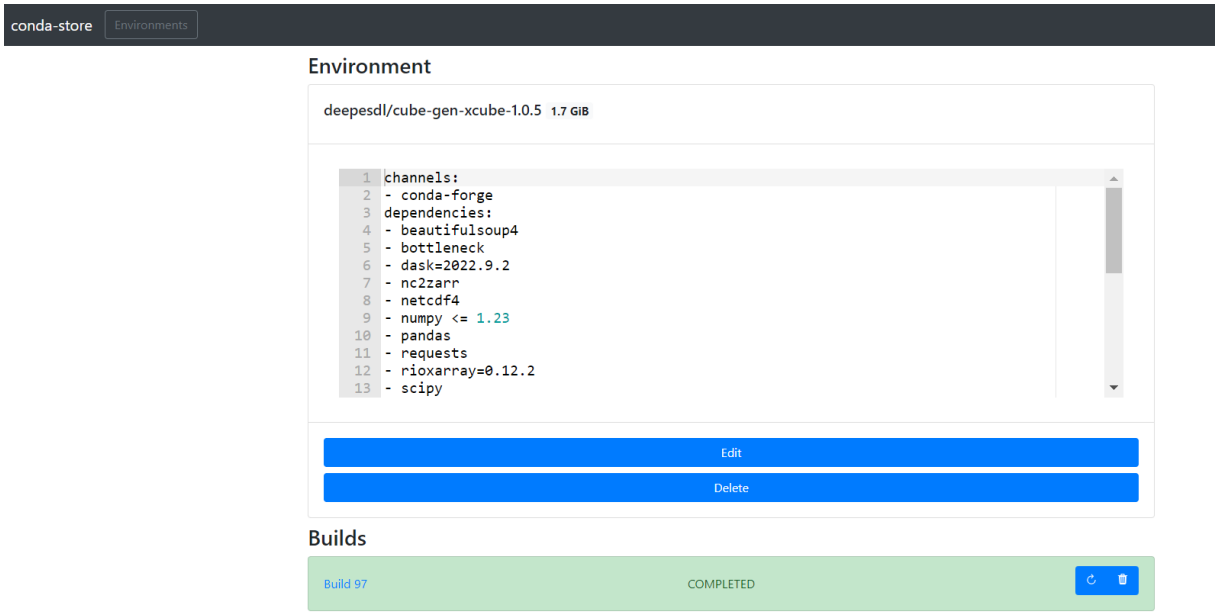

Figure 8: Example of a Conda-store environment configuration (here from DeepESDL)

The kernel or conda environment to use in a specific notebook can be adjusted with a drop-down menu in the top right corner of the notebook as shown in Figure 9.

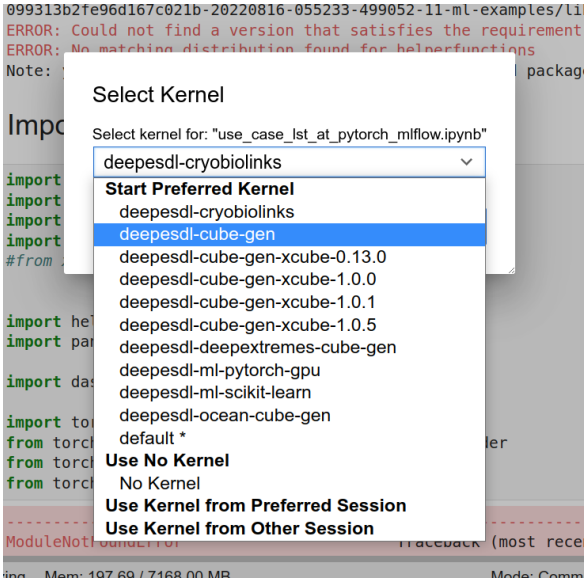

Figure 9: Examples of Kernel selection in a Conda environment (here from DeepESDL)

3.1.7 Shared Secrets

Sometimes it is necessary to share configuration values, particularly secrets, within teams. In order not the share them in external tools, with the danger of unintended disclosure, the FAIRiCUBE Lab supports shared secrets configured via the central configuration management. The secrets themselves are never stored in plaintext but only in a sealed state.

Technically, access credentials are added to JupyterLab sessions by adding them to the qhub kubernetes secret, in the corresponding namespace, via flux. K8s secret values are just base64 encrypted strings and it is always necessary to assess if it is feasible to check them in into git (even if it is a protected git repository) or not. For sensitive values it is recommended to check them in into git as encrypted values and only decrypt them within the k8s cluster. This functionality is established through the sealed-secrets tooling (https://github.com/bitnami-labs/sealed-secrets) as described in the README of the repository at https://github.com/FAIRiCUBE/flux-config#using-access-credentials-via-environment-variables.

The YAML code below shows an example of a k8s secret:

```yaml
apiVersion: v1
data:
  test: cGFzc2Vk #your secret name and its value base64 encoded
kind: Secret
metadata:
  name: qhub
  namespace: fairicubeuc4
type: Opaque
```

3.1.8 Shared Object Storage

FAIRiCUBE Lab users are granted with ready-made access to object storage (S3), i.e., all necessary access details like bucket name and credentials are available during runtime as environment variables.

This allows the users to directly leverage curated datasets made available to their team as well as to curate datasets and other output artefacts themselves and share them within their team.

The team setup and the granted quota for storage is centrally managed in the configuration system using a YAML code like the one below.

```yaml
s3_bucket: "53687091200"
```

It is also possible to disseminate the team's data through other means, e.g., via public endpoints or through Sentinel Hub.

3.1.9 Apps

Common data science and ML tooling may be pre-installed in conda environments on a per team basis and are therefore available during runtime. In addition, it is also possible to install "always-running" apps, to analyse ML experiment runs or share results without the need of a running a FAIRiCUBE Lab JupyterLab session.

One example is MLflow tracking work with different ML frameworks like "scikit-learn" or "pytorch" as shown in Figure 10.

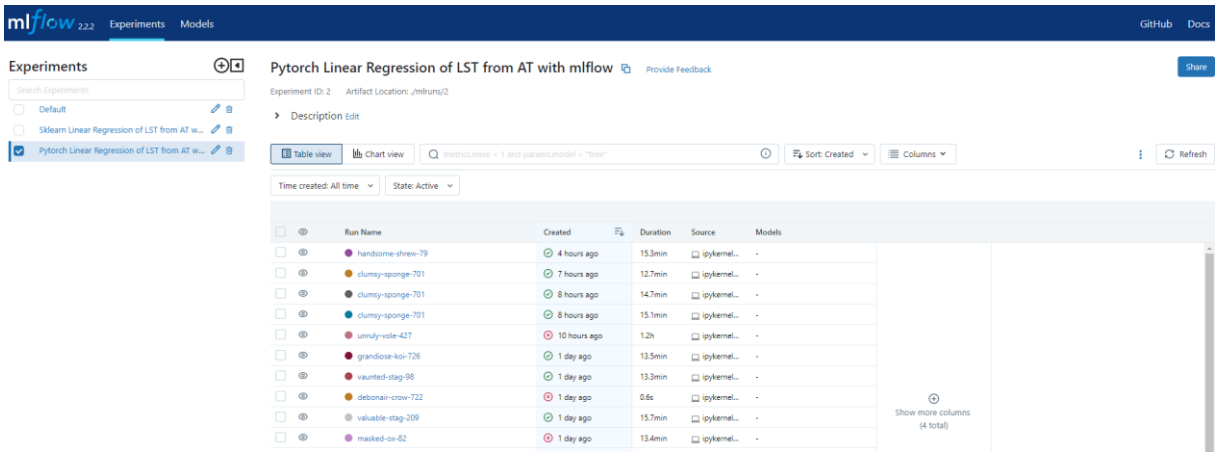

Figure 10: Example of MLflow for experiment tracking (here from DeepESDL)

3.1.10 Complete Configuration Example

The YAML code below is a complete customer.yaml configuration for the fairicubeuc4 the examples above.

```yaml
apiVersion: hub.eox.at/v1alpha1
kind: Customer
metadata:
  name: fairicubeuc4
  namespace: core
spec:
  k8s-namespace: fairicubeuc4
  creationTimestamp: "2023-06-07T00:00:00Z"
  inventory:
  - creationTimestamp: "2023-06-07T00:00:00Z"
    entityId: eoxhub
    entityType: infra
    expirationDate: "2024-12-31"
    productKey: EOxHub - Default
  activations:
  - activationDate: "2023-06-07"
```

```
  creationTimestamp: "2023-06-07T00:00:00Z"
  data:
    profile_fairicubeuc4: "display_name=FAIRiCUBE-
UC4,node_purpose=useruc4,mem_guarantee=30064771072,cpu_guarantee=7,mem_limit=322122547
20,cpu_limit=7.5,s3_bucket_name=hub-fairicubeuc4,secret_names=hub-fairicubeuc4"
    s3_bucket: "53687091200"
    storage: "53687091200"
    user: "2592000" #30d
    useruc4: "259200" #30d
    url: https://eoxhub.fairicube.eu
  entityId: eoxhub
  entityType: infra
properties: {}
allowedLogins:
- approvalTimestamp: "2023-06-07T00:00:00Z"
  creationTimestamp: "2023-06-07T00:00:00Z"
  email: achtsnits@eox.at
  userName: achtsnits
  role: admin
- approvalTimestamp: "2023-06-07T00:00:00Z"
  creationTimestamp: "2023-06-07T00:00:00Z"
  email: stephan@meissl.name
  userName: schpidi
  role: user
```


3.2 Worker Plane

Via the control plane, workloads are deployed and scheduled on the worker plane

The multi-tenant worker plane is responsible for the following tasks:

- Workload orchestration and scheduling on dynamically allocated cloud resources (e.g., GPU nodes)
- User code execution on top of custom environments based on needs of science teams for example on custom base images
- Flexible in terms of installed tooling (i.e., dynamic deployment via API)

These are the apps or tooling which either are deployed or can readily be deployed in the worker plane as shown in the figure above:

- JupyterLab to interactively execute Jupyter notebooks written mostly in Python
- pygeoapi to programmatically execute user workloads for example Jupyter notebooks
- DVC[11] (Data Version Control) for collaborative data management like ML artefacts
- MLflow or TensorBoard[12] to support Machine Learning operations (MLOps[13])
- Almost any Docker image can be deployed and run in the worker plane
- fluentd[14] to collect logs for debugging

11 https://dvc.org

12 https://www.tensorflow.org/tensorboard

13 https://ml-ops.org

14 https://docs.fluentd.org

Further apps can and will be made available in the course of the project execution, for example, to demonstrate the results of the use cases in a FAIRiCUBE Viewer.

There are many apps and tools available to support the Machine Learning Operations (MLOps) pipeline as shown in Figure 11. Jupyter notebooks run in JupyterLab together with jupytext, git, and DVC support. They document the first steps from data retrieval & ingestion via data preparation to model training with versioned source code and data management and pipeline definitions. Further tools like MLflow, TensorBoard, pygeoapi, EOxHub itself, SQLite, S3, Grafana, and Prometheus support the model training, model evaluation & tuning, deployment in production, and monitoring of the system. The workspace allows for sharing ML activities for reproducibility within a team or even making them publicly accessible.

The core app deployed in each EOxHub workspace is a managed JupyterLab, allowing the interactive execution of Jupyter notebooks close to the data. Jupyter notebooks can be executed either interactively, for example to develop an algorithm, or in a headless way using a REST API provided by pygeoapi.

FAIRiCUBE Lab provides Cloud Workspaces through the workflow management runtime for docker containers and relies on object storage to persist data as shown in the figure below. It offers science teams, projects, communities, etc. a cloud footprint with a subscription – a so-called "Workspace as a Service".

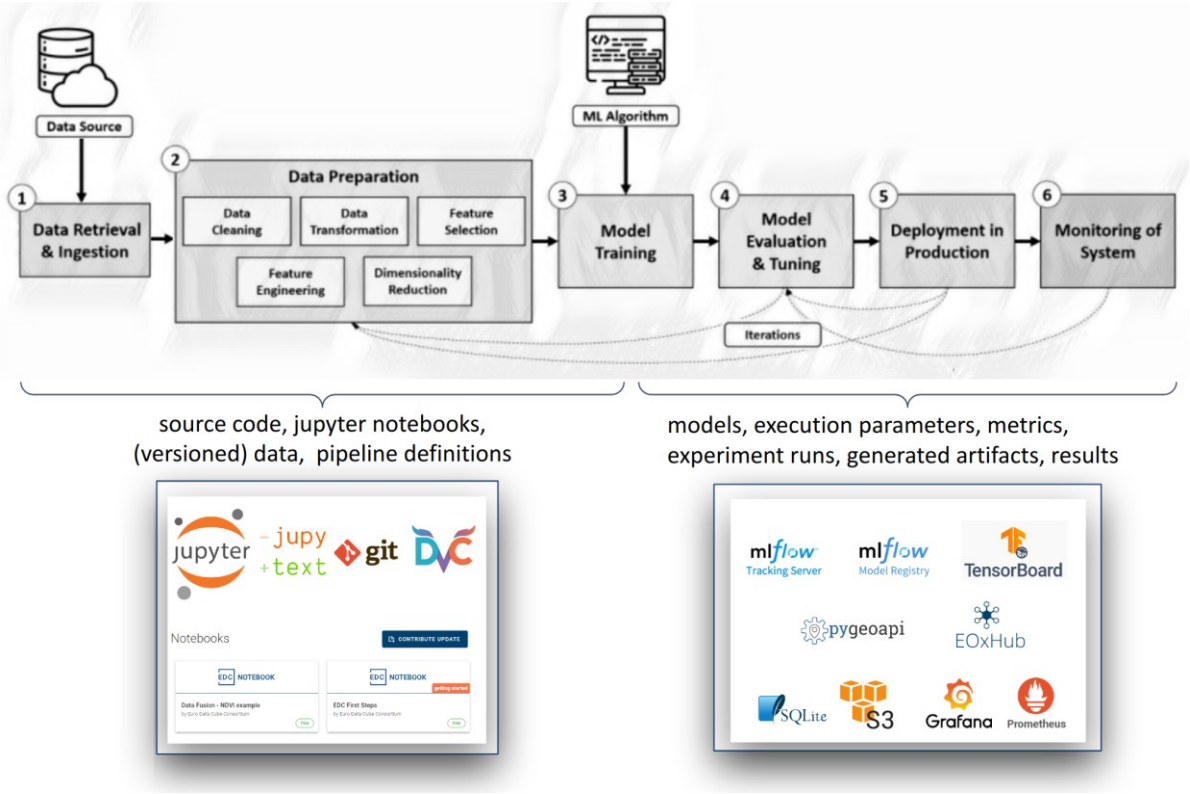

Figure 11: MLOps Tooling

3.2.1 Interactive Development Environment - Jupyter Notebooks

The FAIRiCUBE Lab is centred on the usage of Jupyter Notebooks, allowing to execute them close to the data for simple exchange and sharing of processing modules. The notebooks can either be executed interactively using a managed JupyterLab environment, for example to develop an algorithm, or in a headless way using a REST API, meaning without a graphical user interface.

The available data can be accessed in these Jupyter notebooks via different means depending on what is best suited for the use case at hand and the skill level of the user. The available options span from

direct object storage access via the xcube or xarray Python libraries to the Process API of Sentinel Hub and Open Geospatial Consortium (OGC) defined interfaces like the Web Coverage Service (WCS) and Web Coverage Processing Service (WCPS). Jupyter notebooks can further utilize libraries like dask to parallelize and scale processing jobs.

FAIRiCUBE Lab defines (via control plane) different computational profiles, as required by the Use Cases, which are sets of kernel environments with specific configurations. Configurations differentiate by variables like available GPU/CPU, memory size or storage capacity. Profiles can also be distinguished by configuration of different Conda kernels.

The JupyterLab in FAIRiCUBE offers Conda and Conda store, a popular package manager for Python, to create, manage and install environments with specific sets of packages and dependencies. It is possible to manage different kernels with different settings for separated projects or profiles.

JupyterLab will be used as a development environment for machine learning models training and deployment. In combination with MLflow setup this can serve as a complete machine learning lifecycle development environment.

3.2.2 Data access

Data delivered via SentinelHub API can be integrated to be accessed directly within the JupyterLab environment and used as input for model predictions or for training and development of algorithms. Results of model inferences are storable and downloadable from within the FAIRiCUBE environment or can be transferred elsewhere.

3.2.3 User access

The FAIRiCUBE Lab is deployed at https://eoxhub.fairicube.eu. The authentication is using GitHub as identity provider as shown in Figure 12.

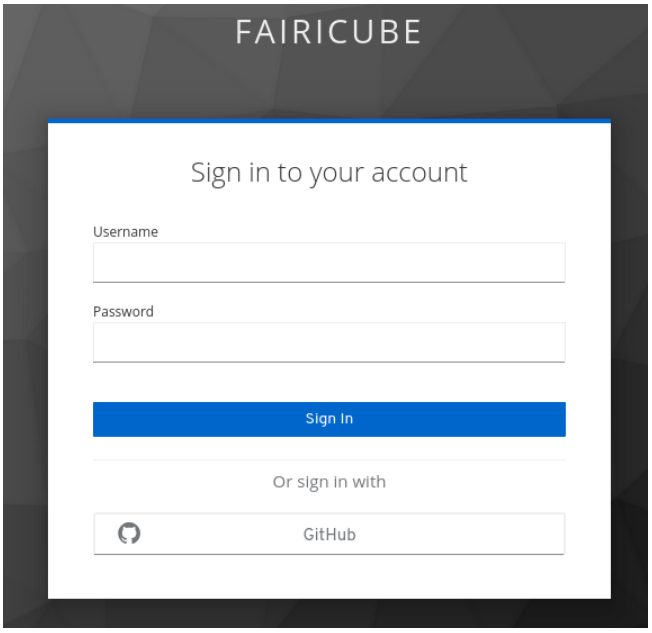

Figure 12: FAIRiCUBE HUB Login

After a successful login the main tool provided by the FAIRiCUBE HUB Lab is JupyterHub allowing to start configured JupyterLab profiles as shown in Figure 6. Configuration details are provided below.

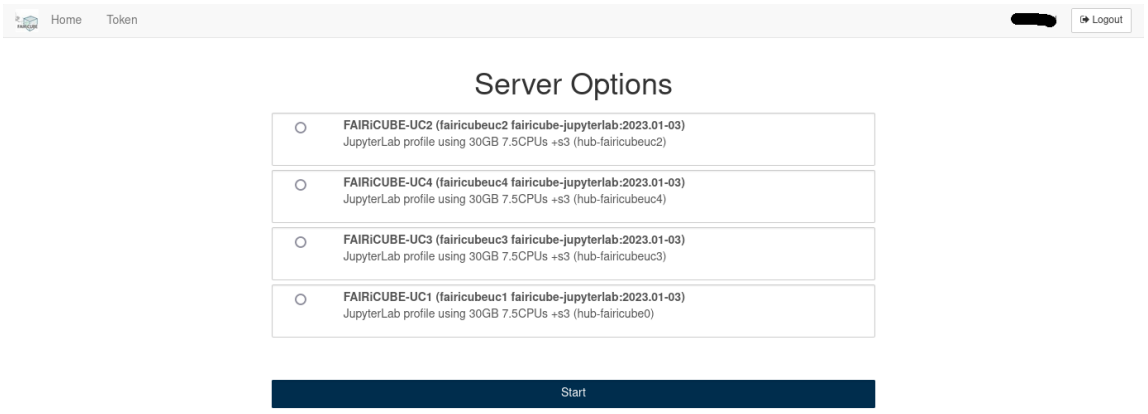

Figure 13: JupyterLab profile selection - Use Case specific workspace profiles

Once the desired Use Case specific workspace profile is selected and the "Start Button" has been pressed, a user specific server will be started, as shown in Figure 14. The start-up procedure might require some minutes since the whole workspace has to be prepared and provisioned.

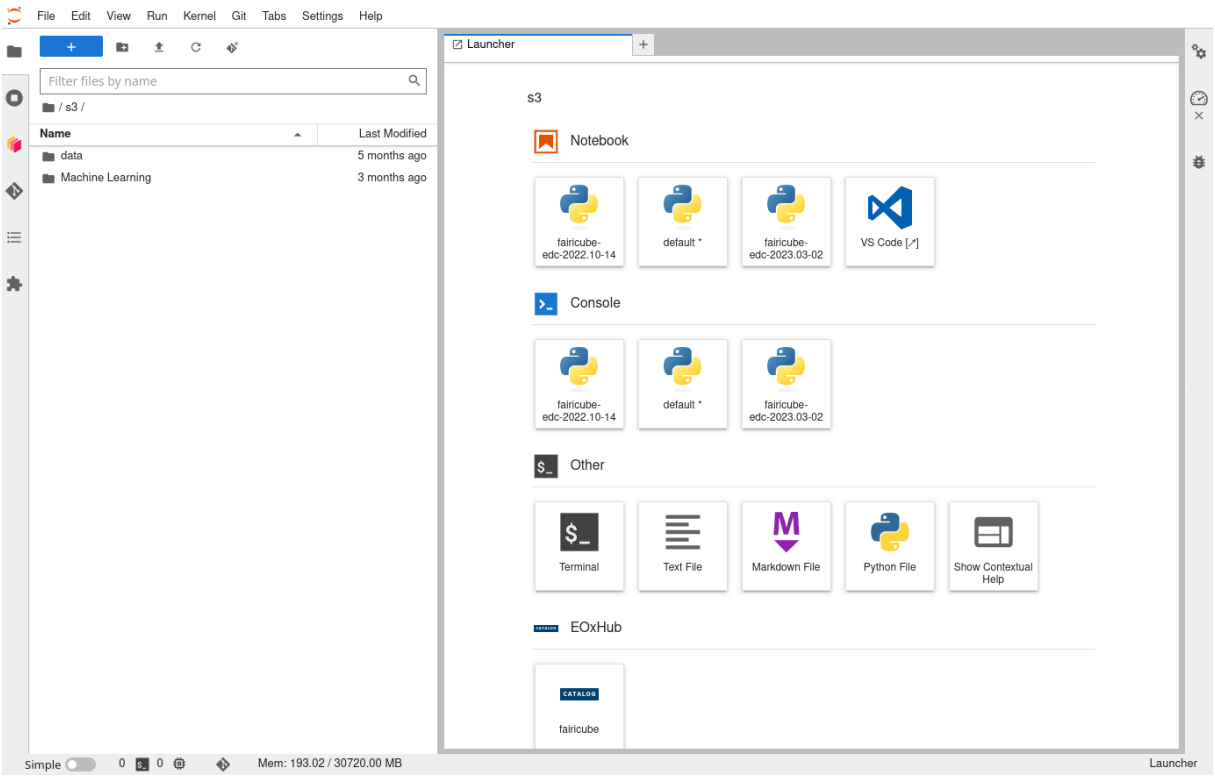

Figure 14: JupyterLab workspace launcher

Beside the User's JupyterLab workspace JupyterHub further provides the JupyterHub Control panel, which can be reached via the Main Menu entries: File → Hub Control Panel.

This Control panel can be used to Stop the User's server at the end of a working Session (Figure 15). However, every inactive User session will be terminated by the system after a pre-defined time interval of inactivity (culling). This feature is implemented to avoid unnecessary costs due to forgotten sessions.

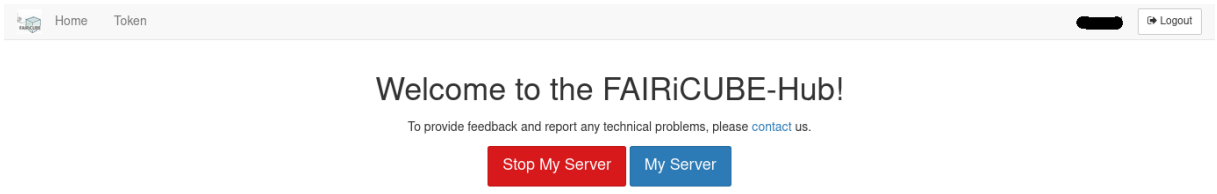

Figure 15: JupyterHub Control panel

In rare cases, e.g., when a user session hangs, it can be necessary to Stop a server manually (Figure 15). Thereafter the server can also be started again manually (Figure 16).

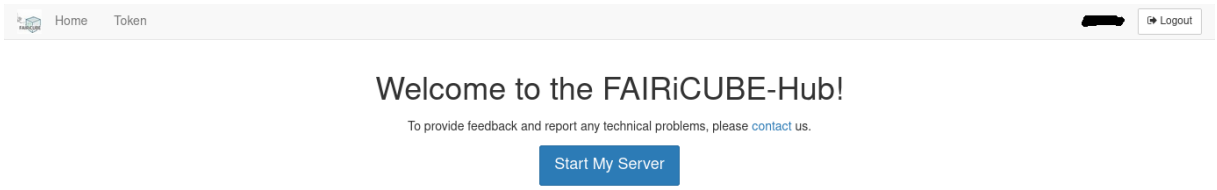

Figure 16: Greeting page after successful Login

If a user logs in and currently has no access rights to any of the Use Case workspaces, the User will be confronted with the following screen (Figure 17).

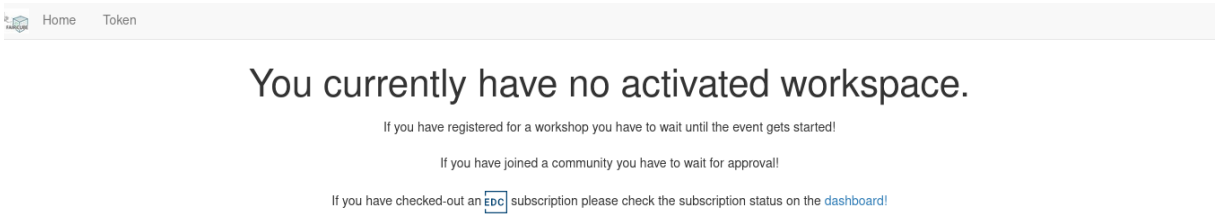

Figure 17: Screen presented to Users not configured to a Use Case

It has to be noted here that the access to every FAIRiCUBE Use Case has to be configured manually by the operators for each Use case via https://github.com/FAIRiCUBE/flux-config/ e.g., for UC4 this can be done at: https://github.com/FAIRiCUBE/flux-config/blob/master/fairicubeuc4/customer.yaml before access is possible for the specific user.

3.3 Machine Learning Platform - MLflow

Each Use Case team decides which apps shall be made available. One of these apps is MLflow.

MLflow is an open-source platform for managing the end-to-end machine learning lifecycle. It allows the user to track experiments, package code into reproducible runs, and share and deploy models. MLflow can be incorporated into Jupyter notebooks or other code and supports multiple programming languages. MLflow provides a comprehensive solution for managing the machine learning lifecycle, from tracking experiments to deploying models. It is widely used in industry and academia and is constantly evolving to support the latest trends and technologies in the field of machine learning. At a high level, MLflow consists of four main components: tracking, projects, models, and registry.

All components can be accessed via Python code in the FAIRiCUBE Lab.

3.3.1 MLflow Tracking

The MLflow tracking component (Figure 10) allows users to log and track training parameters, code, and output metrics from their machine learning experiments. The tracking component provides an API

for Python, R, and other languages, as well as an UI for visualising experiments and comparing different runs. The tracking server can store data in various backends, including a local file system, an Amazon S3 bucket, or a PostgreSQL database.

MLflow Tracking uses the concept of runs, which are executions of some piece of data science code, e.g., training of models. MLflow Tracking supports auto-logging for many classic libraries such as TensorFlow, Scikit-Learn, Spark, or Pytorch, but manual logging is available in other cases. Runs can be stored as local files, on a remote server, or into an SQLAlchemy compatible database. The tracking UI allows to directly visualise tracked metrics and search for the best components.

3.3.2 MLflow Projects

The MLflow projects component provides a standard format for packaging and distributing machine learning code, including dependencies, in a reproducible way. Projects can be run locally or on a cluster, and MLflow can manage the environment and dependencies for each run. In addition, the Projects component includes an API and command-line tools for running projects, making it possible to chain together projects into workflows. In the ML project file, it is possible to define the software environment and entry points with parameters to define workflow.

3.3.3 MLflow Models

The models component allows users to easily package models in a standard format for deployment. Models can be exported in multiple formats, including TensorFlow, PyTorch, and ONNX, and can be deployed using a variety of tools, including Docker, Kubernetes, and Amazon SageMaker. It is also possible to access models with standard ways as REST API or batch inference on Apache Spark

Native flavours allow MLflow models to be treated with corresponding functions without the need to integrate tools with each library. Flavours can be defined in the MLmodel file. Model signatures are defining outputs and inputs needed for deploying models as a REST API. The Model API allows saving, loading, and logging of the model also as adding different flavours. MLflow also provides an evaluate API to evaluate previously built models on one or more datasets.

3.3.4 MLflow Model Registry

Finally, the MLflow Model Registry component allows users to store, manage, and deploy models in a central repository. Models can be versioned, and access can be controlled using role-based access control. The Model Registry works both in UI and API versions. It provides model lineage (which MLflow experiment and run produced the model), model versioning, stage transitions (for example from staging to production), and annotations. Model versioning allows models to be archived and redeployed in the future.

4 Datasets & Models

The FAIRiCUBE Hub provides access to a wider variety of datasets, processes, and models. These can either be available local or external (federated) to FAIRiCUBE. Figure 18 shows the FAIRiCUBE Hub architecture with a focus on datasets, processes, and models.

The FAIRiCUBE Catalog holds metadata for datasets as well as processes. Datasets are either local or external ones. Processes are either algorithms provided mainly as Jupyter notebooks, invokable models, or specific deployed services or apps.

External datasets are federated from the Euro Data Cube (EDC) or from the EarthServer Federation.

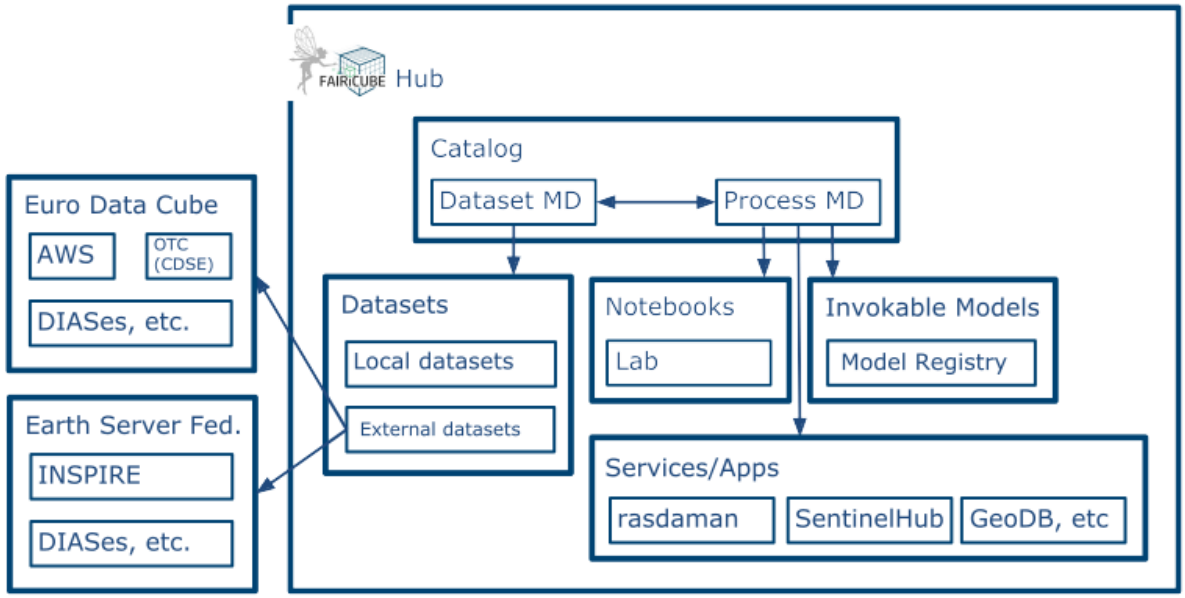

Figure 18: FAIRiCUBE Hub Architecture – Datasets, Processes, & Models

If a dataset or a process/analysis resource is not already available in FAIRiCUBE via one of the provided Datastores (e.g., AWS, DIASes, Euro Data Cube, EarthServer Fed., etc.) then a User can issue a Data Request to get the data ingested into FAIRiCUBE.

Two type of request are provided by FAIRiCUBE:

- https://github.com/FAIRiCUBE/data-requestshttps://github.com/FAIRiCUBE/data-requests/issues/new?assignees=Mohinem%2CSchpidi&labels=data-request&projects=&template=data-request.yml**resource-metadata:** Collect information for processing/analysis resources as well as propose a change to a codelist. (https://github.com/FAIRiCUBE/resource-metadata)
  - Issue resource-metadata: https://github.com/FAIRiCUBE/resource-metadata/issues/new?assignees=&labels=&projects=&template=metadata-request.yml
  - Issue codelist change: https://github.com/FAIRiCUBE/resource-metadata/issues/new?assignees=&labels=&projects=&template=codelist_change_proposal.yml
- **data-requests:** Request for data to be made available within FAIRiCUBE Lab.
  - Such a data request can be initiated via a provided WebGUI available at https://fairicube.eu/datarequest.html

In general, before submitting a request, a user wishing to use a specific dataset or resource on the FAIRiCUBE Hub should first take a look at the FAIRiCUBE Catalog (https://catalog.fairicube.eu, Figure 25-29) to see if the dataset or resource is already available and ready to be used. The contents of this

FAIRiCUBE Catalog are maintained in a GitHub repository[15] as static json files which are exposed via GitHub pages together with this dynamic web interface, i.e. the STAC-fastapi Browser (Figure 25) .

In case it is not yet listed in the catalog, the user should further consult the resource-metadata GitHub issues (https://github.com/FAIRiCUBE/resource-metadata/issues , Figure 19), to check if the dataset is already in a draft request state. In case the dataresource is already listed, the user can directly convert the draft into an issue in the GitHub repository where progress is further tracked.

When the dataset is neither already present in the FAIRiCUBE Catalog nor in the resource-metadata GitHub project, the user should issue a new request utilizing the respective links and the issue template as shown in Figure 20:

- Issue resource-metadata: https://github.com/FAIRiCUBE/resource-metadata/issues/new?assignees=&labels=&projects=&template=metadata-request.yml
- Issue codelist change: https://github.com/FAIRiCUBE/resource-metadata/issues/new?assignees=&labels=&projects=&template=codelist_change_proposal.yml

To add new **resource-metadata** to the FAIRiCUBE Catalog or request a **codelist change** the following request procedure has been defined (links provided above):

- The resource-metadata request procedure relies on GitHub, where each request is handled as an issue created by the resource requester, and then addressed together with one of the ingestion handling partners.
- A resource-metadata request issue template has been created to assure that the data requester provides all required information.
- The GitHub issue management is used for problems raised, addressed, and solved pertaining to a specific data request, providing full traceability.
- Once all uncertainties are resolved the GitHub issue is labelled approved, which will trigger the ingestion process.

15 https://github.com/FAIRiCUBE/catalog

https://catalog.fairicube.eu/

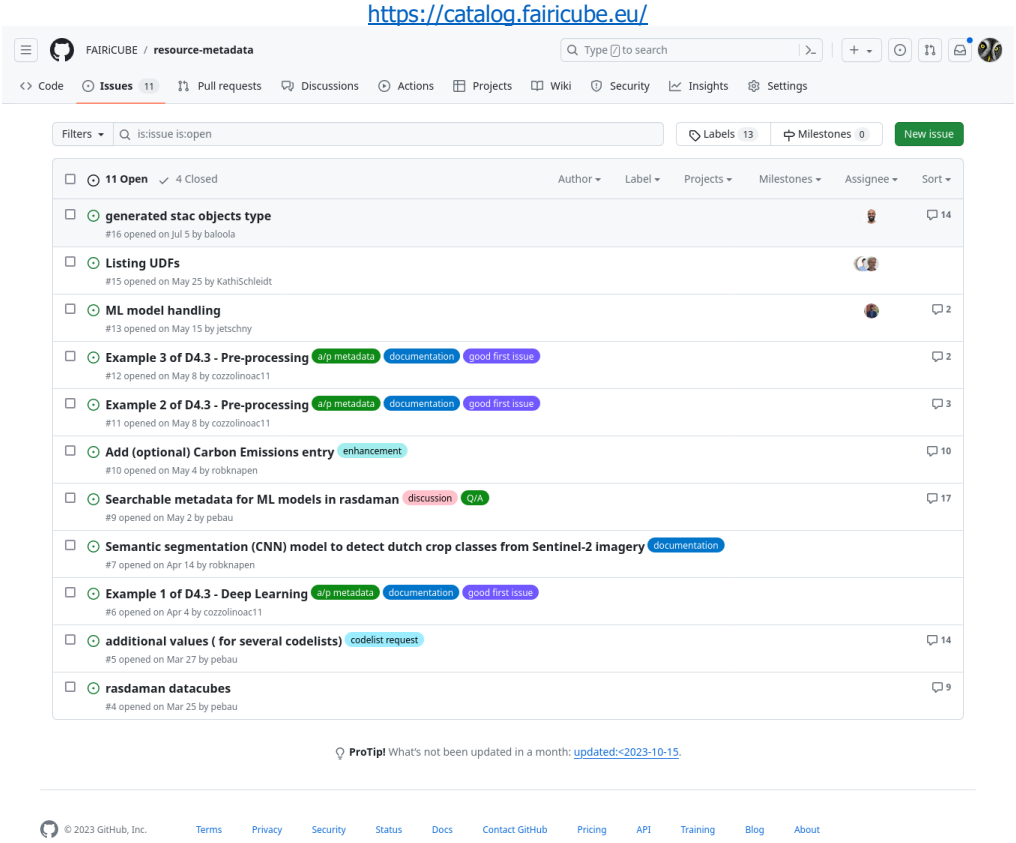

Figure 19: Resource-metadta GitHub Issues

The Resource Metadata Request can be reached at: https://github.com/FAIRiCUBE/resource-metadata/issues/new?assignees=&labels=&projects=&template=metadata-request.yml

https://github.com/orgs/FAIRiCUBE/projects/1https://github.com/FAIRiCUBE/data-requests

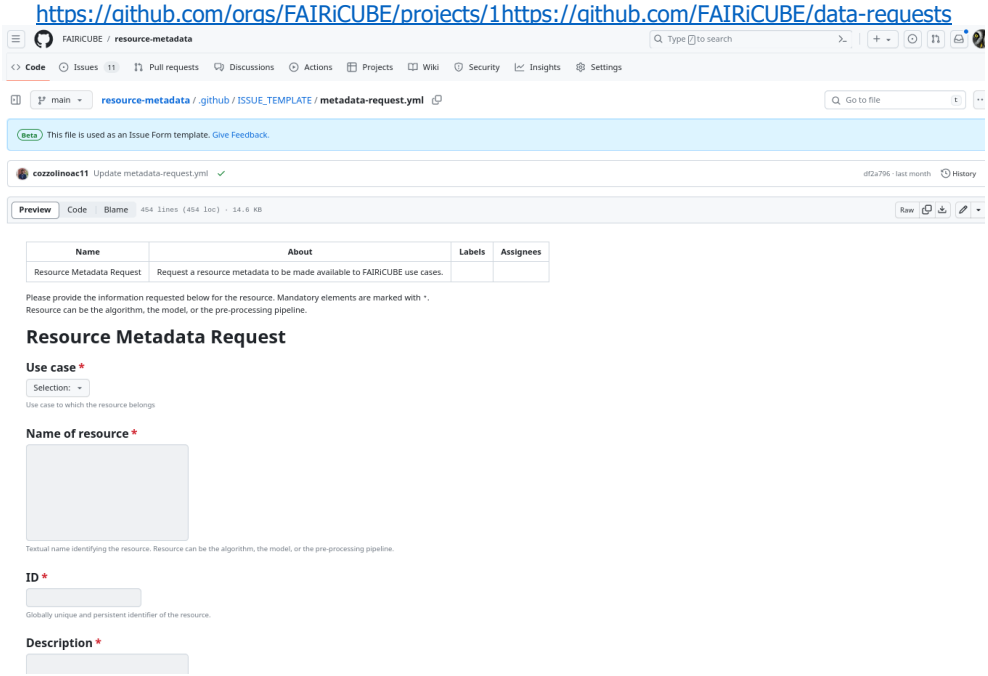

Figure 20: Resource Metadata Request

The Codelist change proposal can be reached at: https://github.com/FAIRiCUBE/resource-metadata/issues/new?assignees=&labels=&projects=&template=codelist_change_proposal.yml.

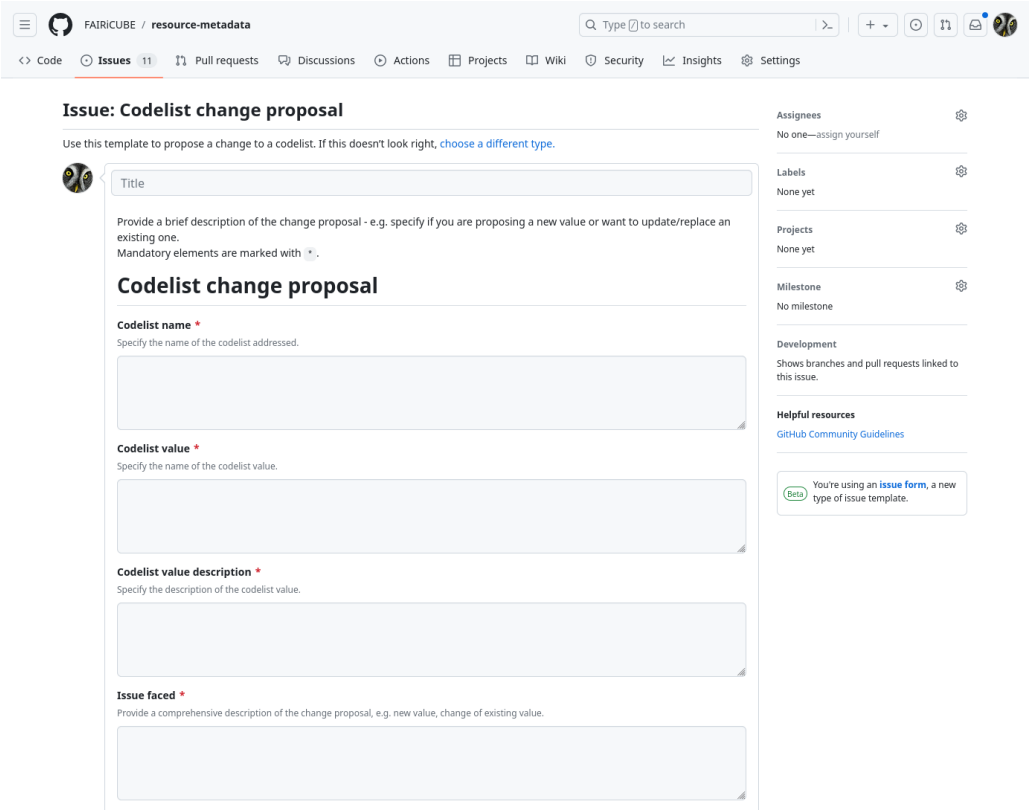

Figure 21: Codelist change proposal

In the case a user wants to have a **additional dataset** added, the user needs to create a data request using the data request WebGUI (https://fairicube.eu/datarequest.html), which ensures that all required metadata is provided, proper tags are added, and all relevant people are notified. With the submission of the WebGUI form a GitHub Pull Request is issued as a new branch which the user is automatically watching and thus receiving notifications of updates depending on their GitHub notifications configuration.

Any new data request is addressed by the requester together with one of the ingestion handling partners. Any progress, problems, discussions, etc. shall be documented in an GitHub issue associated to the respective Pull Request, so that everybody interested can follow the progress and provide additional feedback or information as necessary.

The following procedure for a data request has been set up and is shown in Figure 24.

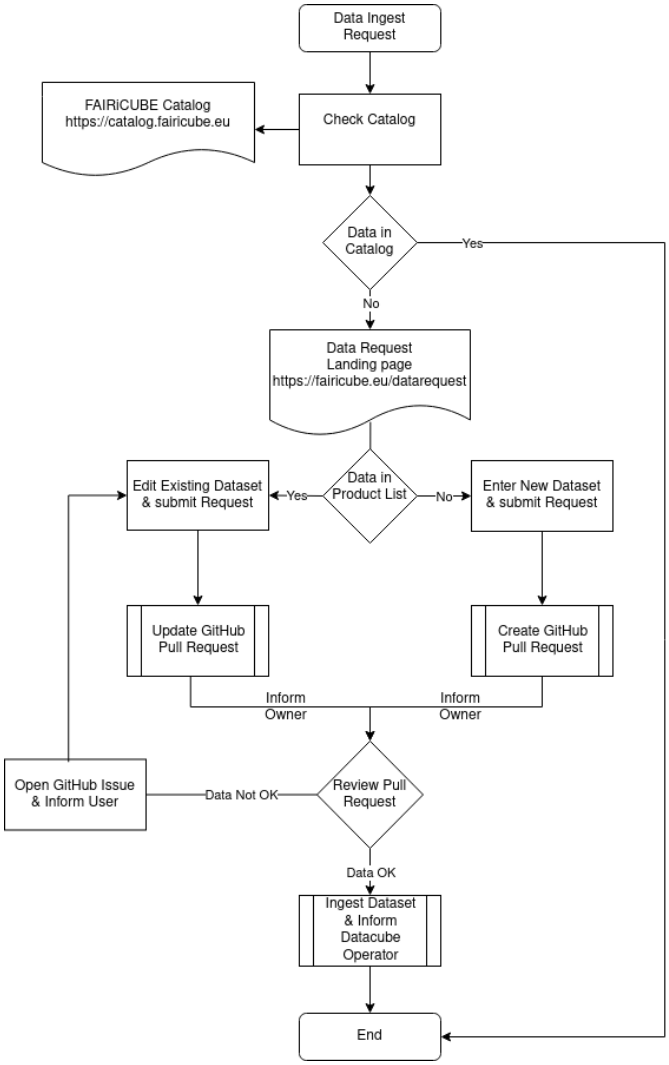

Figure 22: Data Ingestion Request Procedure

Once all metadata and data requirements are fulfilled and confirmed by the data requester, the ingestion handling partners will perform the merge and the Pull request will be closed. The respective branch in GitHub will also be closed and deleted. Any issues and discussions associated with the Pull Request are still available after the branch has been merged and deleted.

Figure 38 shows the Landing page of the Data Ingestion Request, providing also a listing of available datasets, and allowing for editing of already provided metadata.

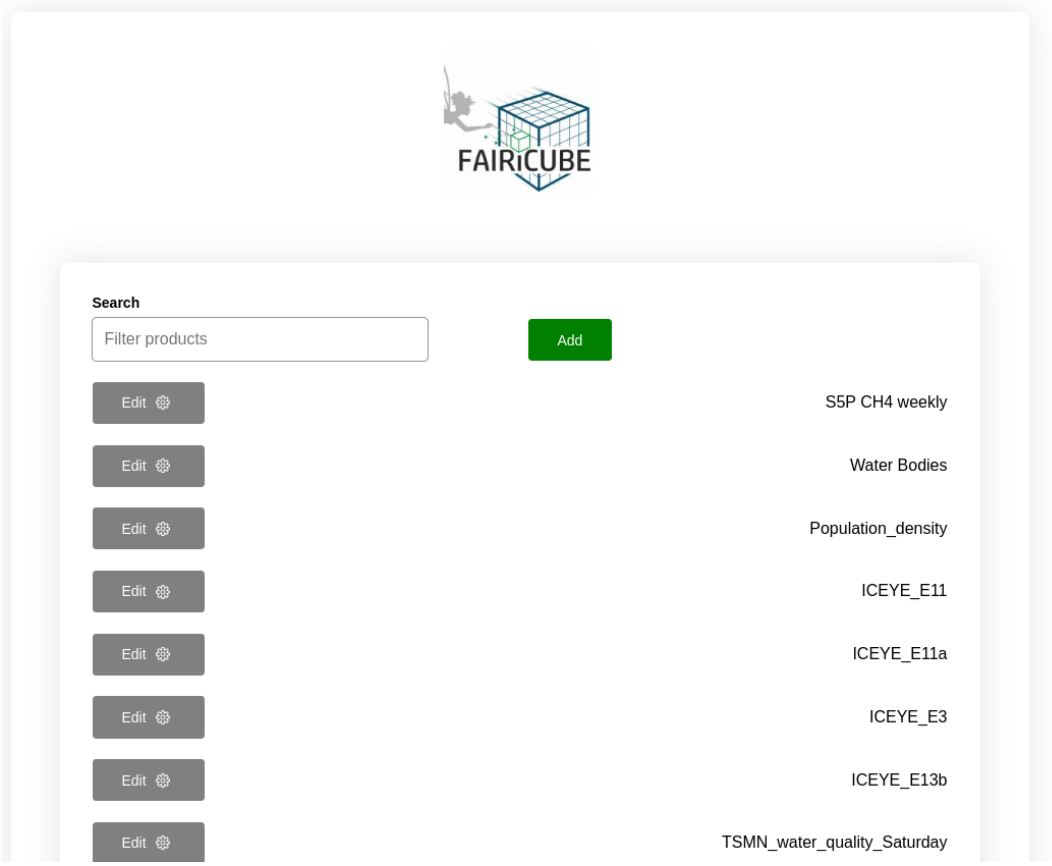

Figure 23: Data request WebGUI - Landing page

Figure 24 shows the Entry Form of the Data request WebGUI. Here the necessary metadata has to be provided in order to enable the data ingestion process.

In the *editing* mode, already provided metadata will be filled into the respective fields of the Entry Form (Figure 24).

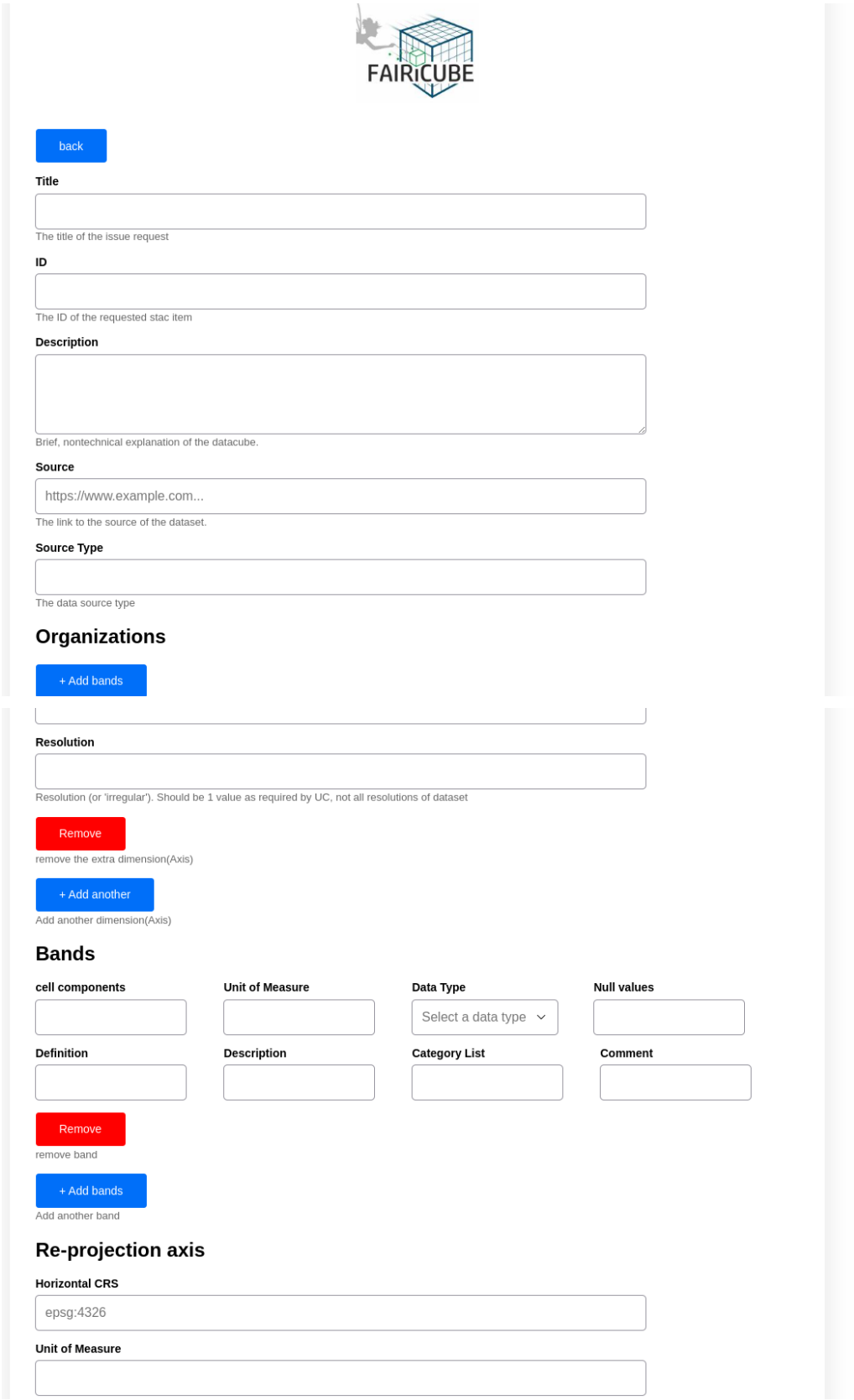

Figure 24: Data request WebGUI - Entry Form (2 sections are shown)

When the merge is done the newly submitted data is available as a STAC item to the STAC Browser. The dynamic catalog using the STAC Browser is currently deployed at https://catalog.fairicube.eu. The STAC Browser provides additional features like searching, which are not available in the static STAC catalog. The next screenshots provide various views on the STAC Browser pages showing available catalogs and the search interface.

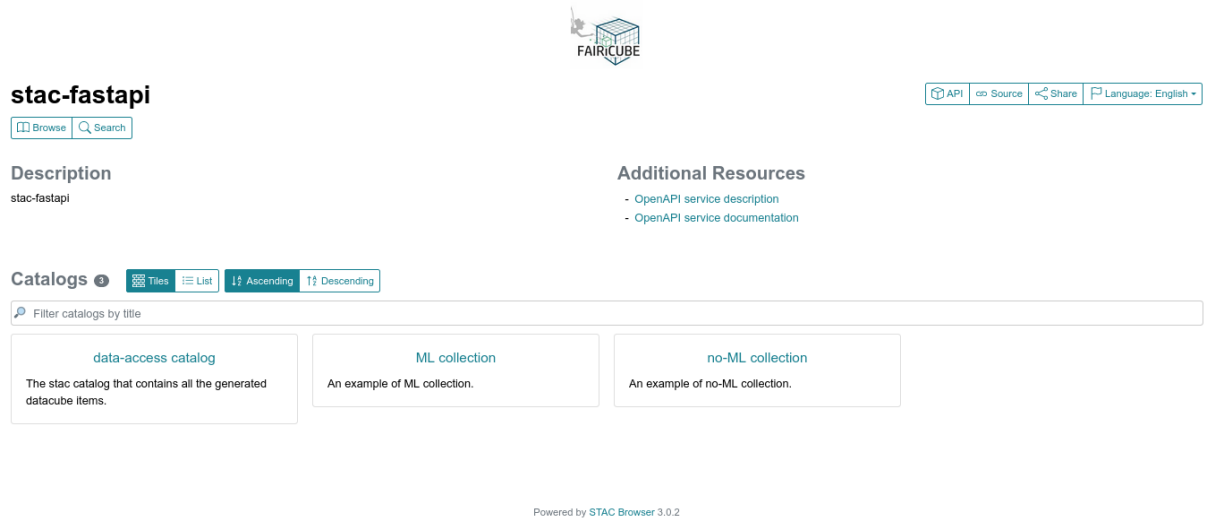

Figure 25: FAIRiCUBE STAC Browser Interface

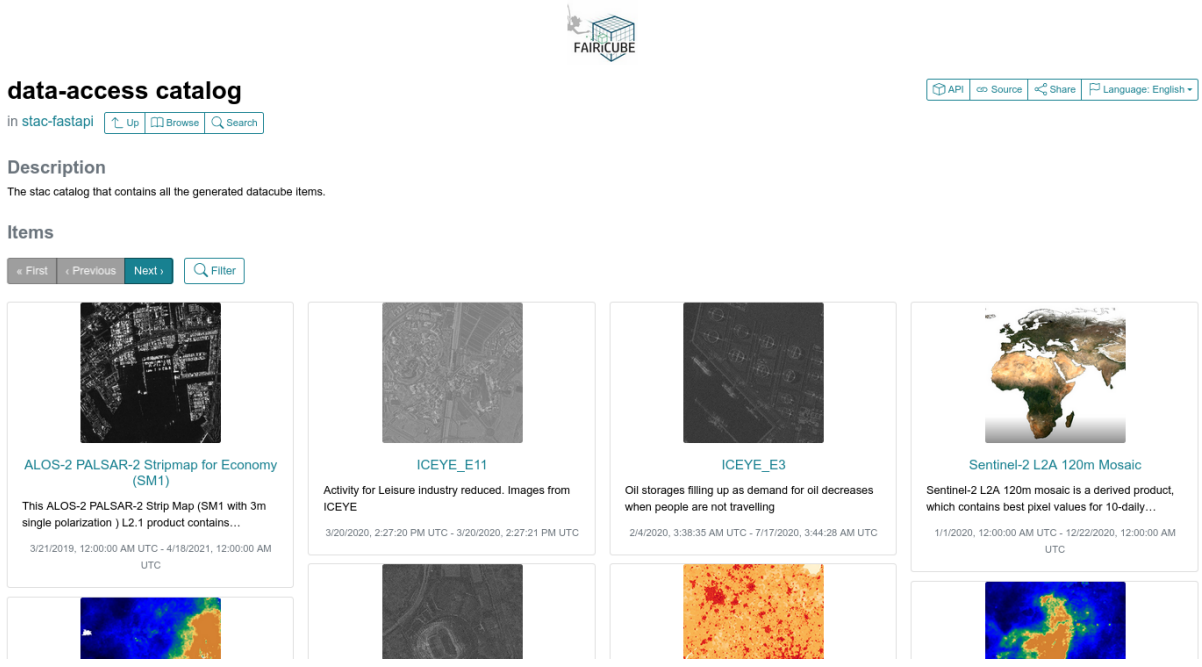

Figure 26: Data Access Catalog

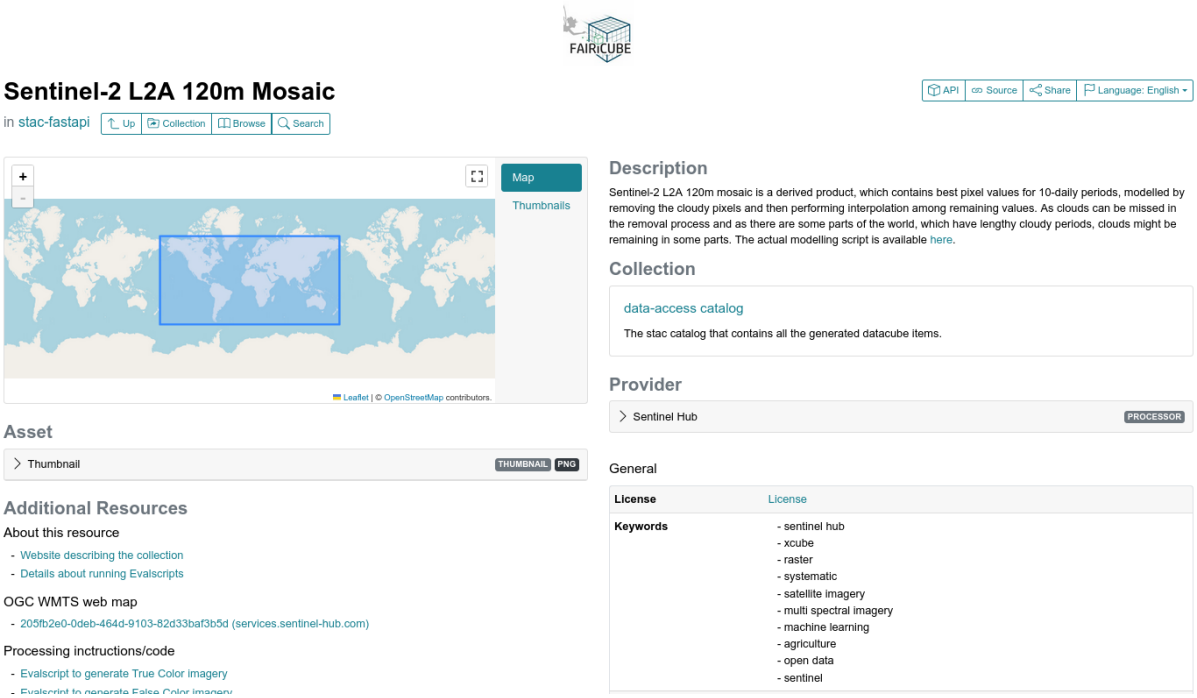

Figure 27: Details of a Dataset (Sentinel-2 L2A 120m Mosaic)

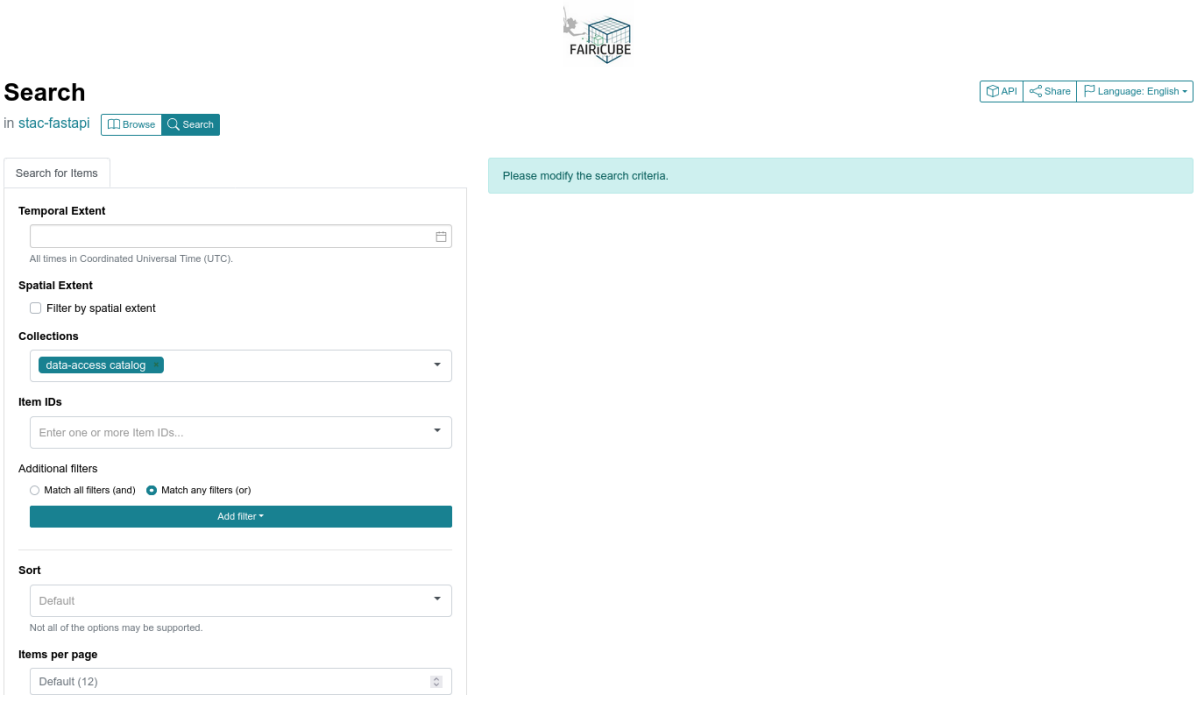

Figure 28 Interface for a Dataset (Sentinel-2 L2A 120m Mosaic)

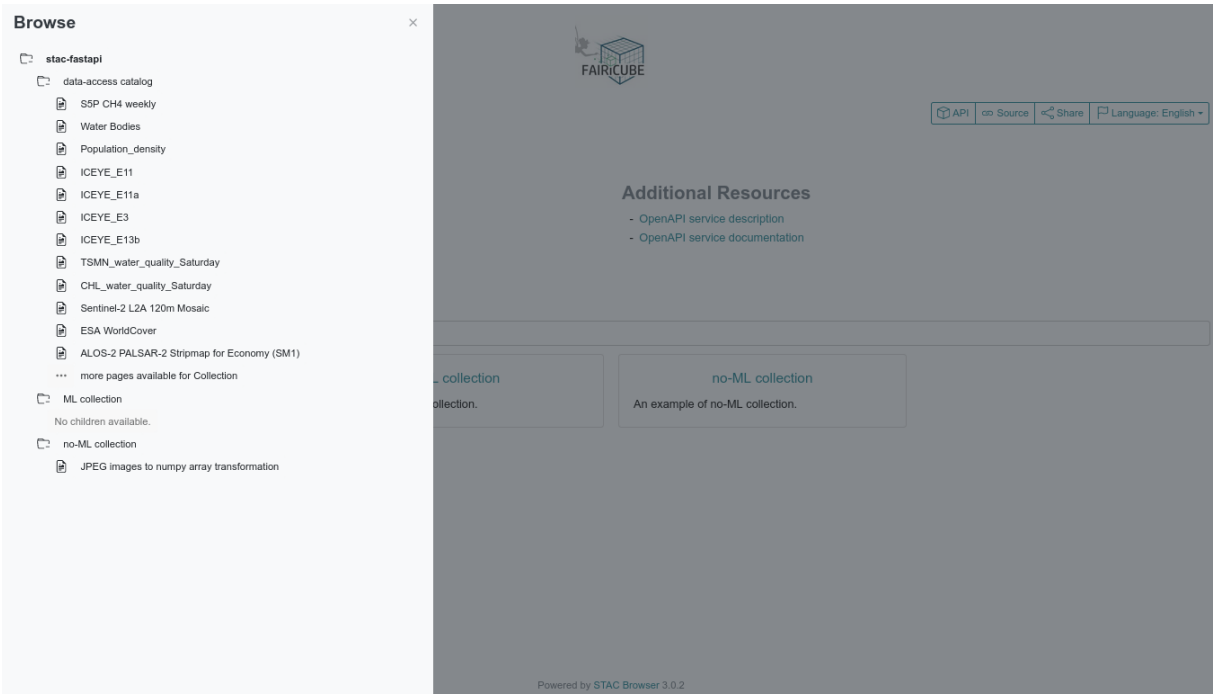

Figure 29: Browse Interface Feature of the STAC Browser

5 Services & Storage

The FAIRiCUBE Hub architecture focusing on services and storage is detailed in Figure 8.

Local datasets and models are either stored in cloud Object Storage in provided buckets or in the provided rasdaman storage. Notebooks are mostly stored on the provided workspaces but can also be shared for example via Object Storage. External datasets are federated from the Euro Data Cube or the EarthServer Federation. Euro Data Cube relies on cloud resources from Amazon Web Services (AWS), the Open Telekom Cloud (OTC) as used in the forthcoming Copernicus Data Space Ecosystem (CDSE), or the DIASes, whereas the EarthServer Federation uses INSPIRE or the DIASes.

The local datasets are available within the FAIRiCUBE Hub via the FAIRiCUBE Services/Apps providing several access mechanisms based on where they are registered or ingested. In summary the data residing in rasdaman storage is accessible via WMS/WMTS, WCS, WCPS, openOE, or 3rd party code provided by rasdaman whereas the data in Object Storage can be accessed via the Process API, Batch API, Statistics API, openEO, WMS provided by Sentinel Hub or directly in xarray, s3, etc. Additionally, vector data can be made available via GeoDB providing a PostgREST API or openEO.

These services are exploited by processes written as Jupyter notebooks or via tools like MLflow, DVC, TensorBoard, etc. running in provided workspaces in the FAIRiCUBE Lab.

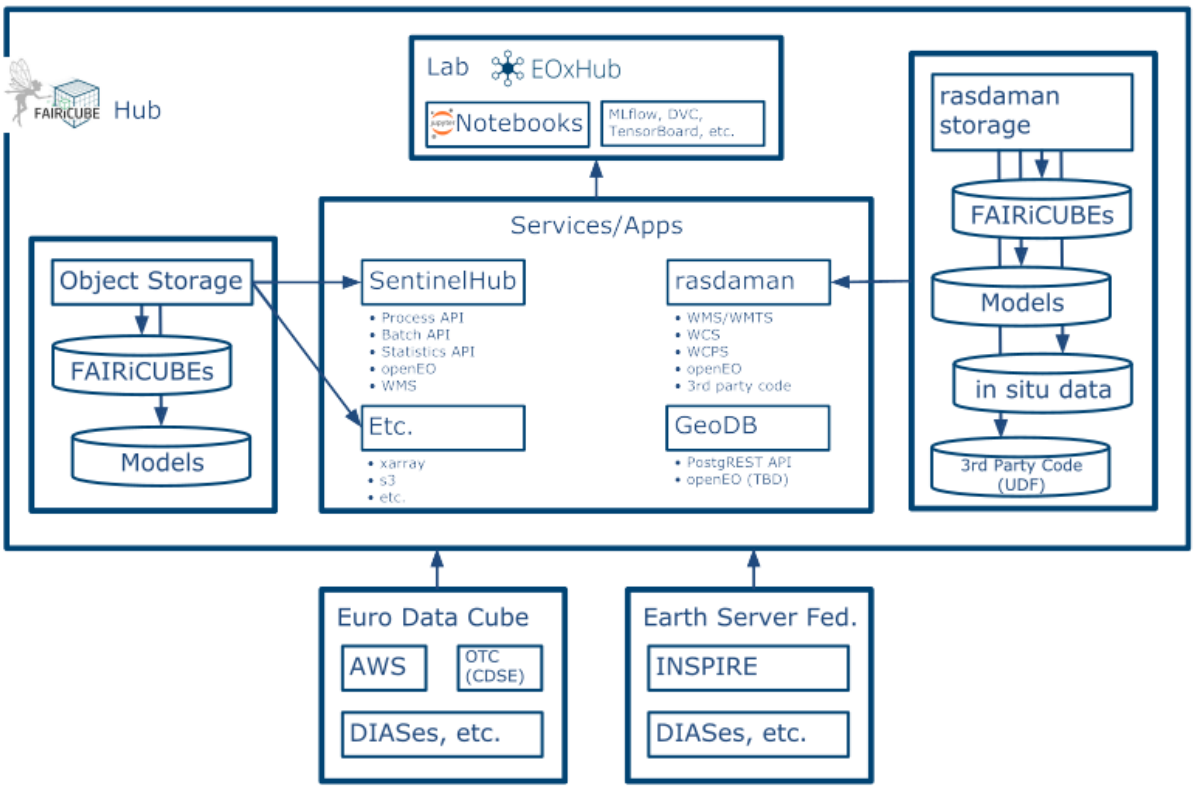

Figure 30: FAIRiCUBE Hub Architecture - Services & Storage

5.1 Euro Data Cube, Sentinel Hub, GeoDB, etc.

The core app deployed in each FAIRiCUBE Lab workspace is a managed JupyterLab. This allows to interactively execute Jupyter notebooks close to the data. Jupyter notebooks can not only be executed interactively in the browser, for example to develop an algorithm, but also in a headless way, i.e., without a graphical user interface, using a REST API provided by pygeoapi (for now please see

https://eurodatacube.com/documentation/headless-notebook-execution for details, note that this documentation will be made available specifically for FAIRiCUBE at the FAIRiCUBE Lab URL).

The concept behind these JupyterLab-based open-source workspaces for earth & data science, powered by EOxHub, has evolved during the Euro Data Cube project. It is now generalised and deployed for the FAIRiCUBE project. Figure 15 shows the deployment of EOxHub in the frame of the Euro Data Cube.

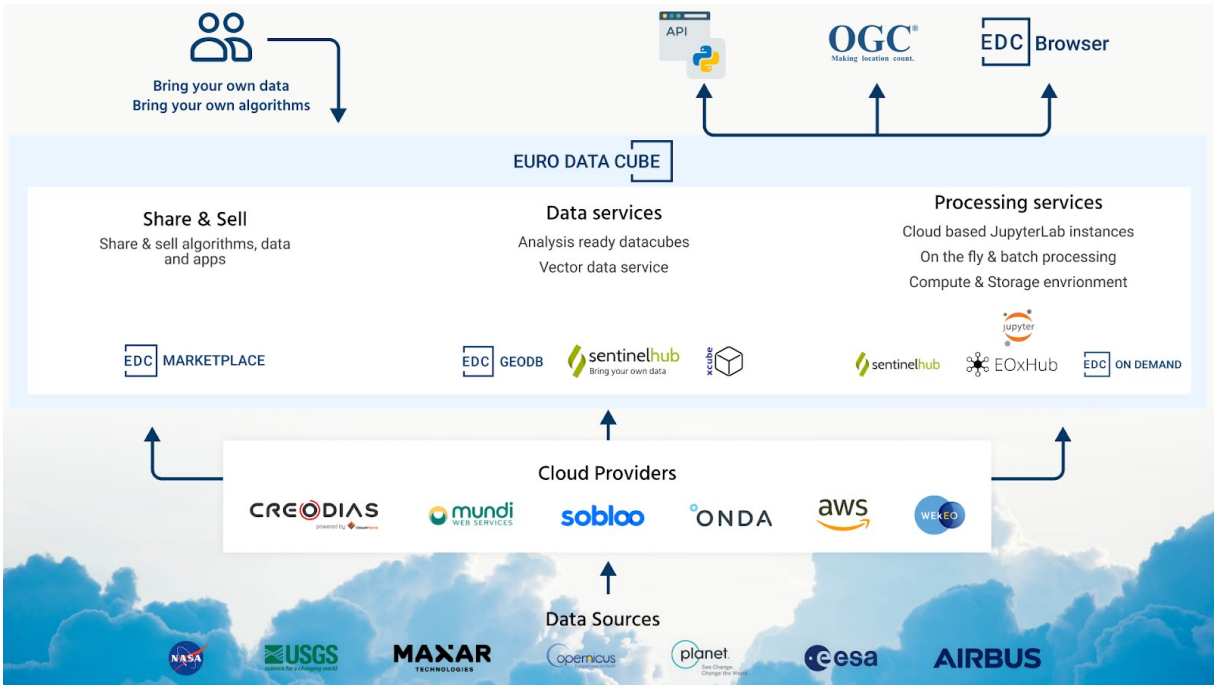

Figure 31: EOxHub as deployed for Euro Data Cube

Figure 10 shows different data workflows that can be realized in the provided workspaces relying on the integrated services. The top flow shows EO data exploitation in a client data processing engine. The user's web browser relying on JavaScript (JS) connects directly to Cloud Optimized GeoTIFFs (COGs) provided via object storage.

The next flow, in the figure, shows on-the-fly data cube access via Sentinel Hub followed by the mass-processing service Sentinel Hub Batch, and lastly pre-generated data cubes with xcube (please see https://eurodatacube.com/documentation/choose-your-workflow for more details about these services, note that this documentation will be made available specifically for FAIRiCUBE at the FAIRiCUBE Lab URL). These APIs can either be consumed directly by external clients, via the EOxHub Workspace, or via an App, for example, one providing a specific OGC API (WCS, DAPA, etc.) on top of the available services.

The FAIRiCUBE project will integrate further services as required. Section **Error! Reference source not found.** provides more details on this service integration.

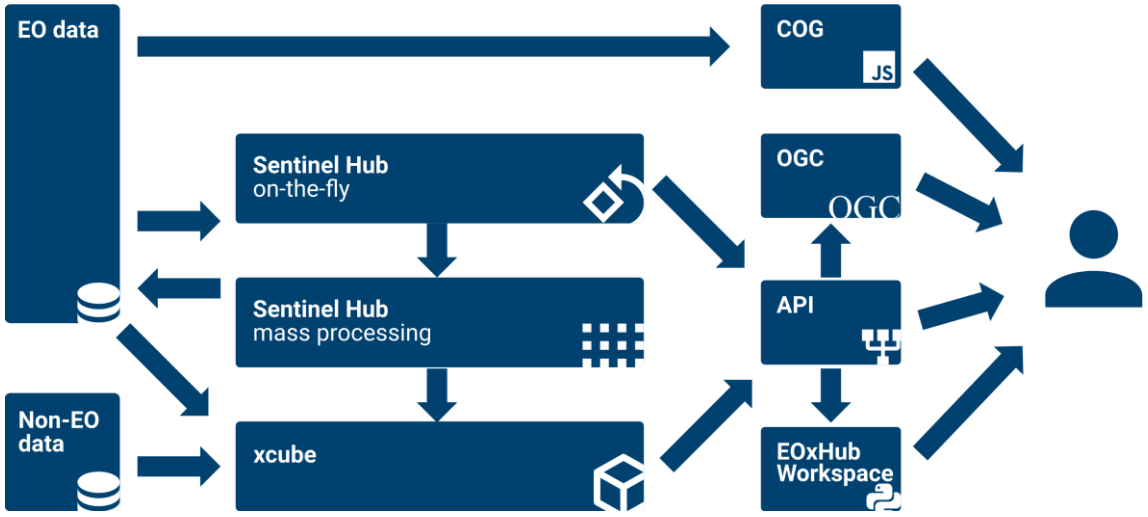

Figure 32: Data Workflows

5.2 EarthServer Federation and rasdaman

Actionable datacubes™ have been pioneered by rasdaman ("raster data manager") early on[16] and which at the same time has coined the domain of Array Databases aiming at high-quality management and analytics on massive multi-dimensional datacubes. In recent years, various alternatives have been developed, usually extending the xarray python library into some service[17]. This is in contrast to rasdaman, where the complete database system has been implemented, with every component hand-optimized for performance and scalability. A deep comparison of 19 datacube engines[18] has been published by Research Data Alliance.

Distinguished service capabilities include zero-coding datacube queries, transaction support, automatic data management, server-side query optimization (including parallelization and federation), versatile access control, and several more relevant features.

5.2.1 rasdaman Architecture

The rasdaman architecture resembles a standard DBMS architecture implemented in C++, with every component specifically implemented and hand-optimized for tiled arrays (Figure 3). These components include client APIs (such as C++), query parsing, optimizing, and execution, storage and index management, and several more. Queries are translated into logical trees, then to physical trees, and finally to executable code.

The rasdaman Array DBMS is domain agnostic and also has been used, e.g., in human brain imaging and cosmology. Geo semantics is added through a layer which understands space and time and, hence, also regular and irregular space-time grids. Access to this geo semantics layer is through OGC WMS, WMTS, WCS, and WCPS. Further, data can be offered compliant with INSPIRE requirements[19].

16 P. Baumann: Language Support for Raster Image Manipulation in Databases. Proc. Int. Workshop on Graphics Modeling, Visualization in Science & Technology, Darmstadt/Germany, April 13 - 14, 1992, Springer 1993, pp. 236 - 245
17 In fact, xarray and xcube and further python libraries can well act as clients to rasdaman using it as the Big Data backend.
18 https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00399-2

19 http://inspire-wcs.eu, the official INSPIRE-WCS Good Practice and only full implementation of INSPIRE-WCS.

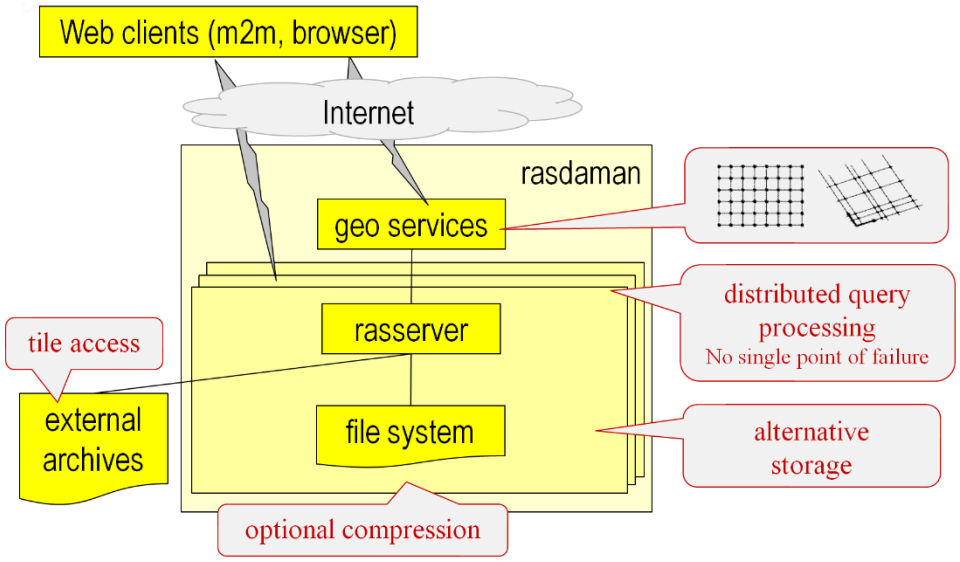

Figure 33: rasdaman high-level architecture

The declarative array query language *rasql*, which in the meantime has been adopted into ISO SQL[20], offers a rich spectrum of declarative operators streamlined for array access, extraction, aggregation, analytics, and fusion.

Storage of arrays relies on partitioning into sub-arrays called tiles. Tiles, which form the unit of storage access, can be stored in rasdaman's own storage manager or in a pre-existing archive; in the latter case, data do not get copied, but registered and evaluated in-situ. This was in fact a sine-qua-non for unleashing Peta-scale archives. Based on a rigorous analysis of possible tiling patterns[21], several strategies have been devised which are available to the administrator as a tuning parameter[22] in addition to storage location, compression, etc.

All this remains completely transparent to the user, as queries describe the result, not the algorithm. Each incoming query gets optimized individually through a series of highly effective techniques, including over 150 tensor algebra rules, cost-based optimization, just-in-time compilation into machine code, parallelization, and distributed processing with peer-to-peer federation. Publicly available benchmarks underline that this full-stack approach has remarkable performance advantages[23]; for example, the query splitting mechanism, evaluated by distributing single queries over more than 1,000 Amazon nodes[24], allows a substantially more effective parallel processing than, say, MapReduce type parallelization where all processors execute the same code.

5.2.2 Datacube Access

Thanks to the support for open standards, a wide spectrum of 3rd-party clients can tap into rasdaman datacubes, ranging from simple map visualization (ex. Leaflet, OpenLayers) over Virtual Globes (ex: NASA WorldWind, Microsoft Cesium) and Web GIS (ex: QGIS, ArcGIS) to high-end analytics (ex: R, python). We detail some basic ways to access datacubes.

20 ISO: 9075-15:2019 SQL/MDA (Multi-Dimensional Arrays). iso.org/standard/67382.html

21 P. Furtado, P. Baumann: Storage of Multidimensional Arrays based on Arbitrary Tiling. Proc. Intl. Conf. on Data Engineering (ICDE), March 23-26, 1999, Sydney, Australia

22 P. Baumann, S. Feyzabadi, C. Jucovschi: Putting Pixels in Place: A Storage Layout Language for Sci-entific Data. Proc. IEEE ICDM Workshop on Spatial and Spatiotemporal Data Mining (SSTDM), 2010, Sydney, Australia, pp. 194 – 201

23 H. Kristen: Comparison of Rasdaman CE & AGDCv2. gitlab.inf.unibz.it/SInCohMap/datacubes/-/blob/master/datacube_comparison/datacube_comparison.md

24 A. Dumitru, V. Merticariu, P. Baumann: Exploring Cloud Opportunities from an Array Database Per-spect¬ive. Proc ACM SIGMOD Workshop on Data Ana-lytics in the Cloud (DanaC), 2014, Snowbird, USA

Visualization. The OGC Web Map Service (WMS) allows the visualization of stacks of 2D map layers. Datacubes containing more dimensions than just x/y can be mapped to 2D through temporal selection using the TIME parameter and vertical selection with the HEIGHT parameter. In summary, layers can be retrieved from stored data or derived dynamically ("virtual layers").

OGC Web Map Tile Service (WMTS) is a related service with reduced functionality, access to predefined tiles via some given identifier. WMTS is supported by rasdaman as well.

Extraction. OGC Web Coverage Service (WCS) is a modular suite for coverage handling. WCS-Core supports subseting and output format choice, where extensions add bespoke functionality. One of these extensions is the datacube language WCPS (see details below), which incorporates all capabilities of the WCS Core and its extensions, but further allows building expressions of arbitrary complexity. Therefore, WCPS generally is recommended over WCS.

Analysis. The OGC Web Coverage Processing Service (WCPS) defines a datacube analytics language that is aware of space and time (and hence, regular and irregular grids). Queries of arbitrary complexity can be expressed and evaluated server-side; in the case of rasdaman, queries undergo highly effective optimization prior to execution. Expressive power includes all Tomlin function categories (local, focal, zonal, global) and reaches up to, for example, the Discrete Fourier Transform. Rasdaman also supports the option of integrating custom code; code can be developed, integrating specialized libraries as required, and be linked dynamically into the server and invoked via the query language as so-called User-Defined Functions (UDFs). The rasdaman engine automatically orchestrates built-in and UDFs during query optimization and evaluation. Additionally, native Machine Learning support is going to be provided in FAIRiCUBE.

5.2.3 Datacube Import and Maintenance

An intelligent ETL (Extract, Transform, Load) suite allows simple setup of automatic import pipelines, building on and enhancing the OGC WCS-T standard. For every datacube, a small configuration file governs import and piecewise growth of its datacube; most parameters are recognized automatically, and all common formats are supported – only missing information must be provided, resulting in a dozen of lines of configuration. Import can be configured to mean copying into the database ("ingest") or just registering the input files for further query processing directly on these files ("in situ"). Any kind of pre-processing can be built into such import pipelines. Also, "virtual datacubes" can be built from existing, even heterogeneous datacubes, such as the integration of Sentinel-2 data in different UTM zone grid tiles into a single Sentinel-2 datacube. Pre-confectioned datacube configuration files for a wide range of common situations are available off-the-shelf.

5.2.4 EarthServer Federation

EarthServer is the currently only location-transparent federation offering large-scale spatio-temporal datacubes. Large-scale data centres such as several DIASs, Julich Research Center, Universita di Napoli, Alfred-Wegener-Institute, and the National Supercomputing Center Taiwan (to name but a few) contribute datacubes of Copernicus data (all Sentinel data, CLMS, etc.), climate/weather datacubes, thematic data on land governance, maritime topics, and more.

Participation of data providers in EarthServer is free and open, based on a transparent and democratic governance.

In FAIRiCUBE, EarthServer data are made available to the user partners. In the future, the FAIRiCUBE specific assets can get integrated into the federation, thereby enhancing the overall critical mass of diverse data offered.

6 Hub & URLs

Figure 10 provides a view on the FAIRiCUBE Hub architecture focusing on the FAIRiCUBE Lab and URLs where the various FAIRiCUBE components are or will be available. The main landing page of FAIRiCUBE is https://fairicube.eu providing general project information and links to further components.

The subdomain https://hub.fairicube.eu will provide the main landing page for the FAIRiCUBE Hub, whereas https://eoxhub.fairicube.eu already provides the entry page to the EOxHub deployed at FAIRiCUBE Lab. Apps deployed in user workspaces will be made available under URLs following the pattern https://<app>.<uuid>.eoxhub.fairicube.eu.

An initial version of the static FAIRiCUBE catalog is available at https://catalog.fairicube.eu. A dynamic version of the FAIRiCUBE catalog (https://catalog.eoxhub.fairicube.eu/) is currently under development and will replace the static one. It is based on the STAC-API which provides additional features such as an extended search functionality. Though the STAC-API itself is still under development, it already provides a good basis and can be extended based on the FAIRiCUBE needs.

The datacube part by rasdaman has been set up under the URL https://fairicube.rasdaman.com (in future: https://rasdaman.fairicube.eu) and provides a landing page and an API endpoint.

Figure 10 also shows the FAIRiCUBE Lab managed JupyterLab and its integration with surrounding services currently available like rasdaman, Sentinel Hub, xcube, or GeoDB. Further services can be integrated in the course of the project execution as requested by Use Cases, for example to connect to the rasdaman deployment for FAIRiCUBE.

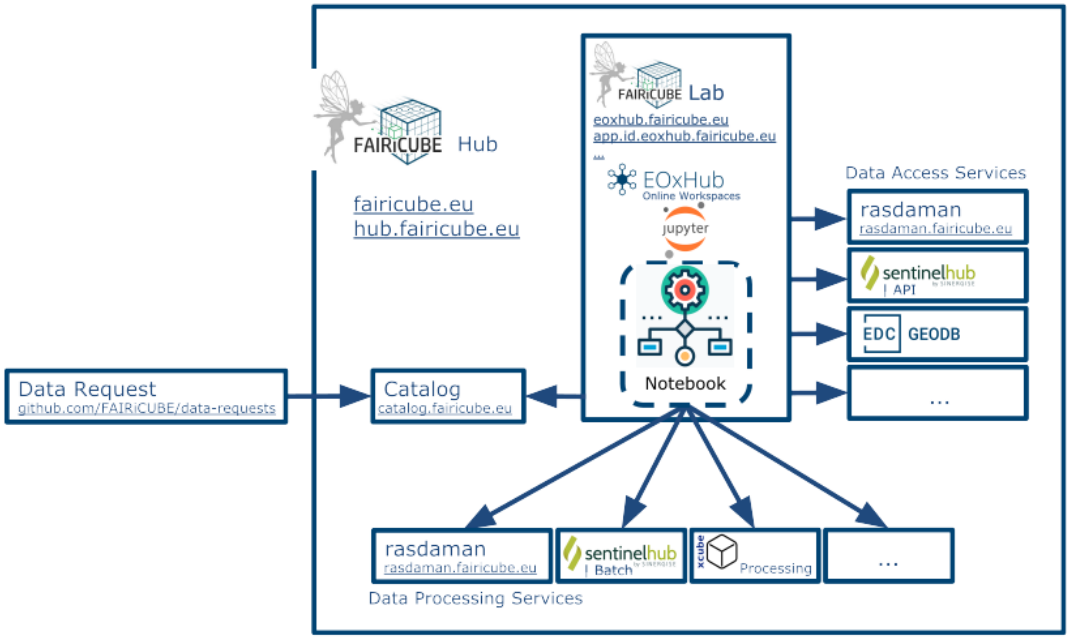

Figure 34: FAIRiCUBE Hub Architecture - Hub & URLs

Once all required data has been put in place, analysis and machine learning can commence. In order to enable easy access to existing processing modules together with available data, we utilize EOxHub (https://hub.eox.at/), a platform and workflow management runtime for Earth Observation services and apps developed and maintained by the project partner EOX. EOxHub is branded to provide the

FAIRiCUBE Lab and deployed on a kubernetes[25] cluster. Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management.

The FAIRiCUBE Lab is centred on the usage of Jupyter Notebooks, allowing to execute them close to the data for simple exchange and sharing of processing modules. The notebooks can either be executed interactively using a managed JupyterLab environment, for example to develop an algorithm, or in a headless way using a REST API meaning without a graphical user interface. The available data can be accessed in these Jupyter notebooks via different means depending on what is best suited for the use case at hand and the skill level of the user. The available options span from direct object storage access via the xcube[26] or xarray[27] Python libraries to the Process API of Sentinel Hub and Open Geospatial Consortium (OGC) defined interfaces like the Web Coverage Service (WCS) and Web Coverage Processing Service (WCPS). Jupyter notebooks can further utilize libraries like dask[28] to parallelize and scale processing jobs.

The FAIRiCUBE Lab provides Cloud Workspaces through the workflow management runtime for docker containers and relying on object storage to persist data based on EOxHub as shown in Figure 12. It offers science teams, projects, communities, and other diverse stakeholders a cloud footprint, i.e., usage of resources of IT cloud providers, with a subscription – a so-called "Workspace as a Service".

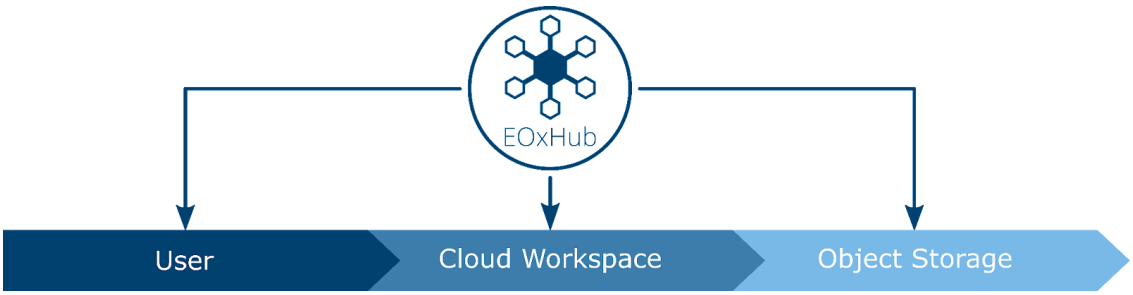

Figure 35: Cloud Workspaces

Figure 17 shows the schematic FAIRiCUBE Lab Architecture based on EOxHub and Apps and Services made available within.

The available Apps reach from the central JupyterLab via ones providing specific OGC APIs to very specialized ones serving individual use cases. Access can either be granted only to the user themself, to a group of users, or access is made open to anybody.

Services on the other hand are the integration of external operationally available services needing a subscription by the user and are ready to be used. Examples of such services are Sentinel Hub[29] operated by Sinergise or geoDB[30] operated by Brockmann Consult.

The App/Service Developer pushes the App/Service software to the code management repository where an automatic CI/CD (Continuous Integration/Continuous Deployment) pipeline tests, builds, packages, and publishes the App/Service as Docker image after which it is registered at the Marketplace. The App/Service Consumer discovers the App/Service and requests or triggers the deployment of the App/Service to their workspace to be run on the Cloud Infrastructure. The App/Service is now available to be used by the Consumer within the resources provided in their workspace subscription.

25 https://kubernetes.io

26 https://xcube.readthedocs.io

27https://xarray.dev

28https://www.dask.org

29 https://www.eurodatacube.com/marketplace/services/edc_sentinel_hub

30 https://www.eurodatacube.com/marketplace/services/edc_geodb

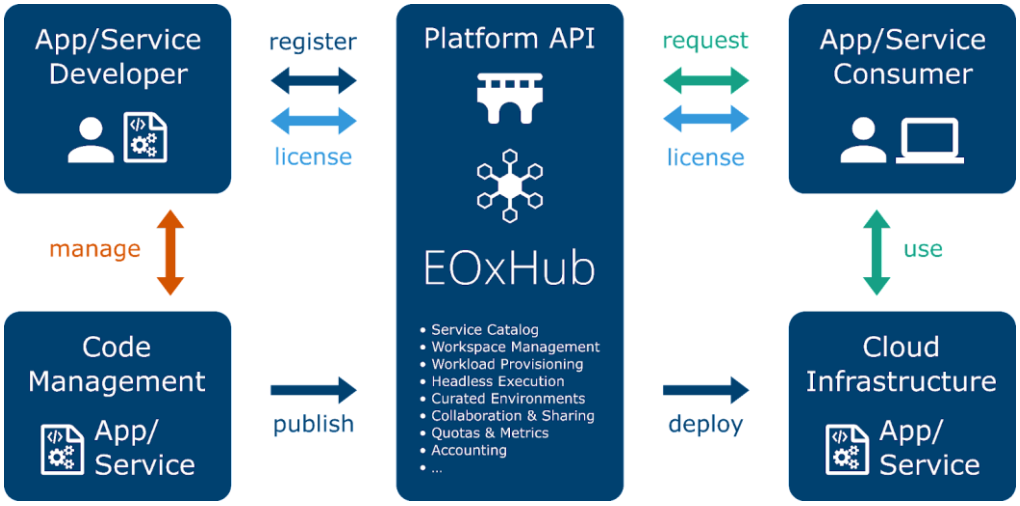

Figure 36: Schematic FAIRiCUBE Lab Architecture

7 Sharing of Notebooks, Algorithms, Services, and Apps

This section provides an overview on how to share notebooks and algorithms within the FAIRiCUBE Hub as well as how to onboard services and apps in the FAIRiCUBE Lab.

Until the upcoming deployment of the FAIRiCUBE components, the available functionality is described in this document using links to the Euro Data Cube documentation. Note that this documentation will be made available specifically for FAIRiCUBE at the FAIRiCUBE URL.

7.1 Notebook and Algorithm Sharing

Jupyter notebooks to be executed in the FAIRiCUBE Lab are the main mechanism to package processes and algorithms devel of Notebooks, Algorithms, Services, and Appsoped by the various teams and use cases as shown for example in Figures 30 and 34.

The metadata requirements for processes and algorithms are described in the deliverable D4.3 "Public Listing (Catalog) of FAIRiCUBE processing/analysis resources".

7.1.1 FAIRiCUBE Lab - EOxHub

There are multiple ways to share notebooks and algorithms in EOxHub. This depends on the audience they shall be shared with.

The most straightforward way to share is to simply store a notebook in the shared storage space. A more advanced mechanism would be to use a common Git repository or any other means the team agrees to.

On the FAIRiCUBE Lab, there is a dedicated wizard available for sharing a notebook publicly under the "My contributions" section (for details see https://eurodatacube.com/documentation/notebook-contributions). Notebooks shared this way will be made available on the marketplace offered by EOxHub (see the Euro Data Cube marketplace at https://eurodatacube.com/notebooks for reference).

There is another option to make an algorithm directly available to users via the Insights On Demand functionality (https://eurodatacube.com/documentation/checking_out_data_on_demand). This way, users can run an algorithm for their specific parameters directly on the platform without any need to deploy it in their workspace.

From the algorithm provider side this is called the "Bring Your Own Algorithm" functionality (https://eurodatacube.com/documentation/offer_algorithms_for_on_demand_data_generation).

Figure 9 shows the steps involved in the "Bring Your Own Algorithm" function. The algorithm *consumer* can, after obtaining credits, directly order Insights On Demand. The algorithm provider develops and offers an algorithm on the platform including signing a *provider* agreement with the platform operator and sending invoices to the platform operator as agreed. The platform operator is responsible to operate the platform, manage credits and orders, and to send usage reports to operators.

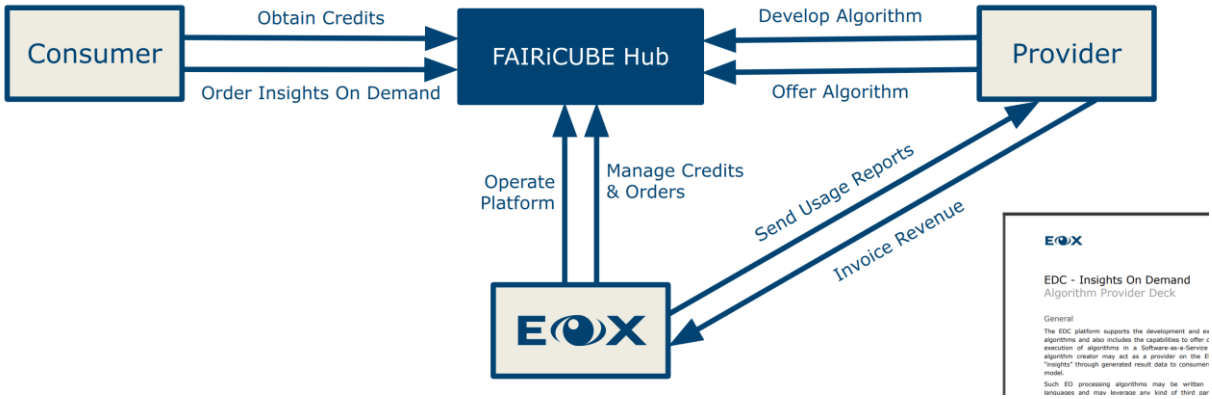

Figure 37: Bring Your Own Algorithm

7.1.2 FAIRiCUBE Services/Apps - rasdaman

Datacube wrangling is likewise possible in rasdaman, both client and server side. Notebooks can easily access, extract and process datacubes with only three lines of code for opening a connection, defining the query, and the sending/receiving request. Manifold examples and tutorials exist. Additionally, UDFs allow integration of any external code into the server, thereby allowing to share such code with other users.

7.2 Service and App Onboarding

The EOxHub powered FAIRiCUBE Lab allows to easily share EO specific workloads with other users via the marketplace. In order to do so, the source code of an app must be available in a GitHub repository which contains a Dockerfile. EOxHub will automatically build a docker image and publish it. There are a number of sample apps available that can serve as a template for further apps.

The application can then be deployed by other users, where it will be available under a respective custom URL to receive network traffic. For more details, please refer to https://eurodatacube.com/documentation/app-contributions with the usual disclaimer that this documentation will be made available specifically for FAIRiCUBE at the FAIRiCUBE Lab URL.

The final selling option offered on the FAIRiCUBE Lab is to sell a subscription to an API service. An external, operationally available service can be added to the marketplace for users to subscribe to. EOxHub will take care of:

- user access and subscription management,
- automated service brokering and user credentials within our kubernetes cluster,
- seamless integration and credential injection in a user's Jupyter notebooks and Apps,
- automated user billing and invoicing, and
- revenue sharing.

For more details, please refer to https://eurodatacube.com/documentation/sell-service-subscription for now.

New services will be integrated in the course of the project execution as requested by Use Cases for example to connect to the rasdaman deployment for FAIRiCUBE.

In the course of the project, the rasdaman capabilities will be embedded seamlessly in the FAIRiCUBE Lab so that users have a wide range of options available.

8 Interfaces and User interactions

This chapter describes the Interfaces between the different components of the FAIRiCUBE Hub from a User's viewpoint. When applicable the respective protocols used are also provided. For existing, especially external systems which may be accessed by the FAIRiCUBE Hub, links to the respective description (e.g., APIs) are, if possible, provided.

8.1 User – Catalog

FAIRiCUBE operates 2 Catalogs, each hosted in a separate GitHub Repository which are accessible at https://github.com/FAIRiCUBE/data-requests, allowing for new data to be made available within FAIRiCUBE, and https://github.com/FAIRiCUBE/resource-metadata, which manages information for processing/analysis resources.

Note: Currently the dynamic catalog is reachable at, https://catalog.eoxhub.fairicube.eu but this will be moved to https://catalog.fairicube.eu once stable enough to replace the current static catalog.

These GitHub Repositories are exposed via a STAC-API and can be accessed via the single endpoint at https://stacapi.eoxhub.fairicube.eu/. As a frontend to the exposed STAC-API implemented using the software STAC-FastAPI an instance of the software STAC Browser acts as Web-GUI at https://catalog.eoxhub.fairicube.eu allowing detailed interactive searches in the offered FAIRiCUBE Catalog.

The following abbreviation is used: IF = Interface

- IF to STAC Browser:
  - Web GUI based on http/s protocol
- IF from STAC Browser to Catalog
  - STAC-API (standard) based on REST http/s protocol
- IF to GitHub Issue:
  - based on http/s REST protocol
- IF to GitHub Project (https://github.com/orgs/FAIRiCUBE/projects/)
  - based on http/s REST protocol

8.2 Data Metadata & Process Metadata interactions

3 scenarios: data exists in Catalog / data not available / data requested but not yet published

- User checks catalog – data is found in Catalog
- User checks catalog – data is not found in Catalog - additional check of Data Access GitHub project, data is not found - Request for new Data via Data Request Issue template – Operator checks – GitHub Issue – static json file are exposed via GitHub pages
- User checks catalog – data is not found in Catalog - additional check of Data Access GitHub project, data is found in project – user can convert project draft into GitHub Issue – static json file are exposed via GitHub pages

See also the procedure provided in Figure 22.

8.3 User – Notebooks

- User access Jupyter Notebooks via Web-GUI. Notebooks provide preconfigured tools which allow easy access and usage of the resources provided by FAIRiCUBE Hub.

    - IF via Browser
- based on http/s
    - Headless execution of Notebooks: for long running processing jobs and possibility for headless execution i.e., running a job non-interactively (asynchronous). The status of these jobs can be queried.
- IF via HTTP-API request
    - based on http/s e.g., using cURL

8.4 User Notebook – Catalog

- IF to STAC Catalog
    - STAC-API (standard) based on REST http/s protocol

8.5 User Notebook – Data Access interactions

- IF to local data
    - Filesystem (mounted)
- IF to Object Storage
    - S3 or SWIFT
- IF to rasdaman
    - REST, WCPS
- IF to EDC SentinelHub APIs
    - HTTP-APIs: Process, WMS
- IF to GeoDB (EuroDatcube)
    - postSQL, postGIS, postgREST
- IF to other services (e.g., EarthServer, DIASes, AWS, etc.)
    - HTTP-API REST

8.6 User Notebook – Data Processing interactions

- IF to rasdaman
    - WCPS, REST
- IF to EDC SentinelHub APIs
    - HTTP-APIs, Process, Statistics, Batch processing
- IF to OpenEO
    - HTTP/S
- IF to other services (e.g., Euro Data Cube, EarthServer, DIASes, AWS, etc.)
    - OGC Application Packages (e.g., CWL, WPS, etc)

8.7 User Notebook – Sharing

- IF sharing a Notebook within a WP/within FAIRiCUBE
    - pull/push to GitHub via https, ssh

8.8 Download Notebooks

- Download interface for shared processing routines; download and then run on local laptop – e.g., for preprocessing steps, later upload to the FAIRiCUBE Hub/Cube
- IF to download Notebooks
  - based on http/s, REST

9 Community Collaboration Platform

The aim of this task is to develop an information exchange community collaboration platform, which will serve as contact points for experts and scientists using datacubes for environmental studies. All partners, particularly those contributing a use case, provide requirements for the Community collaboration platform, to best support their ML workflows and processes. In addition, requirements enabling collaboration, both within and across project teams, are collected and the functionality provided is designed to support these needs.

Based on the collected requirements, new apps are developed and offered for deployment in users' workspaces on the FAIRiCUBE HUB. These apps are based on existing Open-Source software, like MLflow, as much as possible. The apps themselves will be released as Open-Source on GitHub again. Care will be taken to assure close integration of ML capabilities with the data resources being provided. The community collaboration platform focuses on the general technological information and knowhow associated with the setup, usage and processing of datasets on the FAIRiCUBE Hub and the usage of ML-Tools applied to these datasets.

9.1 Implementation

The community collaboration platform is setup as a collection of markup documents, based on the MkDoc definition, and collected in the FAIRiCUBE GitHub repository. This sub-repository is structured according to the requirements provided by "*Read The Docs*"[31], which acts as the target platform for publication of the community collaboration platform. It allows project internal as well as external users to submitt their information, experiences and code to the platform and therefore share it with others.

"Read The Docs" features the automatic deployment of the documentation, and examples e.g. present as Jupyter Notebooks, with every change of the documentaion source, via the usage of a Webhook. Therefore there is no need for an extra maintenance effort to manage the documentation. Changes are submitted to the GitHub repository, and if accepted by te owner, are then immediately present in the community collaboration platform.

The community collaboration platform GitHub repository is accessible at: https://github.com/FAIRiCUBE/collaboration-platform.
The *Read The Docs"* based platform is accessibe at: https://fairicube.readthedocs.io. A screnshot of the landing page is shown below in Figure 38.

31 https://about.readthedocs.com/

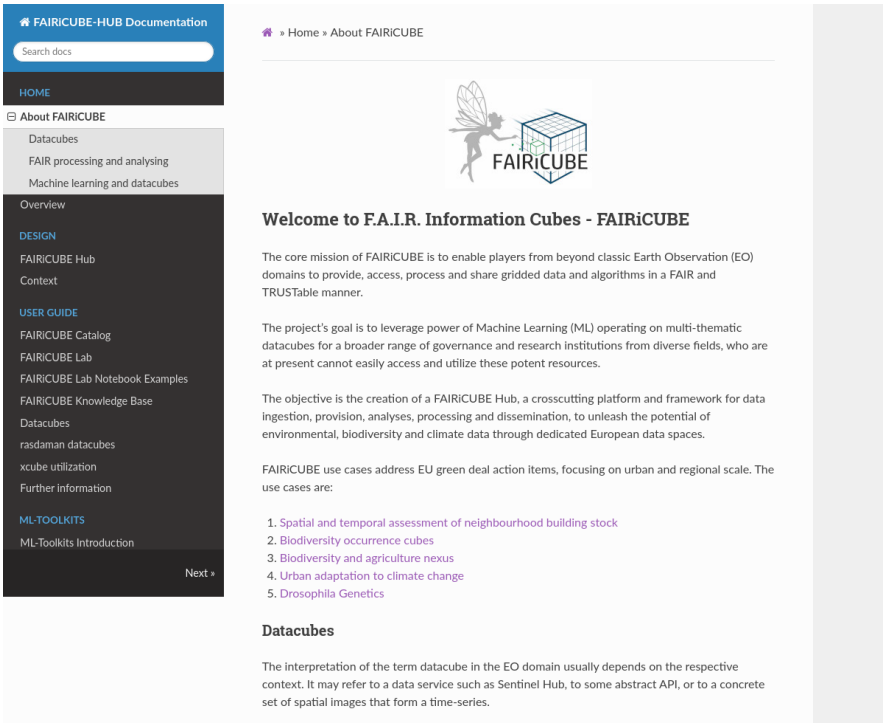

Figure 38: Landing page of the Community collaboration platform

10 Knowledge Base

The Knowledge Base is a toolkit to enable appropriate knowledge of how to apply algorithms and ML techniques to solve UC's and similar demands.

The core task of the Knowledge Base (KB) is to provide to the community a set of tools, documents, algorithms, code, tips and tricks, mistakes to avoid, examples of use and so on.

The architecture of the KB is composed by a web-application, a database and multiple data sources.

The web-application is coded using Python and the Django web-framework. It consists in a set of web pages with static content and an interactive query tool.

The pages:

- "Use Cases" summarizes main aims and results of the use cases and provides access to more in-depth descriptions and specific resources.
- "Query Tool" is the engine of the KB. It allows to query KB resources based on predefined or customized queries.
- "Self-training Library" links to a number of training resources freely available from the web and/or from the project repositories to support understanding and reuse of the project outcomes and resources.
- "Tips & Tricks" shares solutions adopted / workarounds to overcome various challenges faced in the UC lifetime.

The *"Metadata Catalog"* and the *"GitHub Project"* items in the Menu directly interact respectively with the Metadata Catalog and with the GitHub repositories of FAIRiCUBE.

GitHub, websites and content created by FAIRiCUBE are used as sources of the data used in Knowledge Base.

A PostgreSQL database is the engine of the query tool and contains thein formation, originating from the metadata of the resources, on which to make SQL queries. The database does not contain all the metadata fields but only those utilized to filter the various metadata records, where each record corresponds to a resource. The result of a query is a table containing, for each resource found, the Name, Description and link to the resource in the Metadata Catalog. This last is then used for a complete visualization of the resource.

The procedures of reading data on GitHub and ingesting it into this database are done automatically by a specific Python function.

In particular, the ingestion of resource metadata into the PostgreSQL is executed through several steps

- identification of metadata in the 'resource-metadata' GitHub repository: issues representing resource metadata (I.e., created through the resource metadata request form) are identified through the label 'a/p metadata' and become input to step 2.
- Creation of key-value dictionaries: A Python program creates appropriate key-value dictionaries, associating values to the corresponding metadata fields. These are then passed to a builder, specific to the resource type, which creates the metadata in STAC-JSON files. Specifically, if the file is not already present in the metadata folders it is inserted, otherwise the processing of the next issue is done.
- Insertion of the record in the DB: if the file is not present in the DB (the resource ID is used as the identifier) it is inserted using an appropriately selected list of fields.

The phase of checking if data are present or not in the folders and DB is necessary in order to write and load only new files, thus saving resources and avoiding overwriting all files at each execution.

- Process automation: The entire process of creating STAC metadata and ingesting it into the PostgreSQL database is fully automated using a Python script (executes all steps above sequentially).

The Query Tool is organized into two parts:

- "Predefined Queries" allows to select from a list of common UC resources queries that have been prepared to quickly get resources based on most widely utilized search criteria.
- "Custom Queries" allows users to build custom queries by selecting one or more parameters to filter on and/or by specifying keywords

The self-training library contains a set of links to web pages and project resources, appropriately selected and organised into categories, with the aim of provide to the user a basic background on FAIRiCUBE Knowledge Base topics.

The Tips & Tricks section contains a number of solutions to problems and questions encountered during the project. This section gives the user the opportunity to quickly solve known problems and also avoid remaking same mistakes. Also in this case, the data sources are web pages and project resources.

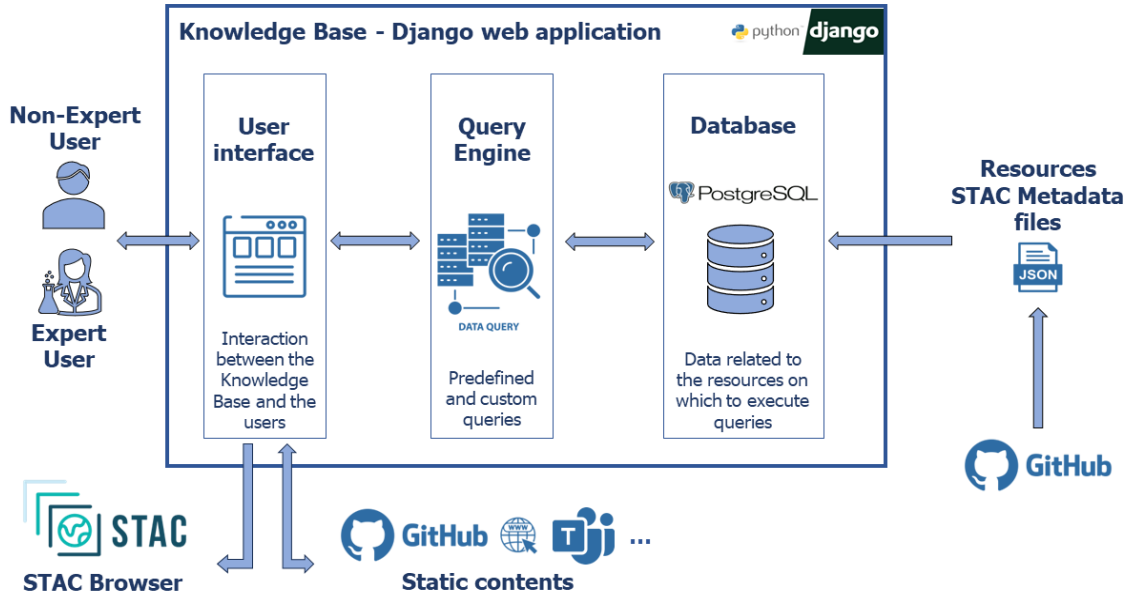

Figure 39: Knowledge Base - Application Architecture

11 Deployment and Operations Strategy

This section details the strategy for deployment and operations of the various components of the FAIRiCUBE Hub as shown for example in Figures 30 and 34

11.1 FAIRiCUBE Catalog

The current static FAIRiCUBE Catalog is deployed on GitHub pages and available at https://catalog.fairicube.eu, (see **Error! Reference source not found.** in section **Error! Reference source not found.**). The contents of this FAIRiCUBE Catalog are maintained in a GitHub repository[32] as yaml files plus accompanying details. The yaml files stored on GitHub are transposed to STAC json files during an automated build step[33]. These static json files are exposed via GitHub pages together with this basic catalog web interface.

The entry point for the static json files is https://catalog.fairicube.eu/stac/index.json providing the STAC Catalog listing and linking to all the available data as individual STAC Collections (e.g., https://fairicube.github.io/catalog/stac/corine-land-cover.json) as detailed in the previous section.

This static catalog will be replaced with a rich web catalog based on STAC Browser which is connected to a STAC API endpoint implemented using STAC-FastAPI. This rich catalog is supporting searching by fields like dates, providing summaries of the available data, etc. A first version of this rich web catalog is available at https://catalog.eoxhub.fairicube.eu, but the the catalog will be transferred to https://catalog.fairicube.eu once he search functionality is implemented.

11.2 FAIRiCUBE Lab - EOxHub

From a deployment and operations point of view, the FAIRiCUBE Lab is a fully managed cloud environment providing scalable compute and (co-located) storage for diverse stakeholders ranging from scientists to decision makers.

Operations of an Exploitation Platform offering Big Data and Machine Learning usually involves substantial cost. It is a challenge to determine beforehand the expected consumption by AI-based Apps of cloud resources for processing and storage. The Marketplace concept therefore must include tools for parametrised cost estimation and quotations for the Consumer. When processing is underway informative and real-time cost accounting results are needed to avoid cost overruns.

It has been decided among the project partners to use the Frankfurt region of the Amazon Web Services (AWS) as the cloud infrastructure provider. The first deployment will be performed on AWS in a public cloud tenant owned by the project partner NILU.

The control plane will be bootstrapped, i.e., the initial cloud resources will be provisioned, and operated by the project partner EOX. Together with use case partners, it will be decided which resources shall be made available in the elastic and dynamically scaling worker plane to run science and other user workloads in their so-called workspace profiles. This decision is an important step, as it defines the incurred costs at runtime. Taking this decision together shall help the use case partners to fully understand the cost implications later. Different profiles have different per hour costs associated which is important to keep in mind when running workloads for example on CPU or GPU resources.

The deployment uses GitOps principles which are shown in Figure 40. Developers push code changes to a Git repository where the Continuous Integration (CI) pipeline automatically tests, builds, packages, and publishes the code to a docker image repository. The flux Automator can be used to update the

32 https://github.com/FAIRiCUBE/catalog

33 https://github.com/FAIRiCUBE/catalog/blob/main/.github/workflows/pages.yml

configuration to use new versions either via an update hint from the CI pipeline or manually by the developer.

For Continuous Deployment (CD) the operators push configuration changes to a Git repository which is deployed fully automatically in the kubernetes cluster by the orchestrator using the deploy service.

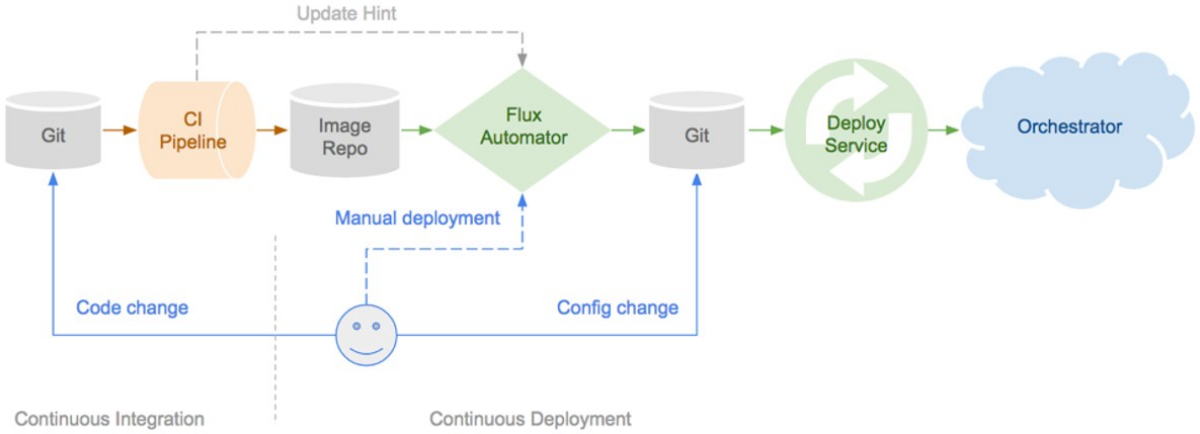

Figure 40: GitOps Principle

This way the deployment relies on declarative configurations to establish tailored workspace profiles for on-boarded users (GPU yes/no, #CPUs, total memory, storage types, conda environments, credentials, etc.). A GitHub repository will be used to store these configurations mainly of assignment of users to team(s) and the associated workspace profiles. Login to the FAIRiCUBE Lab will be primarily performed via GitHub as identity provider.

It is important to note that usage metrics are available on user, group, and cluster level. These metrics are collected using Prometheus and are visualised in Grafana dashboards.

The main advantages of following the GitOps principles are:

- Deployments can be fully automated via git commit (roll-backs via git revert, audit trail via git log)
- Security is largely improved as the Application and Service Developers don't need to access the cluster
- Full auditability and easy replication are achieved by having all state and state changes tracked in git

11.2.1 Principles for Operations

In summary, the main principles of operations are:

- Staff access to AWS services is only necessary for troubleshooting respectively for preparatory setup tasks and maintenance (e.g., EKS version upgrades) but not for daily business operations.
- For the initial setup a temporary bootstrap user with quite elevated permissions is needed to get the terraform based tooling in place.
- IAM users and assumed IAM roles are only leveraged by service accounts, not staff users. They are made accessible to EKS workloads as kubernetes secrets.
- User-facing services & applications are deployed and upgraded with GitOps concepts on EKS, leveraging the fluxcd ("flux") tooling following a pull approach (no push, i.e., no direct kubernetes access is needed).
- Staff access to EKS cluster is only necessary for troubleshooting. This can be managed on kubernetes namespace level via flux.

11.2.2 Deployment Services split into three Segments

The deployment services rely on Infrastructure as Code principles to streamline cloud resource creation and are split into three segments. This separation acknowledges the different set of people involved, frequency of change, and different environments (staging vs. production):

1 Roll out infrastructure for kubernetes based runtime environment
   - EKS (with configured autoscaling nodegroups) running in VPC
   - DynamoDB, S3 (only relevant for terraform)
   - EFS (backing workspace storage)
   - ECR (for all published containers)
   - Core components (helm charts) to deploy apps in kubernetes cluster

2 Roll out infrastructure for storage and access
   - S3
   - IAM (for service accounts, not for users)

3 Deploy user facing services & applications (EOxHub Workspace, Backend APIs, etc.) via Git

The first segment is automated via EOX tooling using terraform ("terrahub"). It supports an automated reconciliation loop and is safe to recreate.

The second segment has no automated reconciliation loop for various reasons like manual archiving or configuration drift.

The third segment is automated via the flux operator.

11.2.2.1 Segment 1 - Roll out Runtime Infrastructure

The first segment relies on "terrahub" which is an EOX customized automation tool to template and orchestrate terraform scripts for AWS infrastructure and kubernetes components. It supports rolling out the runtime environment to host all workloads, user-facing services & applications.

terrahub uses a special Git repository with a declarative configuration file to manage cloud services. This special Git repository is stored in a self-hosted GitLab instance.

The CI/CD pipeline clones this Git repository and applies the configuration file:

```
git clone https://gitlab-ci-token:${CI_JOB_TOKEN}@gitlab.eox.at/eox/hub/terrahub-deploy

cd terrahub-deploy/hub-test

terrahub deploy -c config.yaml --disable-prompt
```

terrahub uses the secrets (AWS credentials) that are configured for the CI/CD pipeline.

11.2.2.2 Segment 2 - Roll out Storage and Access

The second segment is managed via the AWS CLI and console. Different setups for staging and production environments are rolled out. S3 storage is configured with fine grained IAM permissions on bucket level and backup and retention policy. IAM roles and permissions are created which are propagated to EKS cluster as kubernetes secrets.

11.2.2.3 Segment 3 - Deploy User Facing Services and Applications

The third segment is the Continuous Deployment (CD) which is managed via the flux GitOps operator as described above and backed by a Git configuration repository. It is used to deploy new versions of customer facing apps and services. The GitOps principle helps to enforce an "avoid branching" policy, i.e., customization must be possible via configuration. Developers and operators are responsible to integrate their product into an environment, i.e., responsible teams directly commit to the Git configuration repository.