

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

WP4 Share

D4.4 Operational FAIRiCUBE HUB

Deliverable Lead: EOX
Deliverable due date: 30-06-2024

Version: 1.0
14-06-2024



Document Control Page

Document Control Page	
Title	D4.4 Operational FAIRiCUBE HUB
Creator	EOX
Description	This document describes the operational FAIRiCUBE HUB as an Exploitation platform which was deployed early in the project and gets continuously extended with concrete services, apps and data offerings as required by the Use Cases.
Publisher	"FAIRiCUBE – F.A.I.R. information cubes" Consortium
Contributors	CU, All
Date of delivery	30-06-2024
Type	Text
Language	EN-GB
Rights	Copyright "FAIRiCUBE – F.A.I.R. information cubes"
Audience	<input checked="" type="checkbox"/> Public <input type="checkbox"/> Confidential <input type="checkbox"/> Classified
Status	<input type="checkbox"/> In Progress <input type="checkbox"/> For Review <input checked="" type="checkbox"/> For Approval <input type="checkbox"/> Approved

Revision History			
Version	Date	Modified by	Comments
0.1	23-06-2023	Christian Schiller, Stephan Meißl, EOX	Initial draft
0.2	2023-07-10	Christian Schiller, Stephan Meißl, EOX;	Shortened FAIRiCUBE Lab description & cleanup Added dynamic STAC Catalog Added rasdaman description
0.3	2023-07-24	Mohit Kumar Basak	Added information relevant to rasdaman Added operation scenarios
0.4	2024-05-31	Peter Baumann	Enhanced rasdaman parts
1.0	2024-06-14	Christian Schiller Peter Baumann	Updated Catalog Data Requests / Metadata Editor, following reviewer request Update following reviewer request
1.0	2024-06-14	Jaume Targa	Review and format checking



Disclaimer

This document is issued within the frame and for the purpose of the FAIRICUBE project. This project has received funding from the European Union's Horizon 2020 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.

This document and its content are the property of the FAIRICUBE Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the FAIRICUBE Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the FAIRICUBE Partners. Each FAIRICUBE Partner may use this document in conformity with the FAIRICUBE Consortium Grant Agreement provisions.



Table of Contents

Document Control Page	2
Disclaimer	3
Table of Contents	4
List of Tables	8
1 Introduction	9
2 FAIRiCUBE Lab, aka EOxHub deployment.....	11
2.1 Control Plane	11
2.1.1 Configuration Management	12
2.1.2 GitHub as Identity Provider	12
2.1.3 JupyterLab Profiles	12
2.1.4 Keycloak	13
2.1.5 Shared Jupyter Notebooks	13
2.1.6 Shared Conda Environments	15
2.1.7 Shared Secrets	16
2.1.8 Shared Object Storage.....	16
2.1.9 Apps.....	16
2.1.10 Complete Configuration Example	17
2.2 Worker Plane	18
2.2.1 Interactive Development Environment - Jupyter Notebooks	18
2.2.2 Data access	19
2.2.3 User access.....	19
2.2.4 Headless Notebook execution (i.e. non-interactive)	21
2.2.5 Machine Learning Platform - MLflow	22
3 Catalog.....	23
3.1 Requests - Overview.....	23
3.2 Dataset metadata processes	23
3.3 Resource metadata processes	26
3.4 Codelist Change processes.....	30
3.5 Catalog UI – STAC Browser using fastAPI	31
4 Community Collaboration Platform.....	35
5 Knowledge Base	37
6 FAIRiCUBE GitHub Repository	39
7 Access to external resources	41
7.1 On-the-fly data cube access: Sentinel Hub	41
7.2 Mass processing service: Sentinel Hub Batch Processor	41
7.3 Pre-generated data cubes with xcube	42
7.4 Sentinel Hub Dashboard	42
7.5 Data uploading – Example: Bring Your Own COG API	42
8 The rasdaman Datacube Deployment	43
8.1 Rasdaman in the Project Landscape	43
8.2 Design Philosophy	44
8.3 The rasdaman Engine.....	45
8.4 Data & Ingestion	45
8.5 Machine Learning	46
8.6 APIs & Clients	47
8.7 Development Environment.....	48
8.8 Support	50



8.9	Integration with the FAIRiCUBE Hub.....	51
8.10	Access Control	51
8.11	Development outlook.....	51



List of Figures

Figure 1: FAIRiCUBE HUB Architecture.....	9
Figure 2: Keycloak for authorization	13
Figure 3: FAIRiCUBE Catalog launcher tile.....	14
Figure 4: FAIRiCUBE Catalog Notebook Viewer.....	14
Figure 5: Example of a Conda-store environment configuration (here from DeepESDL).....	15
Figure 6: Examples of Kernel selection in a Conda environment (here from DeepESDL)	15
Figure 7: Example of MLflow for experiment tracking (here from DeepESDL)	17
Figure 8: FAIRiCUBE HUB Login	19
Figure 9: JupyterLab profile selection - Use Case specific workspace profiles.....	20
Figure 10: Information that FAIRiCUBE workspace is being prepared.....	20
Figure 11: JupyterLab workspace launcher.....	20
Figure 12: JupyterHub Control panel	21
Figure 13: Greeting page after successful Login	21
Figure 14: Screen presented to Users not configured to a Use Case	21
Figure 15: Data Ingestion Request Web GUI – Landing page	24
Figure 16: Data Ingestion Request Web GUI - Data entry Part-1	25
Figure 17: Data Ingestion Request Web GUI - Data entry Part-2	25
Figure 18: Data Ingestion Request Procedure	26
Figure 19 : webGUI – Landing page	27
Figure 20 : Screenshot of the a/p metadata webform (1).....	28
Figure 21 : Screenshot of the a/p metadata webform (2).....	29
Figure 22: Codelist change proposal Request	30
Figure 23: Dynamic Catalog based on STAC -fastapi.....	31
Figure 24: Data access catalog – List of dataset resources.....	31
Figure 25: Data access catalog – Example of a dataset (NUTS3_2021)	32
Figure 26: Data access catalog – Search interface associated with a dataset.....	32
Figure 27: Analysis and processing resources catalog - Listing of available resources as collections ...	33
Figure 28: Analysis and processing resources catalog – Example of a ML resource	33
Figure 29: Analysis and processing resources catalog – Feature to browse available datasets	34
Figure 30: Analysis and processing resources catalog – Example of a Non-ML resource featuring direct data access.....	34
Figure 31: Landing page of the Community collaboration platform	36



Figure 32: rasdaman in the FAIRiCUBE HUB Architecture, with rasdaman parts highlighted44

Figure 33: Sample model application as a query result (left) and the schematic WCPs query generating it47

Figure 34: sample rasdaman dashboard screenshot (see text for explanation of numbers).....48

Figure 35: Jupyter lab environment on FAIRiCUBE rasdaman server49

Figure 36: rasdaman Web GUI supporting interactive Web request generation.....49

Figure 37: rasdaman Web GUI supporting interactive Web request generation.....50



List of Tables

Table 1: List of GitHub repositories used by FAIRiCUBE39

1 Introduction

This is an overview documentation of the deliverable D4.4 Operational FAIRiCUBE HUB (DEMO) of the FAIRiCUBE project, i.e. the FAIRiCUBE integrated datacube platform. It describes the deployed and operated FAIRiCUBE HUB as an Exploitation Platform provided during the duration of the project. The FAIRiCUBE Hub can be accessed at: <https://eoxhub.fairicube.eu>. Access to the Hub is possible using a valid GitHub account, but currently an additional (manual) registering step is configured. A collection of links to the various resources currently available can be found at: <https://fairicube.nilu.no/fairicube-hub/>

The FAIRiCUBE HUB consists of the FAIRiCUBE Lab, a Catalog, the Data Access Services, and the Data Processing Services. The FAIRiCUBE Lab, acting as the central point of user activities, is based on an instance of EOxHub, as well as rasdaman and other Open-Source components. The EOxHub instance provides a Control Plane, a JupyterHub providing Jupyter Notebooks, and access APIs to the various FAIRiCUBE Services (Figure 1).

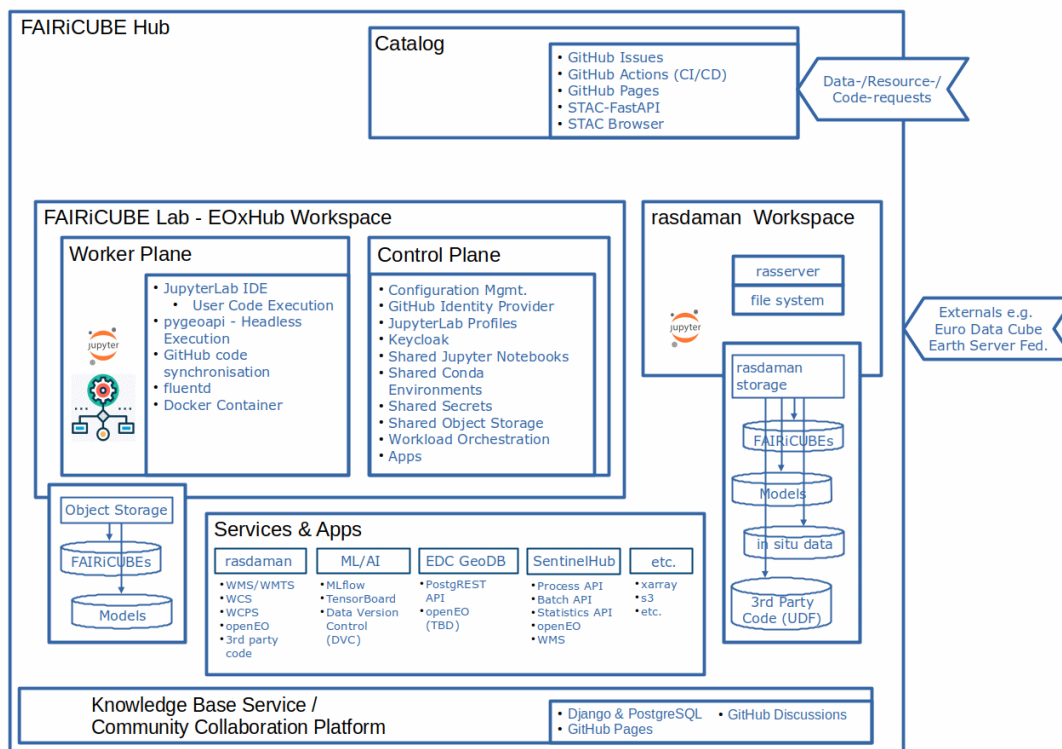


Figure 1: FAIRiCUBE HUB Architecture

Below are the definitions of the components of the FAIRiCUBE HUB provided to avoid confusions and ambiguities:

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

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

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



- **EOxHub Workspace-Worker Plane:** The user area within the FAIRiCUBE Lab where users can collaborate and share the content of their workspaces.
- **EOxHub Workspace-Control Plane:** The administration area within the FAIRiCUBE Lab where all configuration and setup is managed

FAIRiCUBE rasdaman Workspace: consists of the rasdaman datacube engine, a multi-parallel distributed database system specialized on the management and analytics of massive multi-dimensional arrays (Chapter **Error! Reference source not found.**).

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

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

FAIRiCUBE 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.

FAIRiCUBE – GitHub Repository: FAIRiCUBE has setup and uses a GitHub repository very extensively. In this FAIRiCUBE GitHub repository the catalog data, shared code, use case specific code, as well as the content of the Knowledge Base and the Community Collaboration Platform is stored and accessible to FAIRiCUBE users (Chapter **Error! Reference source not found.**).

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 sufficient 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.

A Knowledge Base, enabling simple access to essential information on working with large spatiotemporal datacubes. For more details on the Knowledge Base, please see deliverable D3.4 Processing knowledge base services.



2 FAIRiCUBE Lab, aka EOxHub deployment

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.

2.1 Control Plane

The Control Plane was deployed early in the project and is readily available from EOxHub. It is continuously extended with specific services, apps, and data offerings as required by the Uses Cases.

The Control Plane is initially configured to support at least the following:

- User Workspaces (Tenants)
- Service subscription management
- Marketplace
- Allocation functions for cloud resources and Data Services
- Deployment service
- Workload management functions
- Accounting and billing (voucher handling)

Cloud resources for the Worker Plane will be made available as needed by and agreed with the use cases as they have to cover the corresponding cloud costs.

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

The control plane is declaratively deployed and operated through GitOps principles relying on the Flux CD tooling. 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:

- Kubernetes service to provide a kubernetes cluster
- S3 for object storage
- Open ID Connect (OIDC) service for user management
- Network File System (NFS) for block storage

The main tools running in the control plane are



- Grafana to provide metrics dashboards
- Alert manager to send notifications for example to dedicated operations Slack channels
- Brigade for running scriptable, automated tasks
- Elastic stack of elasticsearch, fluentd, and kibana (EFK) to collect and present logs

2.1.1 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, secrets, 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.

2.1.2 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**.

```

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

```

```

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

```

2.1.3 JupyterLab Profiles

Different JupyterLab profiles can be configured and are available for authenticated and authorized users as shown in Figure 9.

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

```

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_faircubeuc4: display_name=FAIRiCUBE-
UC4,node_purpose=useruc4,mem_guarantee=30064771072,cpu_guarantee=7,mem_limit=32212254720,cpu_lim
it=7.5,s3_bucket_name=hub-faircubeuc4,secret_names=hub-faircubeuc4
  s3_bucket: "53687091200"
  storage: "53687091200"
  user: "2592000" #30d
  useruc4: "2592000" #30d
  url: https://eoxhub.faircube.eu
entityId: eoxhub
entityType: infra
    
```

The main information is encoded in this line:

```

profile_faircubeuc4: display_name=FAIRiCUBE-
UC4,node_purpose=useruc4,mem_guarantee=30064771072,cpu_guarantee=7,mem_limit=32212254720,cpu_lim
it=7.5,s3_bucket_name=hub-faircubeuc4,secret_names=hub-faircubeuc4
    
```

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-faircubeuc4".

2.1.4 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.

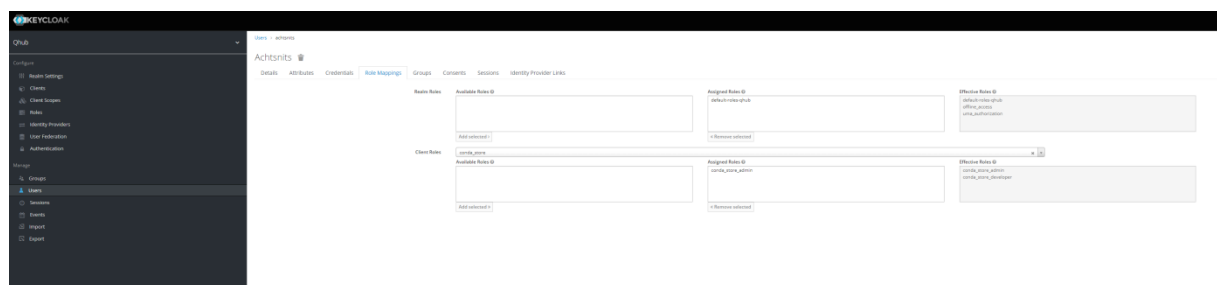


Figure 2: Keycloak for authorization

2.1.5 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 9. 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.

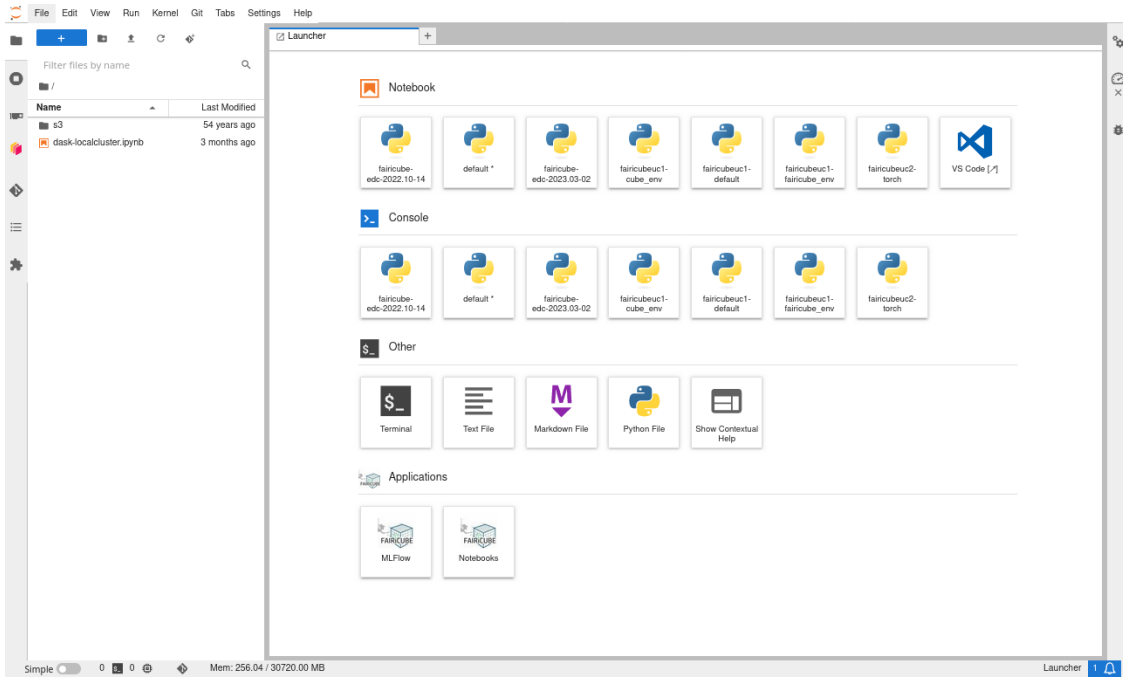


Figure 3: FAIRiCUBE Catalog launcher tile

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).

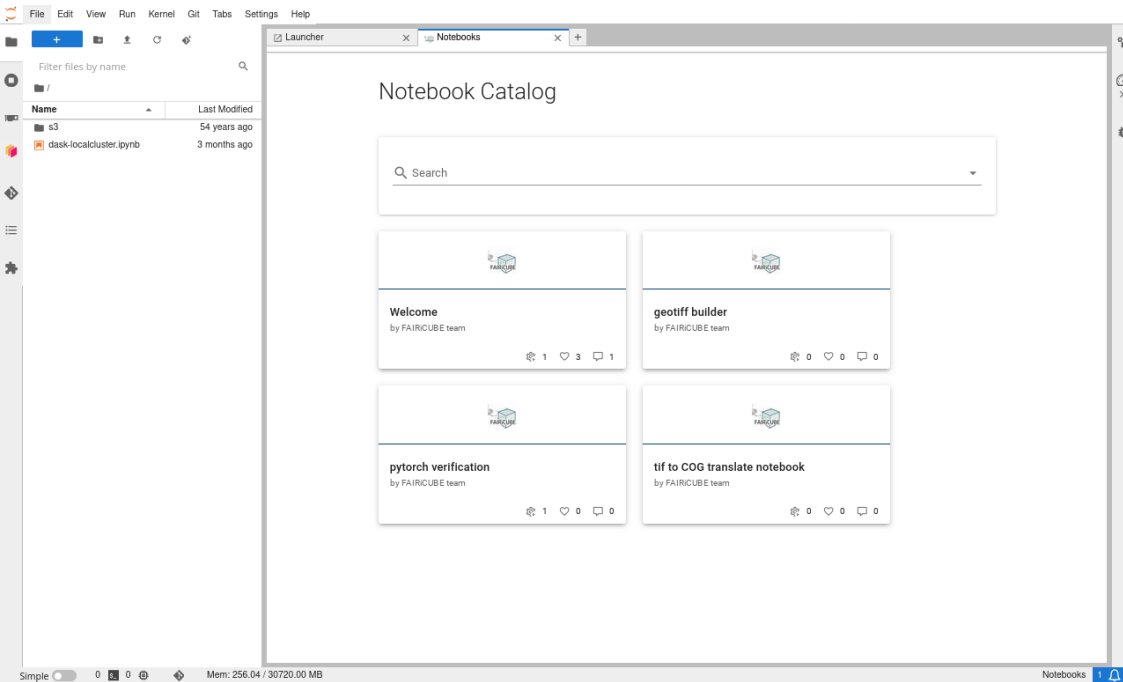


Figure 4: FAIRiCUBE Catalog Notebook Viewer

2.1.6 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 5 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.

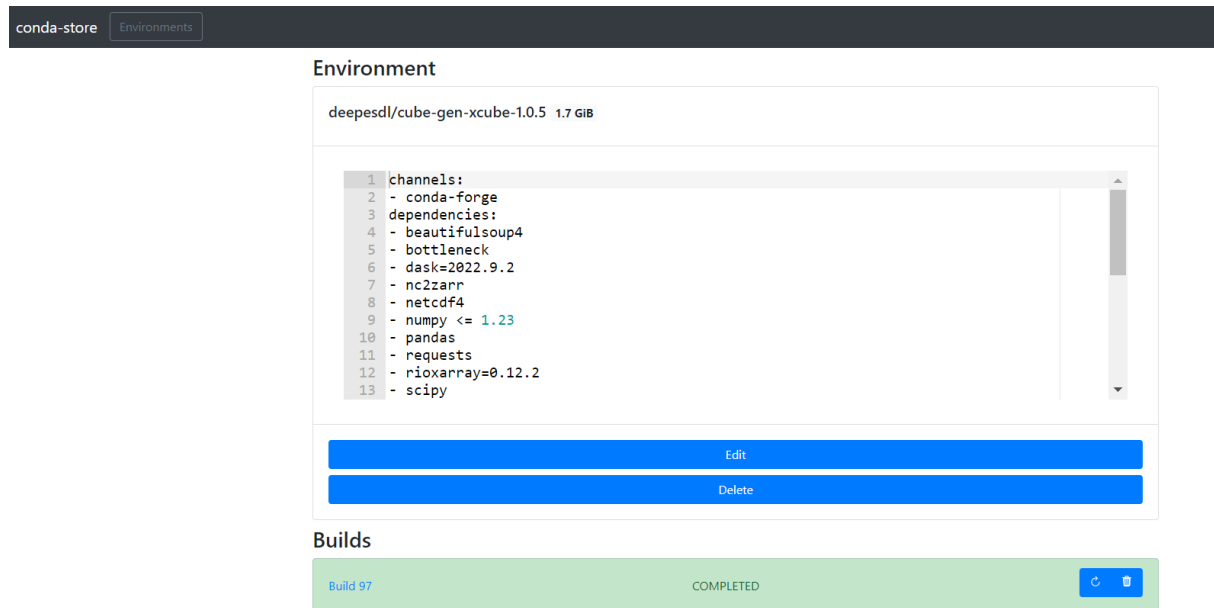


Figure 5: Example of a Conda-store environment configuration (here from DeepESDL)

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 6.

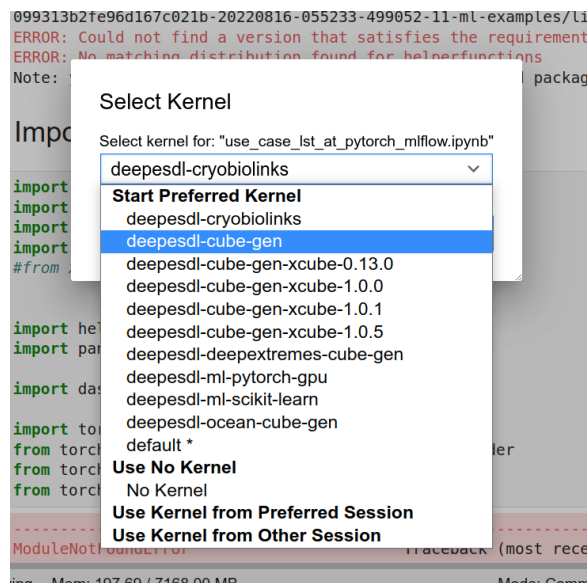


Figure 6: Examples of Kernel selection in a Conda environment (here from DeepESDL)



2.1.7 Shared Secrets

Sometimes it is necessary to share configuration values, particularly secrets, within teams. In order not to 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 git (even if it is a protected git repository) or not. For sensitive values it is recommended to check them in 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:

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

2.1.8 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.

```
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.

2.1.9 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 7.

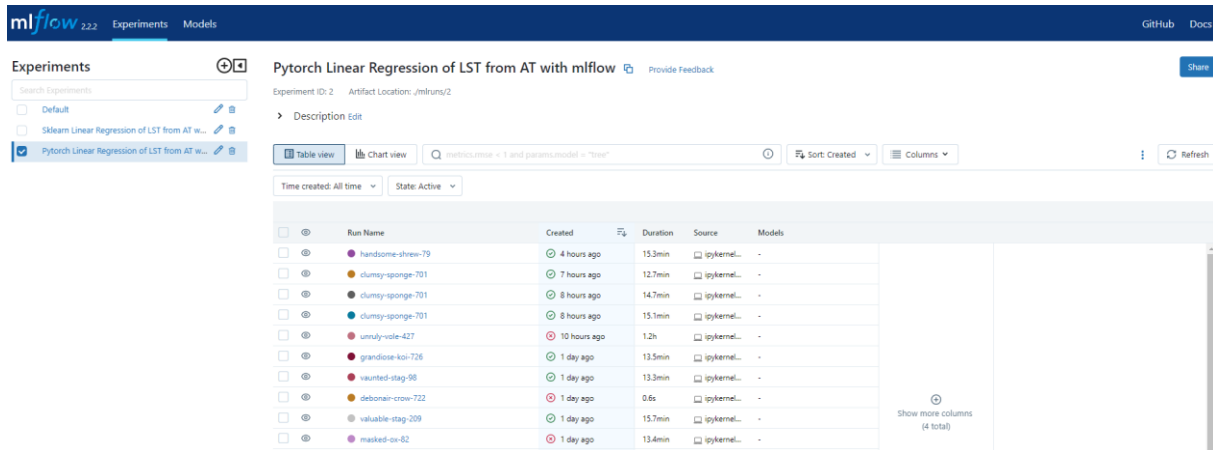


Figure 7: Example of MLflow for experiment tracking (here from DeepESDL)

2.1.10 Complete Configuration Example

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

```

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=32212254720,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

```

2.2 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 (Data Version Control) for collaborative data management like ML artefacts
- MLflow or TensorBoard to support Machine Learning operations (MLOps)
- Almost any Docker image can be deployed and run in the worker plane
- fluentd to collect logs for debugging

Further apps are available, for example to show processing results in a FAIRiCUBE Viewer.

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”.

2.2.1 Interactive Development Environment - Jupyter Notebooks

The FAIRiCUBE Lab is centred on the usage of Jupyter Notebooks (Figure 11), 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.

2.2.2 Data access

Various sources for datasets can be used in FAIRiCUBE. Beside datasets already available in FAIRiCUBE, Data provided by the users can be also be used. These can be stored locally in an S3 bucket or even uploaded to other locations, e.g to SentinelHub for further processing. However, it has to be noted that certain restriction regarding data formats (e.g BYOC) may apply (see also Chapter **Error! Reference source not found.**).

In addition, Data delivered via Sentinel Hub 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. Such an access to external data sources can also be made for other data providers and their offerings (e.g. . WEkEO, and others)

2.2.3 User access

The FAIRiCUBE Lab is deployed at <https://eoxhub.fairicube.eu>. The authentication is using GitHub as identity provider as shown in Figure 8.

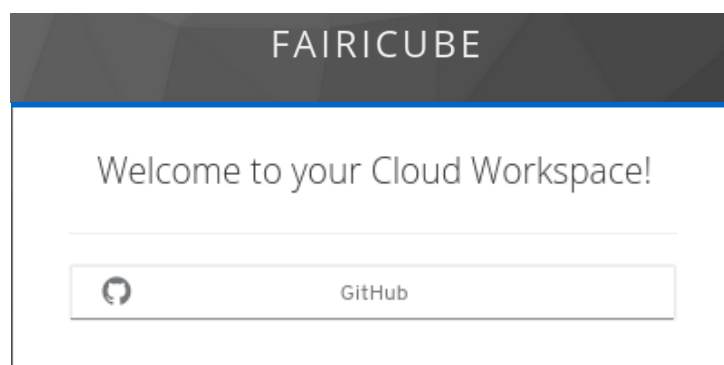


Figure 8: FAIRiCUBE HUB Login

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 9. Configuration details are provided below.

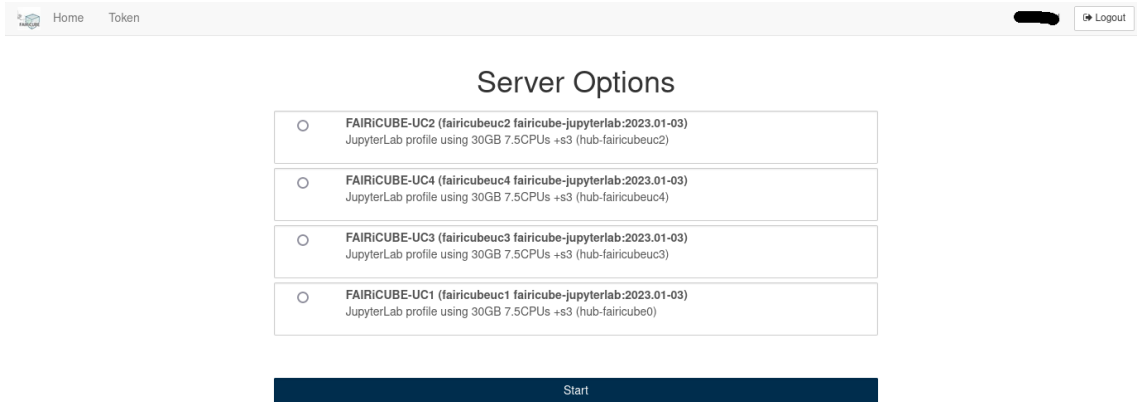


Figure 9: JupyterLab profile selection - Use Case specific workspace profiles

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 10: Information that FAIRiCUBE workspace is being prepared. The start-up procedure might require some minutes since the whole workspace has to be prepared and provisioned.

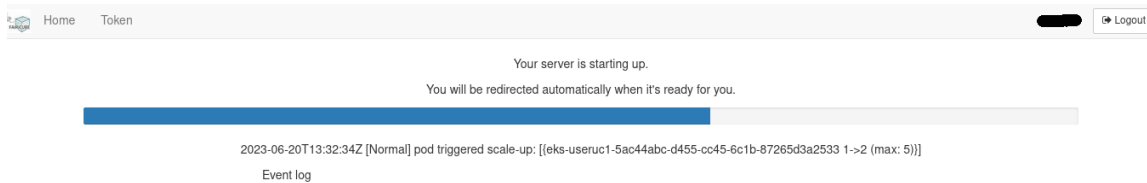


Figure 10: Information that FAIRiCUBE workspace is being prepared

When the start of the respective server has successfully finished, the User's JupyterLab workspace will be provided (Figure 11: JupyterLab workspace launcher).

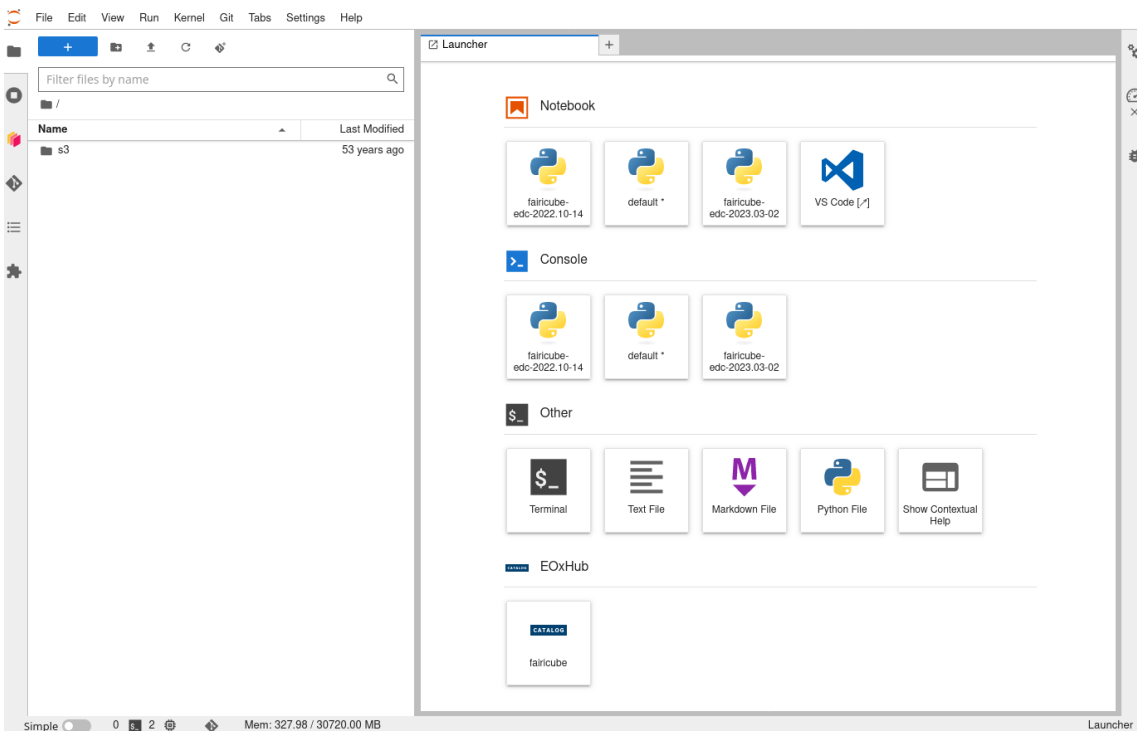


Figure 11: JupyterLab workspace launcher

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 12: JupyterHub Control panel). 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.

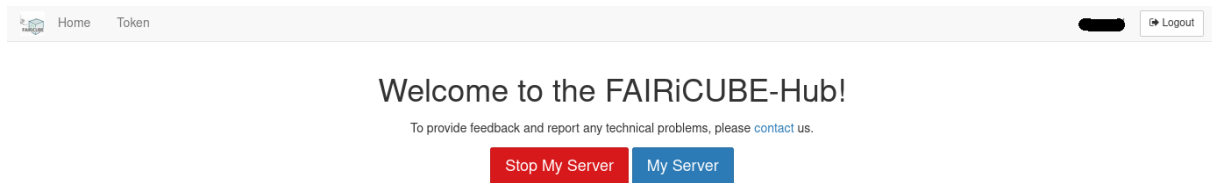


Figure 12: JupyterHub Control panel

In rare cases, e.g. when a user session hangs, it can be necessary to Stop a server manually (Figure 12). Thereafter the server can also be started again manually (Figure 13).

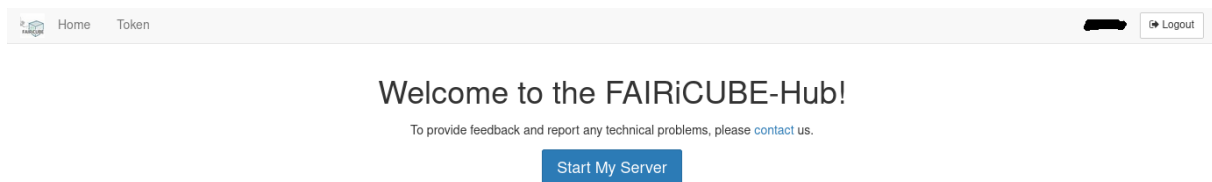


Figure 13: Greeting page after successful Login

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 14).

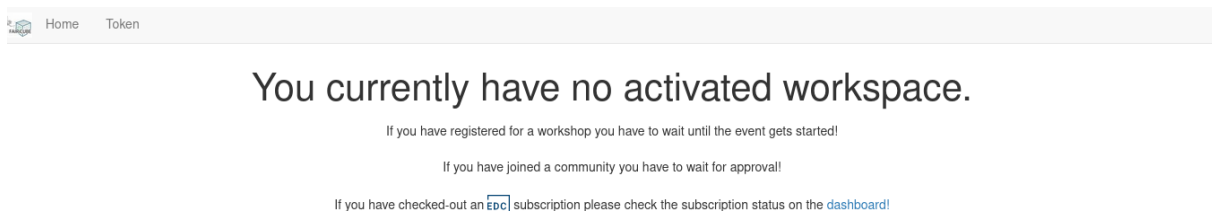


Figure 14: Screen presented to Users not configured to a Use Case

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.

2.2.4 Headless Notebook execution (i.e. non-interactive)

Any Jupyter notebook able to run on the FAIRiCUBE workspace offering (i.e. EOxHub) may also be executed via API, leveraging the same capabilities (libraries, injected environment variables) and respecting the same user resource constraints (quotas) as through direct invocation via the JupyterLab interface.

Detailed information about the usage of this feature is provided at: <https://eurodatacube.com/documentation/headless-notebook-execution>. This information will later be also provided directly within the FAIRiCUBE knowledgebase.



2.2.5 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.

The following components of can be made available for the user:

- MLflow Tracking: allows users to log and track training parameters, code, and output metrics from their machine learning experiment
- MLflow Projects: provides a standard format for packaging and distributing machine learning code, including dependencies, in a reproducible way
- MLflow Models: allows users to easily package models in a standard format for deployment and export in multiple formats
- MLflow Model Registry: allows users to store, manage, and deploy models in a central repository including versioning, and access control using role-based access control

3 Catalog

3.1 Requests - Overview

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, Earth Server Federation, etc.) then a User can issue a Data Request to get the data ingested into FAIRiCUBE. Two type of Data request are provided by FAIRiCUBE:

- **data-requests:** Request for data to be made available within FAIRiCUBE Lab.
 - New Issue Data request (chapter **Error! Reference source not found.**)
- **resource-metadata:** Collect information for processing/analysis (a/p/) resources as well as propose a change to a codelist.
 - New Issue Resource request (see chapter 3.3)
 - New Issue codelist change (see chapter **Error! Reference source not found.**)

The process to submit a data request or a resource-metadata request is implemented utilizing specialized Web Interfaces. Triggering a New Issue provides an input form to help the user to submit and fill all required fields. For the full details of these processes see '*D4_2 Public Listing (Catalog) of FAIRiCUBE data resources.docx*' and '*D4_3 Public Listing (Catalogue) of FAIRiCUBE processing-analysis resources.docx*', respectively.

3.2 Dataset metadata processes

EOX developed a Web-GUI (Figure 15) as an Input Frontend allowing to collect and edit the metadata for the data resources. The input from this Web-GUI is collected, checked for consistency and errors and then directly stored as static STAC json items in GitHub. This ensures that the items stored in the GitHub repository act as the single "*Source of Truth*". The same interface is also available to edit already ingested metadata items. 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 a GitHub issue associated to the respective Pull Request, so that everybody interested can follow the progress and provide additional feedback or information as necessary.

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. When the merge is done, the newly submitted data is available as a STAC item (STAC-fastapi/pgSTAC to provide a STAC API) and available via the Catalog Client, based on STAC Browser, deployed at <https://catalog.faircube.eu>. A detailed flow-chart of this process is provided in Figure 18. The WebGUI for metadata requests can also be used to edit & update metadata on already ingested datasets. In Figure 15, a list of available datasets is shown, each with an "Edit-Button" associated. The available "Link-Button" leads to the respective *.json file located in the GitHub repository.

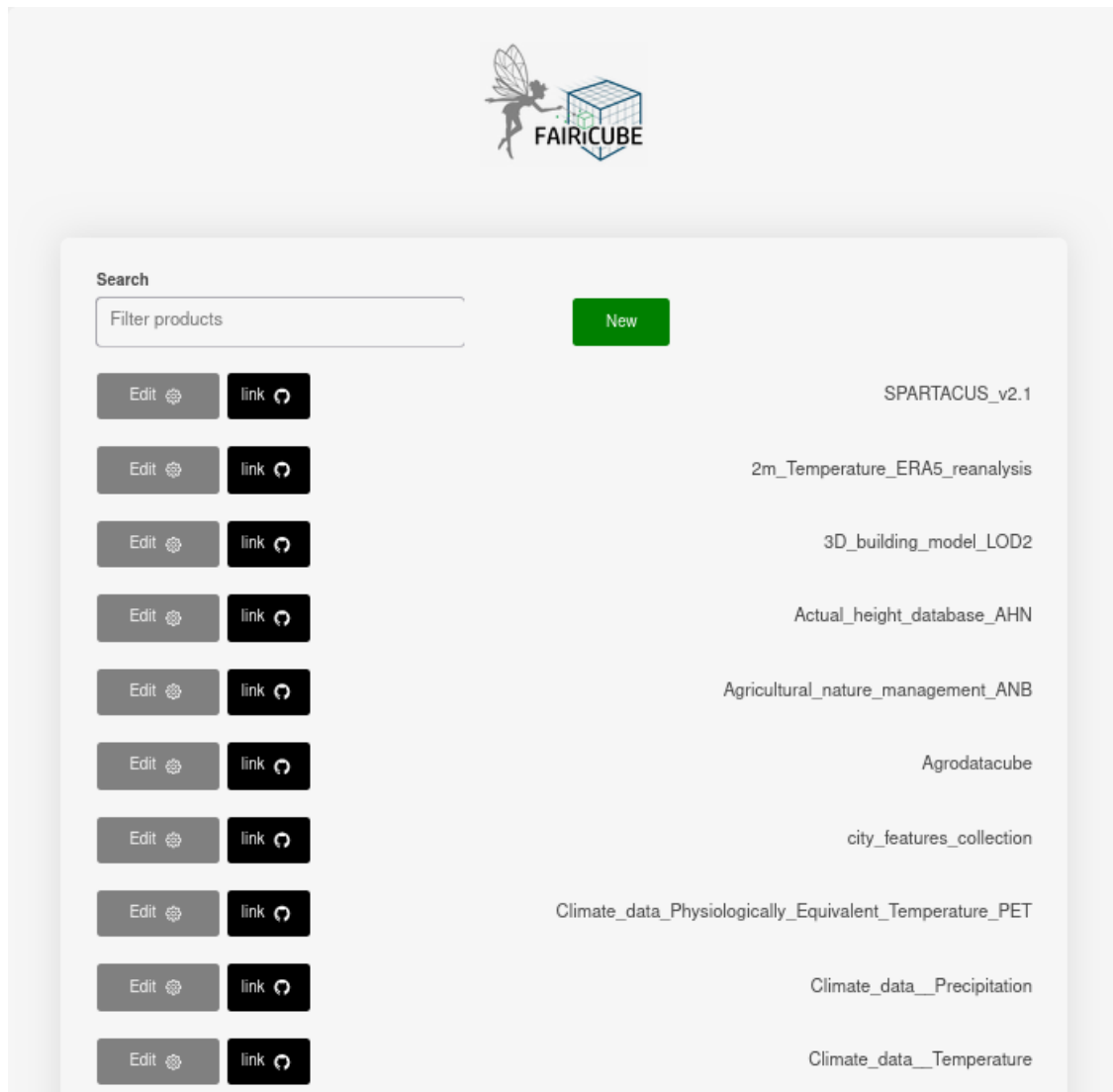



Figure 15: Data Ingestion Request Web GUI – Landing page

Figures 16 and Figure 17 show the input page of the WebGUI, providing the input fields for the metadata. Depending on the input field, drop-down lists and calendar tools are provided. During submission of this form certain plausibility checks are performed.



[back](#)

Title

The title of the issue request

ID

The ID of the requested stac item

Description

Brief, nontechnical explanation of the datacube.

Source

The link to the source of the dataset.

Source Type

The data source type

Organizations
[+ Add bands](#)

Figure 16: Data Ingestion Request Web GUI - Data entry Part-1

Resolution

Resolution (or 'irregular'). Should be 1 value as required by UC, not all resolutions of dataset

[Remove](#)
remove the extra dimension(Axis)

[+ Add another](#)
Add another dimension(Axis)

Bands

cell components <input type="text"/>	Unit of Measure <input type="text"/>	Data Type Select a data type ▾	Null values <input type="text"/>
Definition <input type="text"/>	Description <input type="text"/>	Category List <input type="text"/>	Comment <input type="text"/>

[Remove](#)
remove band

[+ Add bands](#)
Add another band

Re-projection axis

Horizontal CRS

Unit of Measure

Figure 17: Data Ingestion Request Web GUI - Data entry Part-2

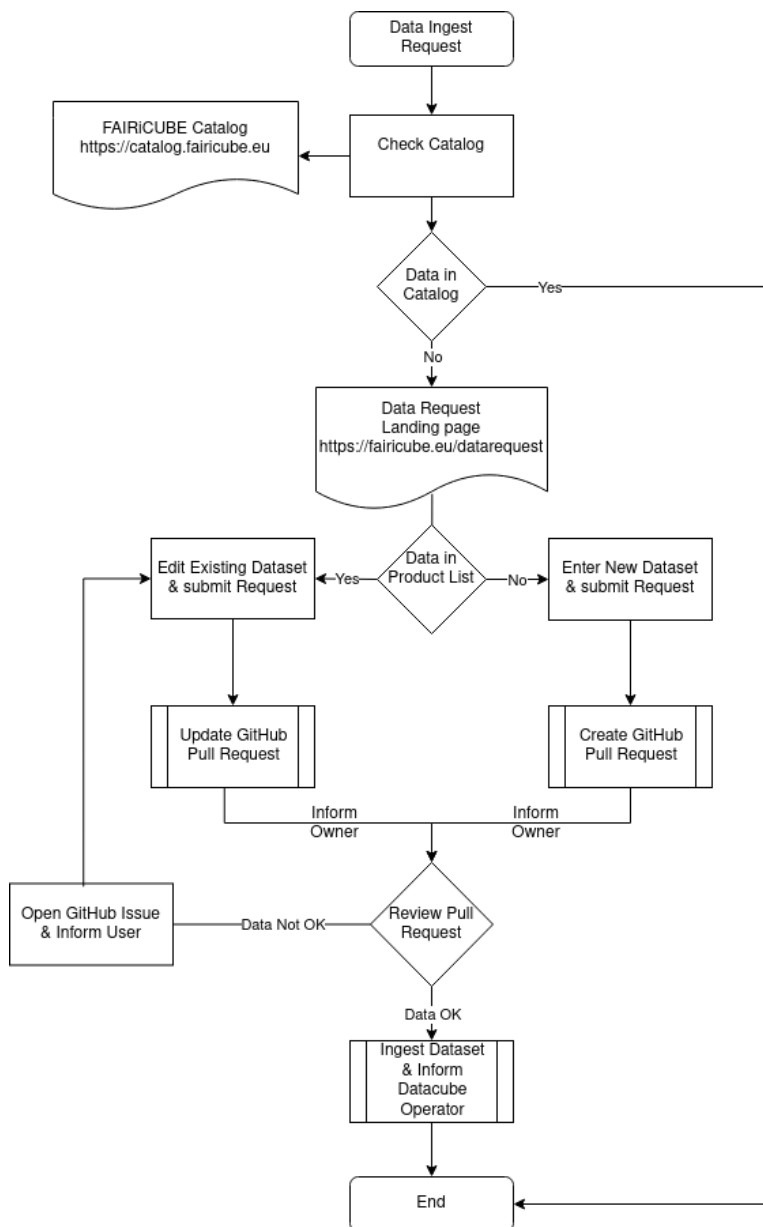


Figure 18: Data Ingestion Request Procedure

3.3 Resource metadata processes

Creating metadata for a new a/p resource as well as updating existing metadata records is handled via a dedicated web application. Specifically, the entry point for the user wanting to create/update a metadata record is the webGUI illustrated in Figure 19 below, analogous in all respects to the webGUI for managing dataset metadata described in deliverable '*D5.2 Description of the datacube ingestion pipelines*'. This process is detailed in deliverable '*D4.3 Public Listing (Catalogue) of FAIRiCUBE processing/analysis resources*'. This webGUI is accessible only to authenticated users.

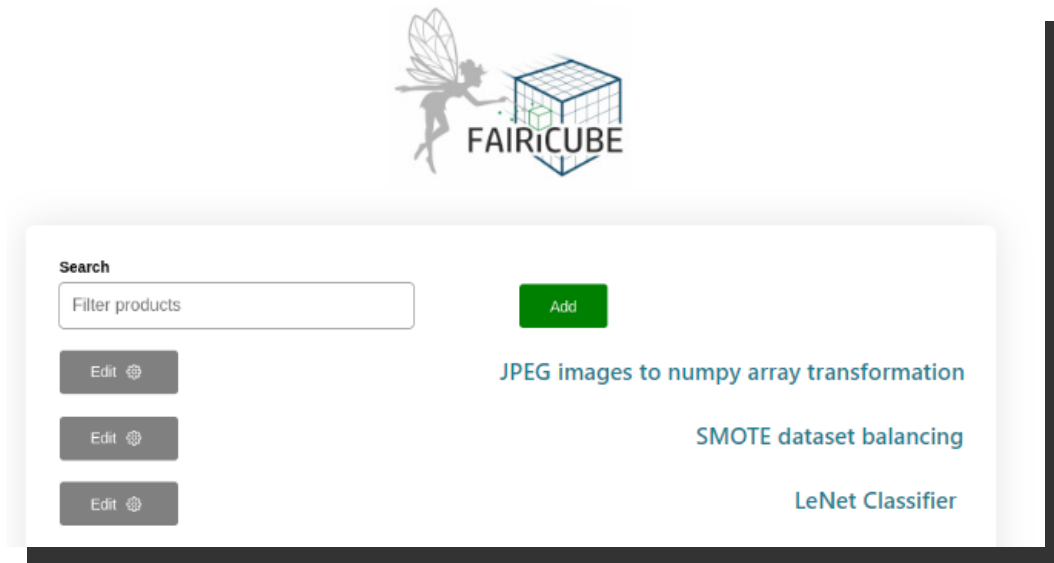


Figure 19 : webGUI – Landing page

Adding and updating of the a/p resource metadata happens via a dedicated a/p metadata entry webform, which is displayed when clicking on the Add or the Edit buttons. When the Edit button is clicked, the webform is prefilled with the information of the metadata file to be updated.

The a/p metadata entry webform

The Python programming language, the Flask web micro-framework and the Keycloak identity and access management solution were the software used to create the a/p metadata entry form application. This form contains all the fields for documenting an analysis and processing resources. Further details are provided in '*D4_3 Public Listing (Catalogue) of FAIRiCUBE processing-analysis resources.docx*'. In this document only a subset of the web form is provided in Figure 20 and Figure 21.



Log out



Analysis and Processing Resource metadata

Resources can be algorithms, models, or pre-processing pipelines and so on.

Please provide the information requested below for the resource or click on the log-out button.

Mandatory elements are marked with *.

Elements marked with ** are mandatory only for Machine Learning and Deep Learning resources.

The username will be placed in the metadata under the provider field and is not editable.

Logged user:

epsit



epsit

Title *

Use case *

Use case to which the resource belongs

Name of resource *

Textual name identifying the resource. Resource can be the algorithm, the model, or the pre-processing pipeline.

ID *

Globally unique and persistent identifier of the resource.

Description *

Description of the resource.

Log out
Edit metadata



Analysis and Processing Resource metadata

Provide the information requested below for the resource.

* mandatory field - ** mandatory fields for ML and DL resource only

Logged user:

epsit

Title *

Use case *

Name of resource *

Name identifying the resource. Resource can be the algorithm, the model, or the pre-processing pipeline.

ID *

Globally unique and persistent identifier of the resource.

Description *

Description of the resource.

General section

Publication date *

Date of publication - i.e., first sharing with the scientific community - of the resource.

Main category *

Main category of the resource, e.g. ML, ingestion, pre-processing. Notice that the codelist contains also some specific elements to further specify the resource (e.g., "DL" as a special case of "ML").

Figure 20 : Screenshot of the a/p metadata webform (1)



Description of the resource.

Main category *

Main category of the resource, e.g. ML, ingestion, pre-processing.
Notice that the codelist contains also some specific elements to further specify the resource (e.g., "DL" as a special case of "ML").

Publication date *

Date of publication - i.e., first sharing with the scientific community - of the resource.

Objective *

This element should be used to provide info about "What does the resource perform", i.e., the purpose of the resource.

Platform *

Platform hosting the resource

Framework *

This field is generally intended as collection of reusable code written by others. In this respect, it includes both frameworks, intended as program scaffolds that supply the blueprint of a product, and libraries, intended as collections of pre-defined methods and classes.

Architecture **

This is a conditional element, applied only to ML algorithms.
This field has a different meaning than "algorithm" because there are often multiple "implementations" of an architecture.

Approach **

This is a conditional element, applied only to ML algorithms, identifying the learning modality.

Algorithm *

This field contains the name of the algorithm, i.e., an implementation of an architecture.
This field has a different meaning than "architecture" because there are often multiple "implementations" of an architecture.

Objective *

This element should be used to provide info about "What does the resource perform", i.e., the purpose of the resource.

Platform *

Platform hosting the resource

Framework *

This field is generally intended as collection of reusable code written by others. In this respect, it includes both frameworks, intended as program scaffolds that supply the blueprint of a product, and libraries, intended as collections of pre-defined methods and classes.

Algorithm *

This field contains the name of the algorithm, i.e., an implementation of an architecture. This field has a different meaning than "architecture" because there are often multiple "implementations" of an architecture.

Processor *

Type of processor used for training or inference in learning tasks, but also for running a pre-processing notebook. If more than one processor is used to perform the entire task, indicate the one with the highest computing power.

OS *

Operating System used.

Keywords *

A series of keywords which can facilitate the resource discoverability. These keywords can be used for a quick search of resources and should highlight foremost characteristics of the resource. For the categorization of the resources via keywords to be effective, we recommend that a maximum of 5 keywords is provided separated by comma.

Reference link

Link to resource web page (including publications) and/or download page

Jupyter Notebook available

Check the box if a Jupyter Notebook for this resource is available. In this case, the link to the notebook must be provided in the following Implementation example field.

Jupyter Notebook available

Figure 21 : Screenshot of the a/p metadata webform (2)

3.4 Codelist Change processes

The proposed approach to governance of the codelists accounts for the way STAC works with the list of values (i.e., using enumerations). However, the integration of external codelists will be investigated as needed. If possible and sensible (e.g., possible to collaborate with other projects on this), investigation results will be considered in the next update to D4.3. The codelists are managed in the 'resource-metadata' GitHub repository in the FAIRiCUBE GitHub organization. It is possible to propose the addition of new codelist values or the updating of existing ones by opening issues in the issue tracker of this repository, using the ad hoc template illustrated in Figure 22 below. The codelist change proposal template can be reached at: https://github.com/FAIRiCUBE/resource-metadata/issues/new?assignees=&labels=&projects=&template=codelist_change_proposal.yml

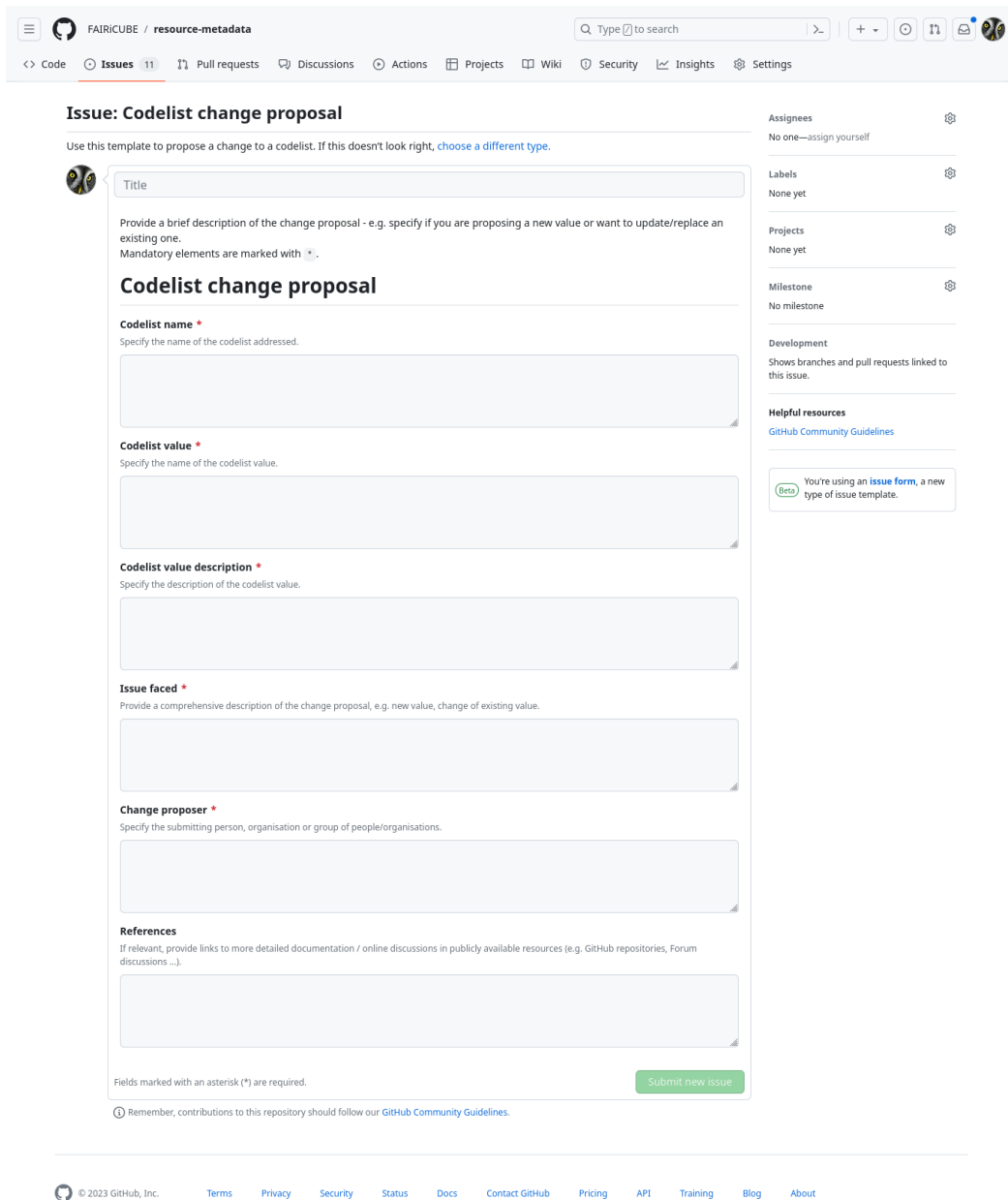


Figure 22: Codelist change proposal Request

3.5 Catalog UI – STAC Browser using fastAPI

The dynamic FAIRiCUBE catalog can be accessed at: <https://catalog.faircube.eu/?language=en>. The FAIRiCUBE catalog currently contains metadata of different collections (currently: data-access, ML-collection, non-ML collection). Efforts to integrate all collections is ongoing. The following Figures below show various aspects of the catalog interface. The first one, Figure 23, shows the landing page of the catalog, with its currently 3 metadata collections for data, ML resources, and non-ML resources.

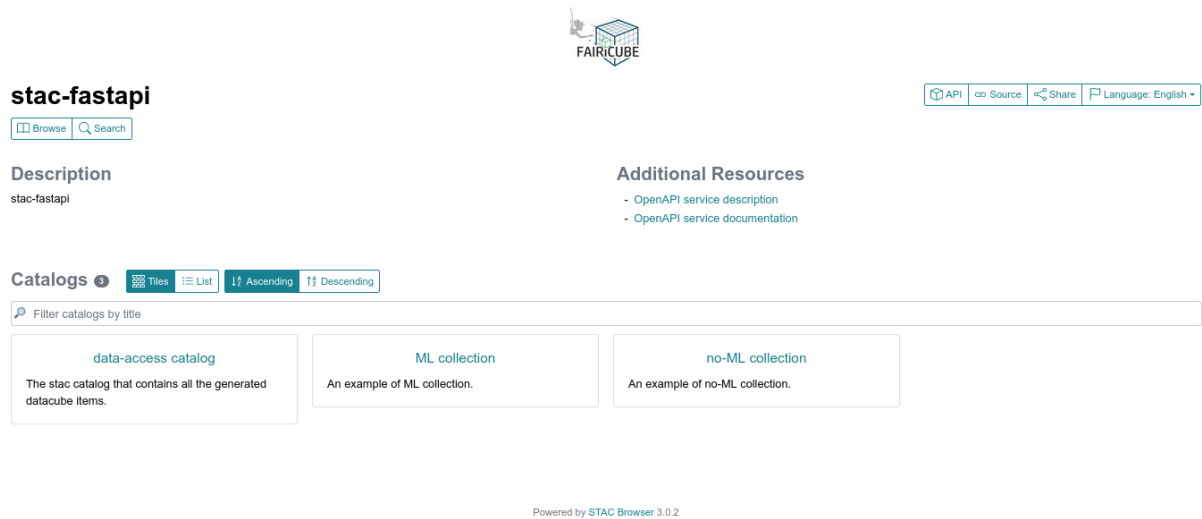


Figure 23: Dynamic Catalog based on STAC -fastapi

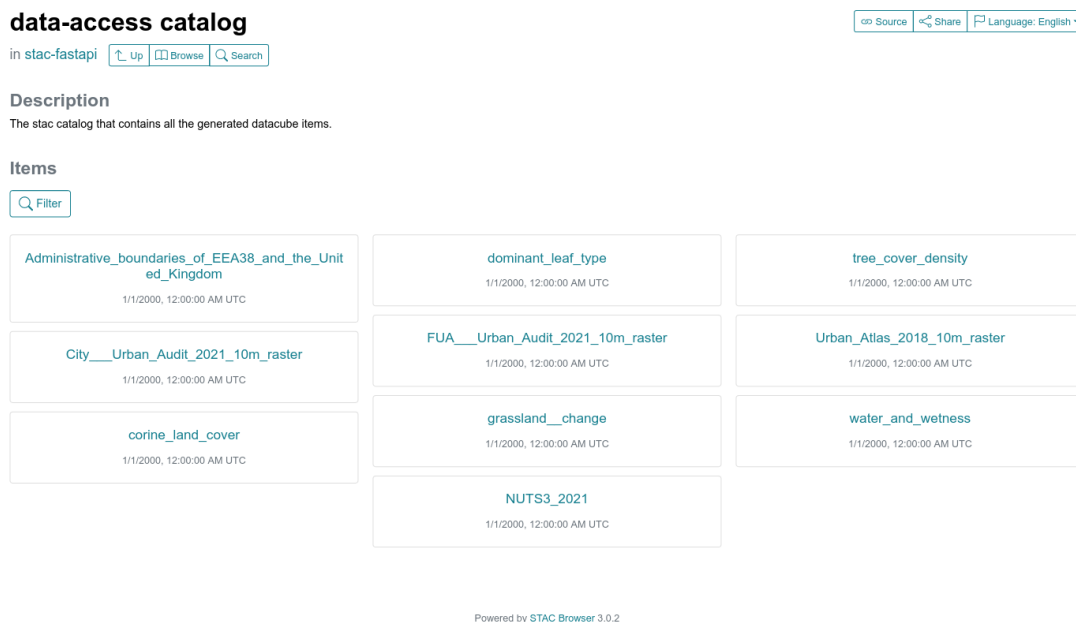


Figure 24: Data access catalog – List of dataset resources

Figure 24, show the "Data access page" with a list of dataset resources, while Figure 25 presents an example of a detailed view of a dataset (here, NUTS3_2021) presenting the view of the geographic coverage as well other details, like time stamp, projection and bounding box.

NUTS3_2021 Source Share Language: English

in stac-fastapi Up Collection Browse Search

Collection

data-access catalog
The stac catalog that contains all the generated datacube items.

General

Time of Data 1/1/2000, 12:00:00 AM UTC

Data Cube

Dimensions					
id	Type	Axis	Extent	Values	Reference System
x	spatial	x	-180 – 180	n/a	ESPG:4326
y	spatial	y	-90 – 90	n/a	ESPG:4326
time	temporal	n/a	n/a	2000-01-01T00:00:00	n/a

Powered by STAC Browser 3.0.2

Figure 25: Data access catalog – Example of a dataset (NUTS3_2021)

Figure 26 present a view of the search interface showing the possibilities to provide search criteria to be applied on the catalog.

Search Share Language: English

in stac-fastapi Browse Search

Temporal Extent

All times in Coordinated Universal Time (UTC).

Spatial Extent

Filter by spatial extent

Collections

Select one or multiple collections...

Item IDs

Enter one or more item IDs...

Additional filters

Match all filters (and) Match any filters (or)

Add filter -

Sort

Default

Not all of the options may be supported.

Items per page

Default (12)

Number of items requested per page, max. 10000 items.

Filter Reset

Please modify the search criteria.

Powered by STAC Browser 3.0.2

Figure 26: Data access catalog – Search interface associated with a dataset

As mentioned above the catalog also currently contains 3 collections, and Figure 27 presents the 2 resource collections of Analysis and Processing resources.

FAIRiCUBE Hub analysis and processing resources catalog

Source Share Language: English

in stac-fastapi Up Browse Search

Description

A STAC catalog that contains all resource collections available on the FAIRiCUBE Hub.

Catalogs 2

Tiles List Ascending Descending

Filter catalogs by title

ML collection
An example of ML collection.

no-ML collection
An example of no-ML collection.

Items

Filter

No items available for this collection.

Powered by STAC Browser 3.0.2

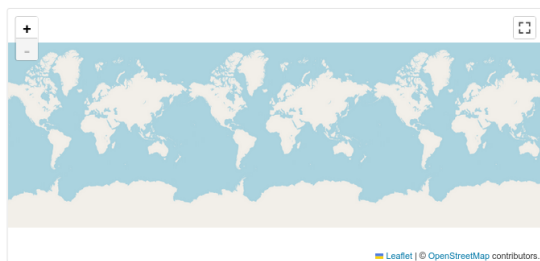
Figure 27: Analysis and processing resources catalog - Listing of available resources as collections

In Figure 28 a detailed view of one item of the ML collection, here the "LeNet Classifier", resource is shown. It details information e.g. about the model, the platform, the framework and much more

LeNet Classifier

Source Share Language: English

in FAIRiCUBE Hub analysis and processing resources catalog Up Collection Browse



Assets

- Input data used DATA JSON
- Model ML-MODEL-CHECKPOINT BINARY

Additional Resources

About this resource

- Reference link
- Example

Description

Multi-layer Convolutional Neural Network for image classification

Collection

ML collection
An example of ML collection.

General

Main Category	Deep Learning
Time of Data	4/4/2023, 12:00:00 AM UTC
Keywords	- classification - CNN - LeNet
Platform	Google Colab
Framework	Keras
Algorithm	LeNet
License	CC-BY-4.0
Use Constraints	no Constraint of Use

MI Model

MI Model Type	ml-model
MI Model Learning Approach	supervised
MI Model Prediction Type	classification
MI Model Architecture	CNN - Convolutional-Neural-Network
MI Model Training Processor Type	gpu
MI Model Training Os	linux

Powered by STAC Browser 3.0.2

Figure 28: Analysis and processing resources catalog – Example of a ML resource

The Figure 29 below provides an view on the browse option available in the dynamic Stac browser.

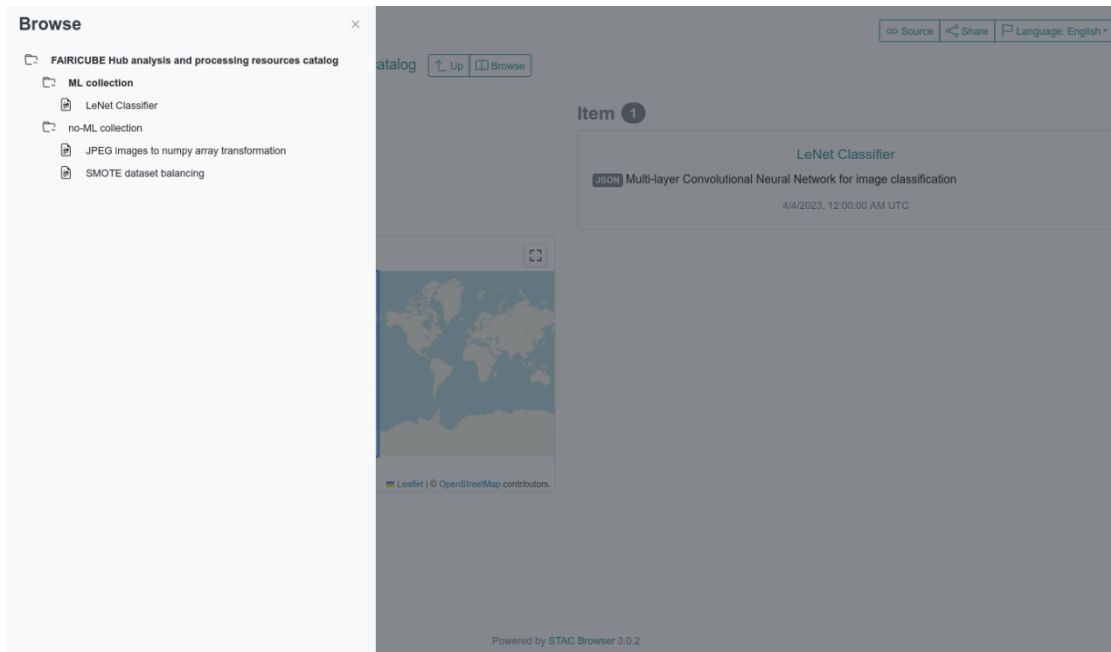


Figure 29: Analysis and processing resources catalog – Feature to browse available datasets

Figure 30 shows an example of a Non-ML resource, providing a detailed description of the resource as well as a button featuring direct data access.

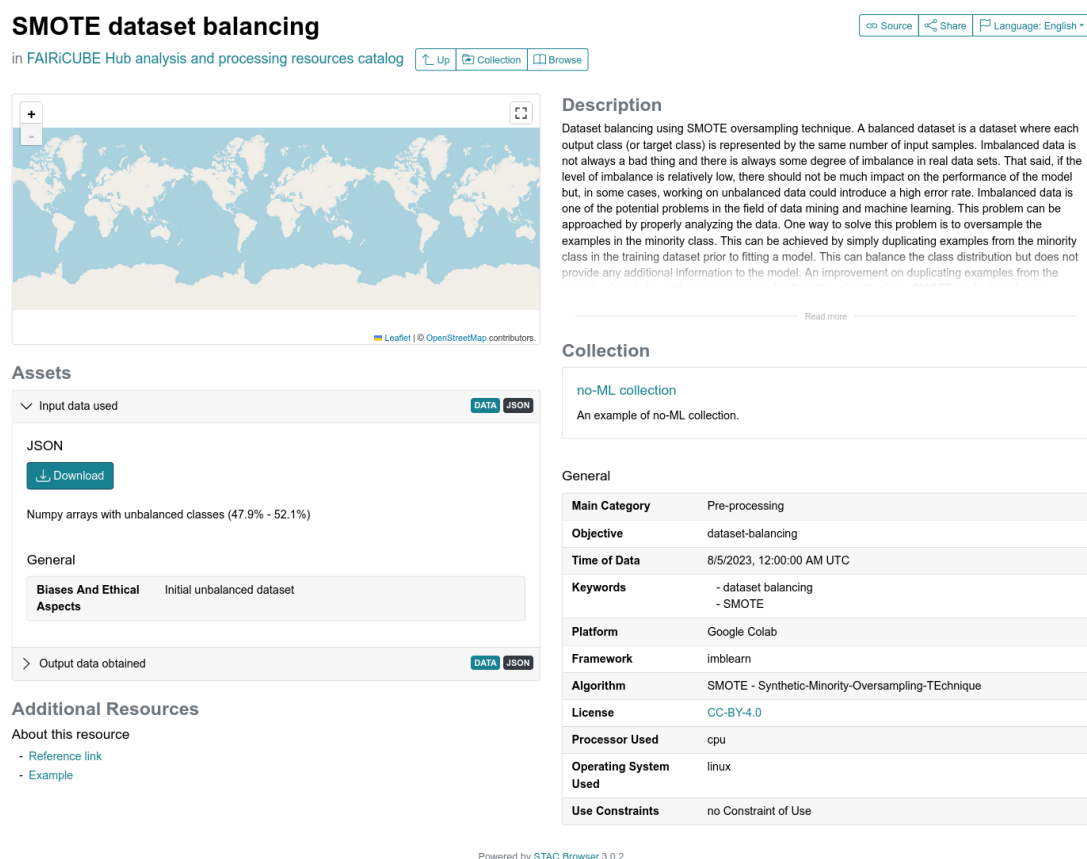


Figure 30: Analysis and processing resources catalog – Example of a Non-ML resource featuring direct data access



4 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.

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*"¹, which acts as the target platform for publication of the community collaboration platform. It allows project internal as well as external users to submit 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 documentation 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 the 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 accessible at: <https://faircube.readthedocs.io>. A screenshot of the landing page is shown below in Figure 31.

¹ <https://about.readthedocs.com/>



FAIRiCUBE-HUB Documentation

HOME

About FAIRiCUBE

- Datacubes
 - FAIR processing and analysing
 - Machine learning and datacubes
- Overview

DESIGN

- FAIRiCUBE Hub
- Context

USER GUIDE

- FAIRiCUBE Catalog
- FAIRiCUBE Lab
- FAIRiCUBE Lab Notebook Examples
- FAIRiCUBE Knowledge Base
- Datacubes
 - rasdaman datacubes
 - xcube utilization
 - Further information

ML-TOOLKITS

- ML-Toolkits Introduction

Next »

» Home » About FAIRiCUBE

Welcome to F.A.I.R. Information Cubes - FAIRiCUBE

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

The project's goal is to leverage power of Machine Learning (ML) operating on multi-thematic datacubes for a broader range of governance and research institutions from diverse fields, who are at present cannot easily access and utilize these potent resources.

The objective is the creation of a FAIRiCUBE Hub, a crosscutting platform and framework for data ingestion, provision, analyses, processing and dissemination, to unleash the potential of environmental, biodiversity and climate data through dedicated European data spaces.

FAIRiCUBE use cases address EU green deal action items, focusing on urban and regional scale. The use cases are:

1. [Spatial and temporal assessment of neighbourhood building stock](#)
2. [Biodiversity occurrence cubes](#)
3. [Biodiversity and agriculture nexus](#)
4. [Urban adaptation to climate change](#)
5. [Drosophila Genetics](#)

Datacubes

The interpretation of the term datacube in the EO domain usually depends on the respective context. It may refer to a data service such as Sentinel Hub, to some abstract API, or to a concrete set of spatial images that form a time-series.

Figure 31: Landing page of the Community collaboration platform



5 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:

- "Query Tool" is the engine of the KB. It allows to query KB resources based on predefined or customized queries.
- "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 their 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 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.

To be noted that a "Self-training Library", containing a set of links to web pages and project resources, appropriately selected and organised into categories, aiming to support understanding and reuse of the project outcomes and resources, originally designed to be included as an additional section of the Knowledge Base, has been subsequently moved into the FAIRiCUBE documentation section of the Community Collaboration Platform.

6 FAIRiCUBE GitHub Repository

FAIRiCUBE has setup and operates a GitHub repository (<https://github.com/FAIRiCUBE>) to manage the large amount of information, configurations, know-how and code collected and developed in the project. Currently this repository holds 24 sub-repositories.

Table 1: List of GitHub repositories used by FAIRiCUBE

browser	STAC browser configured for FAIRiCUBE (STAC-fastapi)
catalog	A repository of publicly available collections that are available for access through FAIRiCUBE Hub ² . Note that collections in this registry are available via FAIRiCUBE Hub, but owned and maintained by different providers.
collaboration-platform	Community Collaboration Platform repository
common-code	Common and useful code for all use cases
data-requests	This space is used for issues and documentation of data ingestion as part of FAIRiCUBE WP5 work.
FAIRiCUBE-Hub-issue-tracker	This space is used for issues and discussions on general FAIRiCUBE topics. In addition, issues are being utilized within a few of the other FAIRiCUBE Repos:
flux-config	This repository holds the teams and profiles configuration of EOxHub on the NILU AWS tenant available at https://eoxhub.fairicube.eu .
issues-yaml-generator	A tool that downloads GitHub issues from the data-requests repository and generate yaml files from the response markdown.
Knowledge-Base	This repository is intended for sharing our current understanding and getting feedback on the proposed architecture, structure and functionalities of the KB as well as related look & feel.
lessons-learnt ³	FAIRiCUBE lessons learnt consist of use cases challenges and related successes, failures, solutions and workarounds, documented using a dedicated template.
project-updater	A now obsolete python tool that updates number and text fields in Data Access project FAIRiCUBE from Inventory.xlsx using github's GraphQL API mutations.

² <https://fairicube.eu/>

³ <https://github.com/FAIRiCUBE/lessons-learnt>

resource-metadata	Manage information for processing/analysis resources, specifically: issue form to collect md requirements, issue template to manage codelists
Schemas	For some data provision options, additional schema files are required (e.g. when extending the Coverage data models as required, XSD files are required to document the extensions).
stac-browser	This is a Spatio-Temporal Asset Catalog (STAC) ⁴ browser for static catalogs. Minimal support for APIs is implemented, but it not the focus of the Browser and may lead to issues.
stac-GUI-backend	Repository for the stac-GU_backend developments
stac-GUI-front	Repository for the stac-GU_frontend developments
uc1-eodash-catalog	Template for creating eodash catalog repository
uc1-eodash-client	This is a template repository for configuring and deploying eodash-v5 ⁵
uc1-urban-climate	FAIRiCUBE Urban adaptation to climate change Use Case
uc2-agriculture-biodiversity-nexus	FAIRiCUBE Agriculture - Biodiversity Nexus Use Case
uc3-biodiversity-occurrence	FAIRiCUBE Occurrence - Biodiversity Use Case
uc3-drosophola-genetics	FAIRiCUBE drosophola-genetics Use Case
uc4-building-stock	FAIRiCUBE Building Stock Use Case
uc5-occurrence-cubes	Validation of Phytosociological Methods through Occurrence Cubes

⁴ <https://github.com/radianteearth/stac-spec>

⁵ <https://github.com/EOX-A/eodash-v5>

7 Access to external resources

External resources often provide data and/or information which is not readily available within FAIRiCUBE. However, such resources can be accessed as external resources, and their data/information holdings can be utilized. One such external resource, already integrated into the FAIRiCUBE setup, is the Sentinel Hub. The Sentinel Hub not only provides access to Sentinel data, but also provide methods and tools for working with the large amount of data available. Detailed information about the usage of this feature is provided at: <https://eurodatacube.com/documentation/choose-your-workflow>. This information will later be also provided directly within the FAIRiCUBE Community Collaboration Platform, accessible via the Knowledge Base.

7.1 On-the-fly data cube access: Sentinel Hub

RESTful APIs for seamless access to the data in ARD format.

- RESTful APIs for seamless access to the data in ARD format
 - Data access through a unified API
 - Whole archives of Sentinel, Landsat and Modis data available on-line
 - Any custom raster data can be added
 - Fast access, for real-time consumption and for intense compute processes such as machine learning
- More than data access
 - Package your algorithm with a request to get the results you need
 - Let us do the work but keep control over the processing
 - Cloud optimized and robust system, reducing processing time, and costs
 - No need for super computer or huge storage on user's side
- Cloud-agnostic
 - Currently operational at AWS, CreoDIAS and Mundi
 - Data from different sources is available in one place in the same manner
- Proven scalability
 - 100-200 million processed requests per month
 - Users around the globe

Detailed information about the usage of this feature is provided at:

<https://eurodatacube.com/documentation/choose-your-workflow#on-the-fly-data-cube-access-sentinel-hub>. This information will later be also provided directly within the FAIRiCUBE Community Collaboration Platform, accessible via the Knowledge Base.

7.2 Mass processing service: Sentinel Hub Batch Processor

One of the main features offered by Sentinel Hub is the mass processing EDC Sentinel Hub service with asynchronous response. This can be accessed via the EDC Sentinel Hub Batch Processing API. The service allows you as a user to:

- Request data at large scale – either spatial or temporal
- Run your algorithm for a whole continent
- Pre-process vast amount of data



- Execute, go for a coffee and have results ready

Detailed information about the usage of this feature is provided at: https://docs.sentinel-hub.com/api/latest/#/BATCH_API/batch_processor. This information will later be also provided directly via the FAIRiCUBE Community Collaboration Platform, accessible via the Knowledge Base.

7.3 Pre-generated data cubes with xcube

This provides a Python-based solution for generating, exploiting, and publishing data cubes in an easy way. This allows a user to utilize Sentinel, Landsat, and MODIS archives through Sentinel Hub as data source, and additional data sets, e.g. from ESA CCI or Copernicus Services. Detailed information about the usage of this feature is provided at: <https://eurodatacube.com/documentation/choose-your-workflow#pre-generated-data-cubes-with-xcube-python-based-solution-for-generating-exploiting-and-publishing-data-cubes>. This information will later be also provided directly within the FAIRiCUBE Community Collaboration Platform, accessible via the Knowledge Base.

7.4 Sentinel Hub Dashboard

The Dashboard provides access to the *Configuration* Utility tool, your *collections*, and the information about your usage of the services. You can also access your *user settings* and *billing information*. Further information is provided: <https://www.sentinel-hub.com/develop/dashboard>

7.5 Data uploading – Example: Bring Your Own COG API

Bring Your Own COG API (or shortly "BYOC") enables you to import your own data (or other external data not available directly) in Sentinel Hub and access it just like any other data you use. To be able to do so, the following conditions should be met:

- Store your raster data in the cloud optimized GeoTIFF (COG) format on your own S3 bucket in the supported region.
- Configure the bucket's permissions so that Sentinel Hub can read them.
- Import tiles using the Dashboard or API.

Further information is provided at: <https://docs.sentinel-hub.com/api/latest/api/byoc> and https://sentinelhub-py.readthedocs.io/en/latest/examples/byoc_request.html. This information will later be also provided directly within the FAIRiCUBE Community Collaboration Platform, accessible via the Knowledge Base.

8 The rasdaman Datacube Deployment

Datacubes are the natural data structure for space/time-varying phenomena, such as 1D sensor timeseries, 2D geo imagery, 3D x/y/t image timeseries and x/y/z geophysical data, 4D x/y/z/t atmospheric and ocean data, and so on. Among the advantages of datacubes are: per sensor / data source a single cube instead of zillions of files; homogenized semantic data description making them analysis-ready; a single paradigm across all dimensions, allowing cross-dimensional data fusion.

Constructor (formerly: Jacobs) University (CU) provides datacube management and analytics based on the rasdaman ("raster data manager") engine. In this section, first the positioning in the overall project landscape is explained, followed by a brief overview of rasdaman components, services, and functionality. Finally, the current status is summarized. The rasdaman deployment is part of the FAIRiCUBE Hub which offers the two technology pillars foreseen in the Grant Agreement:

"In order to assure that FAIRiCUBE functions across diverse technology providers, we have integrated some of this diversity within the consortium, including two major European experts on datacube standardization with their competing products. We believe that the specific expertise of these players complements each other, while at the same illustrating the power of standards to enable compatibility across systems:

- *Rasdaman has pioneered management and analytics of massive multi-dimensional arrays ("datacubes"), providing powerful tools to enable easy access and processing to these complex data sources*
- *EOX has made a name for itself enabling access to gridded data and workflow management runtime for Earth Observation services and apps for example in the Euro Data Cube (<https://eurodatacube.com/>)*

As both technology stacks utilize the OGC datacube standards CIS and WCS (plus WMS and WMTS – not datacube standards per se, but useful), data can easily be accessed from both sides, enabling users to utilize the frontend technology best suited to their requirements.

8.1 Rasdaman in the Project Landscape

In the overall orchestration the rasdaman components are integrated as highlighted in Figure 32. The yellow area on the right-hand side reflects the backend for datacube management and analytics, the bottom-left yellow area the frontend components of rasdaman (catalog, Jupyter notebooks, data requests, user-provided server-side code, etc.).

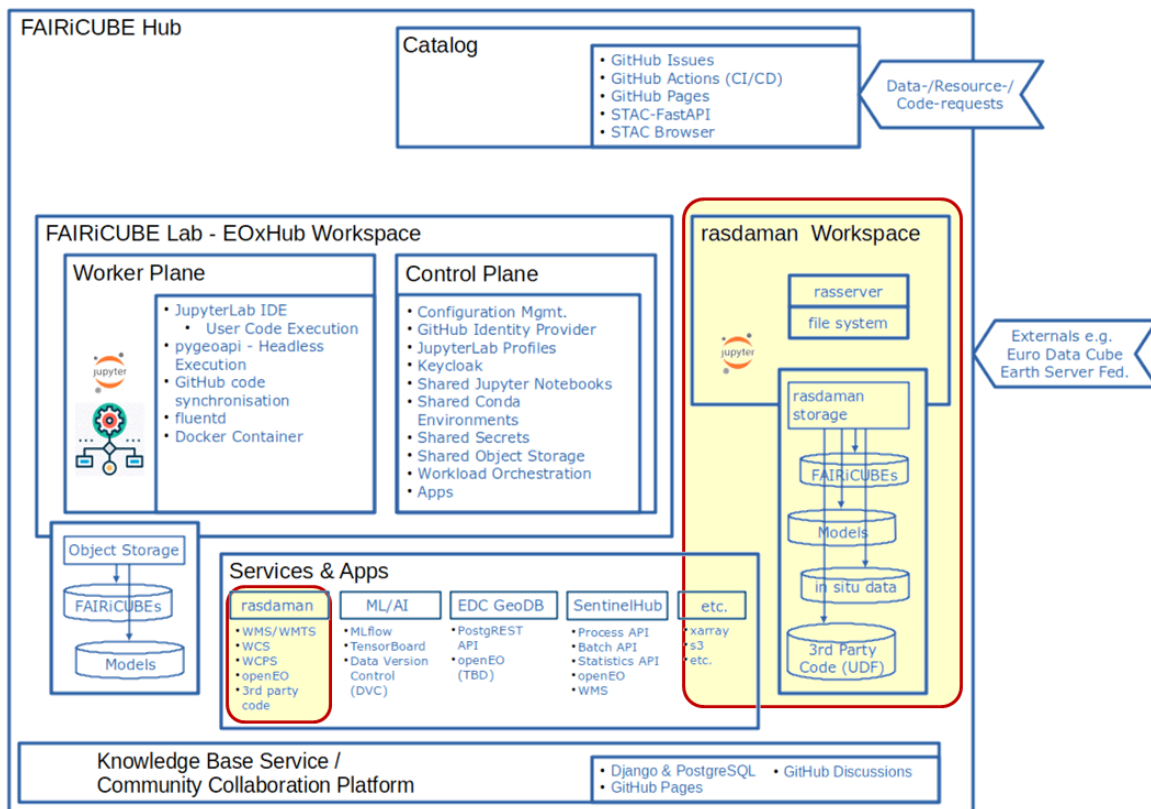


Figure 32: rasdaman in the FAIRiCUBE HUB Architecture, with rasdaman parts highlighted

8.2 Design Philosophy

The rasdaman approach is to support and advance FAIR data, ultimately pushing all technicalities “behind the curtain” so that users can focus on their real task on hand. In particular, users should NOT need to know and deal with

- Location: where in the federation do my data sit, and do I have to bring them together for fusion?
- Zillions of files: rather than a huge number of files with cumbersome file conventions to be understood, let there be one single spatio-temporal datacube per variable (such as temperature) or group of variables (such as x/y wind components).
- Divergent resolution, coordinate reference systems, and other low-level but important data properties: the system knows them anyway, so can generate a homogenized, analysis-ready view.

- ...and many more such aspects. Notably, research on how to make data more analysis ready is a field of active research and standardization, with CU contributing actively through publications⁶ and input & guidance in standards development⁷.

For achieving this, the highly divergent input data have to be analysed and understood first (see D5.2), which puts more effort on the data provider side – however, to the benefit of the data users who had to do this job by themselves in the past. Therefore, rasdaman puts much emphasis on capturing as much of the datacube semantics as possible, and utilizes that internally not only for easier-to-use service, but, e.g., also to optimize queries.

8.3 The rasdaman Engine

The central component rasdaman workspace is the rasdaman engine, a multi-parallel distributed database system specialized on the management and analytics of massive multi-dimensional arrays (rather than conventional tables). Access interfaces consist of a domain-neutral array query language (which ISO has adopted as Part 15 / Multi-Dimensional Arrays of the SQL standard⁸) and a geo-specialized datacube query language, WCPS, which additionally knows about the semantics of space and time, coined by CU and adopted by OGC⁹, ISO¹⁰, and EU INSPIRE¹¹ (optional).

In the server, incoming queries undergo a series of highly effective optimizations on data (configurable tiling, compression, etc.) and processing (logical and physical query optimization, parallelization, optimized distributed processing, semantic caching, etc.). The resulting queries are efficient close to the theoretical limit. Altogether, rasdaman's GreenCubes[®] green computing approach, when compared to, e.g., Python, makes evaluation faster and therefore more energy efficient. Server functionality can be extended with any external code, written in languages such as Python, C++, Java. This allows for faster evaluation (close to the data source) and providing functionality without disclosing the code. The complete engine stack is implemented by the rasdaman team, making rasdaman a genuine European technology asset.

8.4 Data & Ingestion

Based on the ingestion process (described in the WP5 deliverables D5.1 and D5.2), datacubes have been established as requested by the Use Cases. The first step in the ingestion process, as per common rule in FAIRiCUBE, is the creation of the Data Request via the WebGUI and feed it with all relevant

⁶ P. Baumann: On the Analysis-Readiness of Spatio-Temporal Earth Data and Suggestions for its Enhancement. Environmental Modelling and Software, Volume 176, 2024, <https://doi.org/10.1016/j.envsoft.2024.106017>

⁷ <https://myogc.org/go/coveragesDWG> and other worklines

⁸ <https://www.iso.org/standard/84807.html>

⁹ <https://www.ogc.org/standard/wcps/>

¹⁰ <https://www.iso.org/standard/83611.html>

¹¹ https://knowledge-base.inspire.ec.europa.eu/ogc-compliant-inspire-coverage-data-and-service-implementation_en



information about the dataset. Tracking and management of the data request, as well as the creation of the corresponding STAC metadata item, is done by using this GitHub mechanism. From this form a machine-readable description is prepared listing input data, the datacube to be generated and updated automatically thereafter, and other parameters. This so-called ingredients file is used by rasdaman to drive the automatic ingest process. After ingestion, a validation process is performed first by the CU team, then by the UC members to ensure the validity of the dataset. Only after successful finalization, the relevant GitHub issue is then closed. See D5.2 for details. In addition to these datacubes, further items had to be established, such as ML models stored server-side to be invoked in queries (see next).

8.5 Machine Learning

The models created by the Use Cases can be run directly inside the rasdaman server. Via the extensibility feature mentioned above the PyTorch framework (upon recommendation by WER) has been integrated into rasdaman so that now ML models build with PyTorch can be activated from a query, together with the input data for the inference.

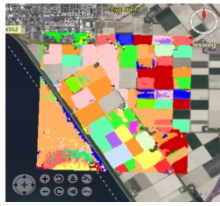
Implementation was done using the rasdaman extensibility mechanism of User Defined Functions (UDFs). This feature allows dynamic loading of external code at query time, where this functionality can be invoked seamlessly integrated as if it were a function of the query language. To this end, some adapter code has to be written, originally in the only supported language, C++.

For PyTorch, the initial idea was to use torch directly which is implemented in C++ and, therefore, allows for a direct coupling. It turned out, however, that PyTorch is not just a python wrapper but performs own data preparation, and so the torch coupling delivered wrong results. Consequently, it was necessary to add Python as an additional language for UDFs. This was accomplished in the project, leading to a dual result:

- Via a UDF function `nn.predict()`, models that have been registered with rasdaman can be passed to PyTorch as part of the query, and the result can be processed further in the query or delivered directly.
- In general, any Python code can act as a UDF in rasdaman. As Python is so common among data scientists such a Python coupling provides an additional advantage for them.

Figure 33 shows a sample model application to a region in the Netherlands using a model provided by WER. This close integration allows a very simple, interactive application of models to any datacube, any region, and any time – if desired, combined with the full set of WCPS operators.

Current status is that models, in order to be available in queries, have to be provided on the server in a designated directory by someone authorized (such as WER). In future, it is planned to generalize this to also allow passing of models on the fly, as part of a query. Further, models should migrate into the database and get attributes which allow users to meaningfully select their model of choice. One such criterion is the region (in space and time) of applicability of a particular model (minimizing the risk of applying a machine learning model on out-of-distribution data).



```
for $s2 in (Sentinel_2),
  $m in (CropModel)
return
  encode( nn.predict( $s2[...], $m ), "tiff" )
```

Figure 33: Sample model application as a query result (left) and the schematic WCPS query generating it

8.6 APIs & Clients

Manifold geo APIs are offered for clients, with strong support (OGC Reference Implementation, INSPIRE Good Practice, etc.), all compliant with OGC standards WMS, WMTS, WCS, WCPS and the OGC specs in progress, OAPI-Coverages and GeoDataCube. This enables a wide range of 3rd party clients to effortlessly tap into datacubes, such as map navigation (ex: Leaflet), virtual globes (ex: NASA WorldWind, Microsoft Cesium), Web GIS (ex: QGIS, ArcGIS), and high-end analytics (ex: GDAL, Python, R, openeo, ML). [\[12\]](#)

Additionally, the rasdaman FAIRiCUBE dashboard¹² offers a configurable frontend which can be set up, through a single JSON file, ranging from simple kiosks to powerful expert mode. Figure 34 shows the current configuration for the project, with several features in use:

- The simplest way to view data is through the blue buttons at ①. These are configured to display a dataset on the central background map; in the Figure, forest types have been selected for display, over ESA's Sentinel-2 true colour image.
- For datacube discovery, search and catalog widgets are available by clicking the esp. blue tiles ②. The hierarchically structured catalog is shown in the Figure as well, including all datacubes available to the project partners: Cubes ingested (cf. WP5), cubes made available via the EarthServer federation (such as the CoperniCUBE datacubes based on the Copernicus dataspace ecosystem), and cubes ingested for sister project AD4GD.
- A timeslider ⑤ is shown at the bottom of the dashboard when datacubes with a time axis are displayed. Any point in time or time range can be selected for display. Similar selection mechanisms (not shown here) exist for height/depth selection if datacubes shown have a vertical axis.
- Power users can enter queries in the WCPS query editor ③. In the example shown, a classification of coastal regions in the Netherlands is performed, highlighting (in red) areas with near-coast settlements at low elevation, hence endangered by rising sea levels ④. The query performs, invisible to the user, distributed data fusion between a local dataset (settlement) and EU_DEM sitting in CoperniCUBE.

As can be seen in the catalog window in Figure 34, currently the total of data amounts to about 1.6 PB. Further functionality is available in addition, and more widget types are being added continuously. For example, there is a juxtapose comparison of different layers of height/time/etc, and wind barbs (arrows) as used in aviation will become available soon.

¹² <https://faircube.rasdaman.com/rasdaman-dashboard>

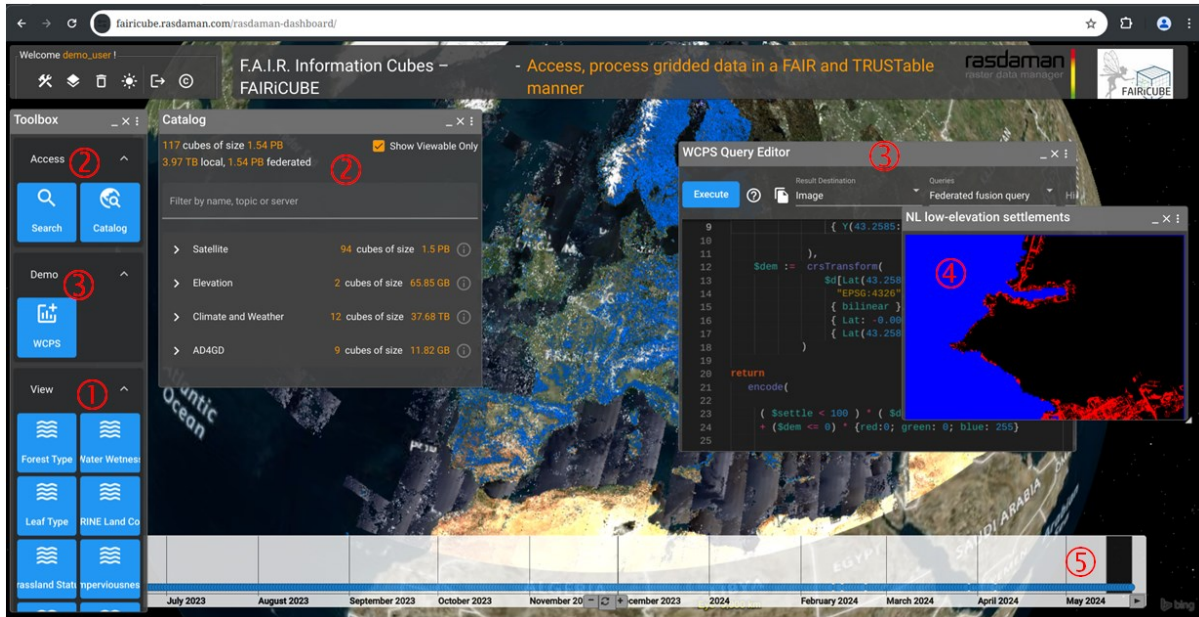


Figure 34: sample rasdaman dashboard screenshot (see text for explanation of numbers)

8.7 Development Environment

A datacube development ecosystem for developing python- and ML-based applications or doing zero-coding analytics was prepared for the Use Cases:

- Jupyter notebooks.** A fully fledged Jupyter Lab installation¹³ is available on the rasdaman FAIRiCUBE server, configured after the advice of the WER AI experts (Figure 35). It can be invoked from anywhere via its Web API, but Use Cases have also own logins to the server, to operate close to the data sources. Headless operation is possible, i.e. Jupyter notebooks can be start/run a command line/script.

¹³ <https://faircube.rasdaman.com/jhub/user/jupyter/lab>

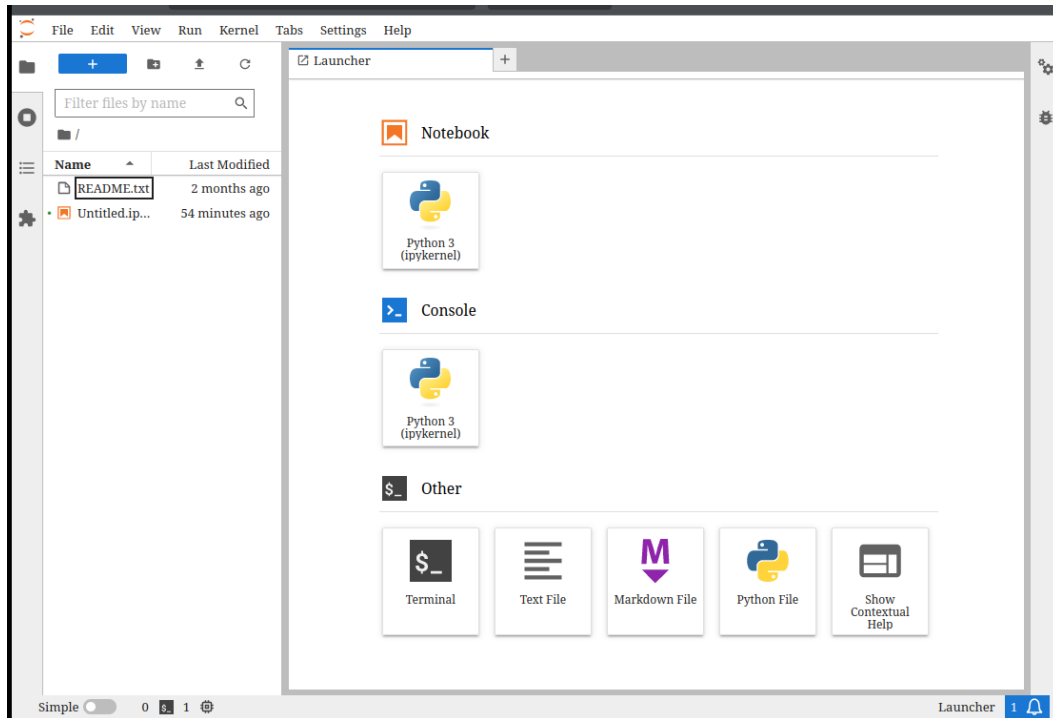


Figure 35: Jupyter lab environment on FAIRiCUBE rasdaman server

- OGC-compliant Web APIs.** As listed above, the rasdaman stack supports OGC WMS, WMTS, WCS, and WCPS, and additionally [INSPIRE-compliant data](#). Further, the not yet adopted OGC specifications of OAPI-Coverages and GeoDataCube are supported experimentally. An additional Web developer frontend¹⁴ (Figure 36) aims at supporting developers. It operates on the level of Web requests, allowing developers to compose requests in a point-and-click fashion and then copy the resulting request string.

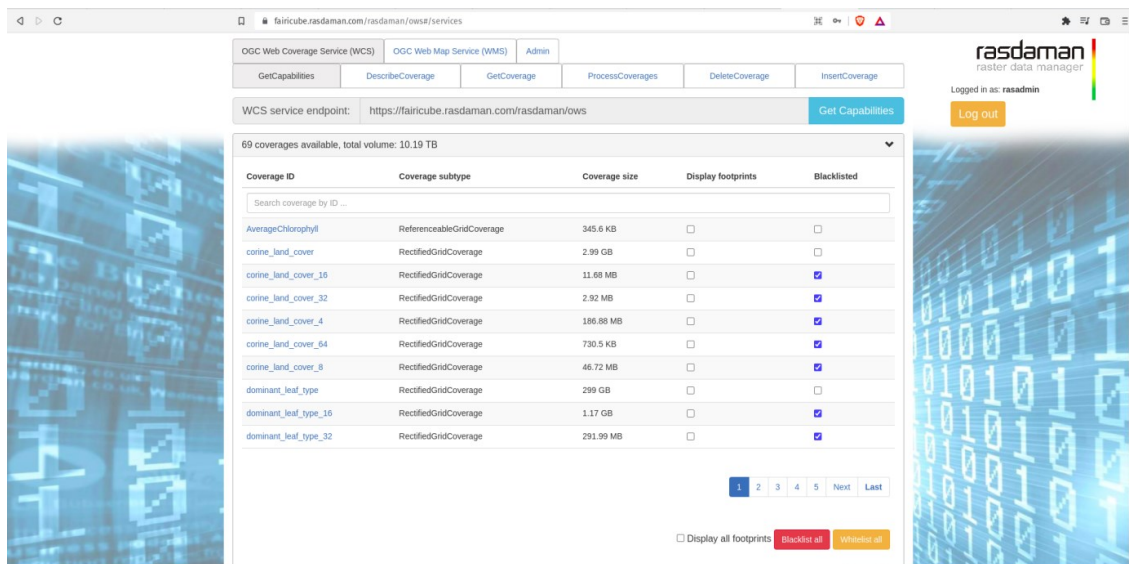


Figure 36: rasdaman Web GUI supporting interactive Web request generation

¹⁴ <https://fairicube.rasdaman.com/rasdaman/ows>

- **ChatCUBE.** Finally, based on the WCPS consulting CU gave over time, a chatbot named ChatCUBE was implemented using OpenAI for explaining WCPS (Figure 37). For experts, this enhances productivity (“what was the exact syntax?”); for non-experts it allows immersion at their own pace (“how can WCPS aggregate?”). ChatCUBE is available in beta on the rasdaman project landing page¹⁵.

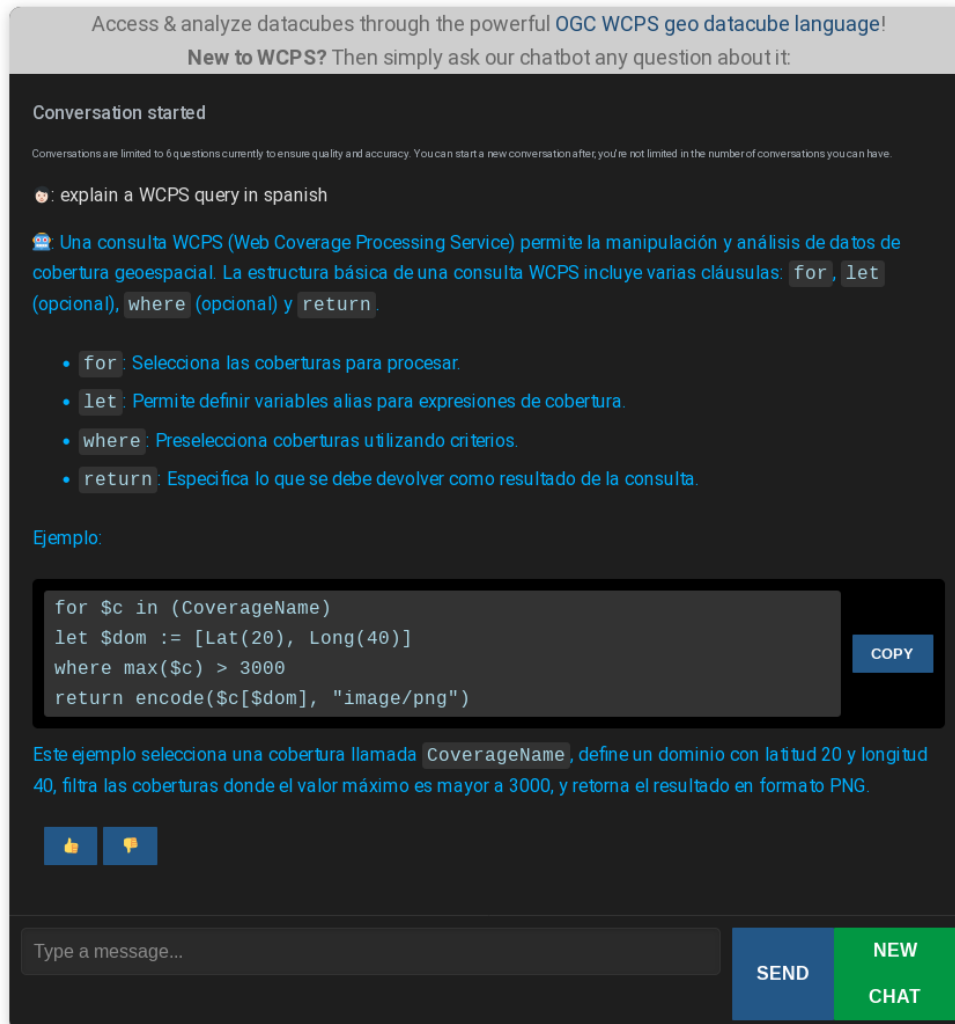


Figure 37: rasdaman Web GUI supporting interactive Web request generation

8.8 Support

Several webinars have been held to the FAIRiCUBE partners, addressing standards concepts, how to use datacubes in rasdaman, new functionality added, etc. GitHub Issues are used for exchange, but also direct email contact with the CU team.

¹⁵ <https://fairicube.rasdaman.com/>

8.9 Integration with the FAIRiCUBE Hub

At the time the report was due, the high-level concept for integration was given to CU by the coordinators: For each rasdaman datacube, a STAC record is generated referencing the datacube; conversely, in the datacube metadata a backlink to its STAC was foreseen. Based on this, refinement was done between CU and EOX which allowed a first demonstration at the review. Meantime, integration is growing, and routine procedures are in place.

8.10 Access Control

Commonly, access control contains the aspects of authentication (to verify authenticity of a user) and authorization (what an authenticated user is allowed to do). For authentication, rasdaman offers a built-in user/password management. An alternative mechanism, based on some external identity provider such as eduGAIN, has been explored in an OGC testbed earlier. In FAIRiCUBE, GitHub has been proposed as an option (after original deliverable deadline, so this is reflecting the time of update of this deliverable). This has to be explored and agreed further as a next step.

For authorization, the rasdaman database engine employs standard Role-Based Access Control (RBAC) enhanced with datacube-specific features. RBAC is based on privileges the system offers (in case of rasdaman datacubes: read datacube, write datacube, perform OGC W*S operations, etc.). Users can get assigned such privileges directly. Alternatively, roles can be defined which bundle privileges. Such roles then can get assigned to users, granting these users the privileges coming with the roles. Specifically, for datacubes, this concept common to databases has been extended to define privileges not only on complete datacubes, but on space/time regions inside. Just like in SQL, the query language provides statements so that the administrator (having appropriate privileges) can control such access rules.

At ingestion time, access rights get assigned to the objects created which henceforth determine the degree of availability to various user groups. As at this time it is still undecided under what regime the data will be available to the public, in rasdaman several roles have been defined in preparation. Project partners have logins, further ones can be added anytime. This way, access privileges can easily be adjusted once a common policy becomes available. Currently, access is denied to the general public, but this can be changed anytime as soon as the project has a general policy.

On top of these technical mechanisms, an overall project governance decision needs to be made on accessibility of the FAIRiCUBE datasets (obviously, with a bias towards open access). This is still pending at the time of this deliverable update. Currently, access is limited to project partners; this will be changed anytime from a technical perspective, depending on the outcome of the governance discussion.

8.11 Development outlook

The rasdaman pillar in FAIRiCUBE is fully operational and accessible to the partners, including ML integration and JupyterLab. In the strive for analysis-readiness, some further research needs have been spotted:

- Enriched semantics of datacubes by differentiating between numerical and categorical data; in standardization, this is foreseen in Sensor Web Enablement (SWE), and support for it was implemented in rasdaman. However, work remains to be done as (i) the SWE model part is rather complex, from a coverage perspective, and has certain shortcomings and (ii) more

information is to be added, see next. For example, the multispectral image example in the SWE Common example¹⁶ contains URLs that no longer resolve (such as <http://sweet.jpl.nasa.gov>; the SWEET Ontology has long been transitioned to BioPortal¹⁷), lacks unit of measure, and defines pixel intensity as having type „count“ while it should be „quantity“. However, in the SWE Common¹⁸ textual specification which actually comprises the standard, correct and complete examples are available, underscoring the necessity of both text and examples. On a side note, some units are quite complex, such as Radiance in Sentinel-2 data whose correct uom is "W.m-2.Sr-1.um-1" (watt per steradian per square metre), but this is due to the underlying physics. This complexity is not generally a concern as the data of interest to Use Cases tend to have simpler uom such as degree Celcius ("CEL") or meter per second ("m/s").

Significant progress has been achieved in FAIRiCUBE towards a consistent modelling, but this needs to be continued and documented as a best practice in the second project period.

- Due to erroneous interpretation of the Sensor Web Enablement (SWE) Common data models, relevant semantic information is lost. The element foreseen for provision of information about the variable (also referred to as Observable Property) the band provides is misused for provision of the datatype. This section of the encoding must be revisited and aligned with the requirements of the SWE Common data models.
- While the OGC coverage standard, CIS, through the utilization of SWE Common types is bound to the use of UCUM for provision of UoM, other alternatives have developed since the publication of the SWE Common models. An example of this is QUDT¹⁹, that takes a more semantic approach, and provides URIs for each concept. Altogether, this requires further work in standardization to establish and communicate common best practices.
- Based on particular requests for time handling, the standardized concept of gridded coverages was extended with information about "period of validity" and "temporal query granularity". This has been implemented, a publication is under way, and an interesting carry-over into spatial coordinates is being investigated which might solve the age-old discussion about "pixel-in-center" versus "pixel-in-corner" as well as "pixel-is-area" and "pixel-is-point".
- ML has been integrated, forming AI-Cubes™, however this needs further work. At this stage, one model has been received from WER, but more models are needed to validate the new service feature.

¹⁶ https://schemas.opengis.net/sweCommon/2.0/examples/image_data.xml

¹⁷ <https://bioportal.bioontology.org/ontologies/SWEET>

¹⁸ https://portal.ogc.org/files/?artifact_id=41157

¹⁹ <https://qudt.org/>