Climate Change Initiative Extension (CCI+) Phase 2
New Essential Climate Variables (NEW ECVS)
High Resolution Land Cover ECV (HR_LandCover_cci)

System Specification Document

(SSD)

Prepared by:

Università degli Studi di Trento
Fondazione Bruno Kessler
Università degli Studi di Pavia
Università degli Studi di Genova
Université Catholique de Louvain
Politecnico di Milano
LSCE
CREAF
University of Exeter
e-GEOS s.p.a.
Planetek Italia

Changelog

Issue	Changes	Date
1.0	First issue	14/01/2025
1.1	Secondo issue with applied RID	26/02/2025

Detailed Change Record

Issue	RID	Description of discrepancy	Sections	Change
1	ESA-01	Inputs: o Before 2020: Optical data from Landsat (e.g., Landsat 8) sourced from STAC catalogs, aggregated seasonally (e.g., 4 seasonal composites per year). o After 2020: Optical data from Sentinel-2 retrieved from STAC catalogs, aggregated monthly (12 monthly composites per year). It looks like that only Landsat data will be used before 2020 and S2 after 2020. This wasn't reported in the ATBD where Harmonized Landsat Sentinel-2 data were mentioned. S2 can be used from 2015 and HLS could be used to fill the gap after 2020 as well.	Operational Processing Chain Component / 3.2	The system is able to ingest all the mentioned satellite data. The use of such data is described in ATBD. The reference to the years are removed form SSD as ATBD is already an applicable document.

Sommario

1 Introduction

1.1 Executive summary

The **System Specification Document (SSD)** provides a detailed technical description of the High Resolution Land Cover (HRLC) system developed under Phase 2 of the ESA Climate Change Initiative (CCI). The system is designed to process and deliver global land cover data products by integrating multi-sensor inputs, advanced processing workflows, and cloud-based automation. Key system capabilities include robust metadata management, distributed processing for scalability, and temporal harmonization for historical consistency. The document outlines the system architecture, data management strategy, deployment schemes, and interface definitions, ensuring alignment with CCI program requirements and international standards such as Obs4MIPs and CF conventions.

1.2 Purpose and Scope

The purpose of this document is to define the **technical specifications** and components of the HRLC system. It provides a comprehensive description of the system's design, workflows, interfaces, and deployment strategies. The document serves as a reference for system developers, stakeholders, and operational teams to ensure consistent implementation, integration, and maintenance of the system.
The scope of the HRLC system includes:

- **Multi-sensor Data Processing:** Integration of Sentinel-1 (SAR), Sentinel-2 (Optical), and Landsat data for land cover classification.
- **Distributed Cloud-based Processing:** Leveraging Kubernetes, RabbitMQ, and Docker for scalable processing.
- **Metadata and Data Management:** Use of STAC, PostgreSQL (pgSTAC), and S3 for efficient data storage and discovery.
- **Training set management:** based on a new component that guarantees consistency across different training sets
- **Ancillary data management:** based on a new component that allow the organized storage of ancillary data
- **Temporal Harmonization:** Backward correction of historical data using post-2020 classification outputs.
- **Standards Compliance:** Adherence to Obs4MIPs, CF conventions, and CCI Data Standards for interoperability and usability.

1.3 Applicable documents

[AD1] CCI HR Technical Proposal, v1.1, 12/07/2023

[AD2] CCI-PRGM-EOPS-TN-13-0009 (Issue 2.3).

[AD3] D2.2 - ATBD (Algorithm Theoretical Basis Document) - ESA_CCI_HRLC_Ph2-D2.2_ATBD_v1.1

1.4 Reference documents

[RD1] The Global Climate Observing System: Implementation Needs, 01/10/2016, GCOS-200, Updated version in 2022 (GCOS-244) available at: https://library.wmo.int/idurl/4/58104

[RD2] User Workshop Report (UWR)

1.5 Acronyms and abbreviations

Acronym	Definition
CCI	Climate Change Initiative
CRC	Climate Research Community
CMUG	Climate Modelling User Group
CREAF	Centre de Recerca Ecològica i Aplicacions Forestals
ECV	Essential Climate Variables
ESM	Earth System Models

EVI	Enhanced Vegetation Index
GCOS	Global Climate Observing System
GDPR	General Data Protection Regulation
HR	High Resolution
LAI	Leaf Area Index
LBA	Large-Scale Biosphere-Atmosphere Experiment
LC	Land Cover
LCC	Land Cover Change
LCCS	Land Cover Classification System
LCML	Land Cover Meta Language
LCZ	Local Climate Zone
LSCE	Laboratoire des Sciences du Climat et de l'Environnement
MR	Medium Resolution
NDVI	Normalized Difference Vegetation Index
PFT	Plant Functional Type
RS	Remote Sensing
SFT	Surface Functional Type
STAC	SpatioTemporal Asset Catalog
OpenEO	Open Earth Observation API
SoW	Statement of Work
URD	User Requirements Document
VM	Virtual Meeting
WP	Work Package
PG	PostgreSQL Database
pgSTAC	PostgreSQL Extension for STAC
OGC	Open Geospatial Consortium
COG	Cloud Optimized GeoTIFF
S3	Amazon Simple Storage Service
CI/CD	Continuous Integration / Continuous Deployment
SAR	Synthetic Aperture Radar
FORCE	Framework for Operational Radiometric Correction for Environmental Monitoring
SNAP	Sentinel Application Platform
ML	Machine Learning
DL	Deep Learning
K8s	Kubernetes
API	Application Programming Interface
VPC	Virtual Private Cloud
ELB	Elastic Load Balancer

Table 1-1: Acronyms and Definitions

2 High-Level Description

2.1 System Objectives and Improvements Over Phase 1

Considering the evolution of technologies over the last 5 years, the new CCI-HRLC will maintain similar concepts with respect to Phase 1 system but will be redesigned also to face the requirements identified in the Technical Proposal. Here is a list of the Requirements

Title	REQ Code	Requirement	Improvement
Scalability and Distributed Processing	**HRLC-TR-12**	The system shall enable distributed task execution to ensure scalability across cloud resources.	Horizontal scalability through task orchestration with RabbitMQ and deployment automation via Kubernetes.
Multi-Sensor Data Handling	**HRLC-TR-11**	The system shall support the ingestion, processing, and harmonization of multi-sensor datasets such as Sentinel-1, Sentinel-2, and Landsat datasets.	Improved integration and harmonization of multi-sensor datasets (Optical and SAR) across the workflow.
Metadata Management and Harvesting	**REQ-3, HRLC-TR-13**	The system shall implement metadata management to manage, index, and expose metadata for discoverability and interoperability.	Enhanced metadata management using pgSTAC for indexing and STAC API for compliant metadata sharing.
Automation of Deployment and Processing	**REQ-19, HRLC-TR-12**	The system shall automate containerized deployment, testing, and versioning using GitLab CI/CD pipelines and Docker Registry.	Automated CI/CD pipelines ensure reproducible builds, reliable deployments, and version management.
Standardized Data Access and Distribution	**HRLC-TR-23, HRLC-TR-16**	The system shall provide standardized access to products via API for metadata distribution.	Enhanced discoverability and interoperability through STAC catalog and API.
Enhanced User Interaction	**REQ-19**	The system shall provide an interactive interface through for job execution, analysis, and visualization.	Simplified access for scientists and developers through interactive, user-friendly interfaces.
Robust Storage and Data Management	**HRLC-TR-14, HRLC-TR-17**	The system shall implement an S3-based Data Lake for storing data and support delivery to long-term archiving by the mean of CCI Portal.	Efficient storage, retrieval, and archiving of large datasets using cloud-native and CCI-compliant solutions.

Table 2-1: Main requirements summary derived from Technical Proposal

3 Logical Architecture

3.1 Overall Logical Architecture

3.1.1 Static View

This logical architecture integrates user interfaces, standardized APIs, scalable back-end services, reliable data storage, and distributed processing in the CCI-HRLC system for processing multi-sensor geospatial data. Here is the static view of the architecture

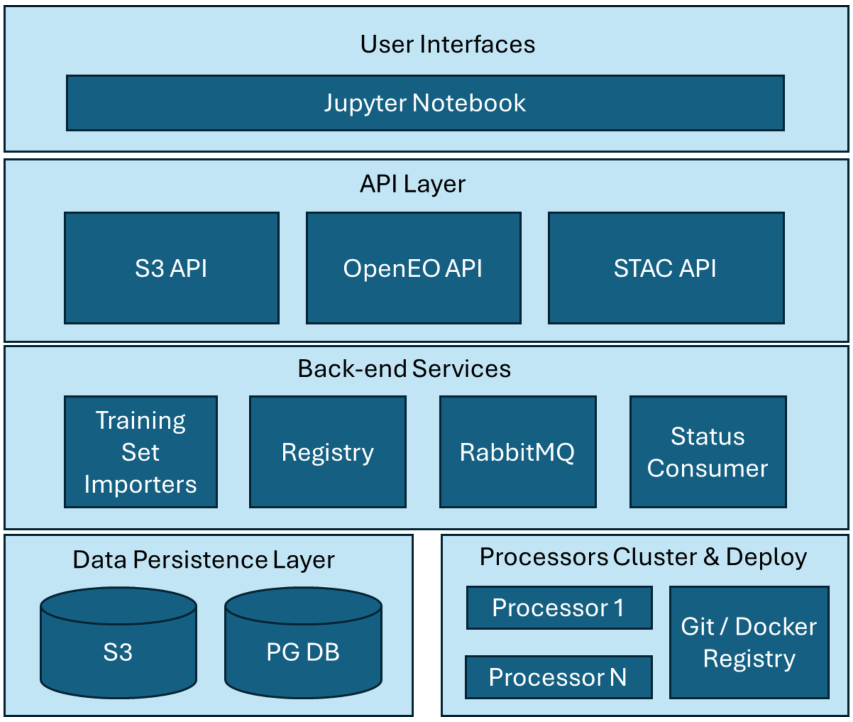


Figure 1: Logical Architecture of the CCI-HRLC system

1. User Interfaces

- The **Jupyter Notebook** serves as the primary interface for users (e.g., developers, scientists) to interact with the system.
- Through the notebook, users can:
    - Execute processing workflows.
    - Submit and monitor jobs.
    - Access and analyze results.

This interface simplifies access to APIs and back-end services, providing a user-friendly environment for running and testing workflows.

2. API Layer

The **API Layer** provides access to system services and data through standardized interfaces:

- **S3 API:** Manages interactions with the S3 storage system for retrieving and storing raw, intermediate, and final datasets.
- **openEO API:** Serves as the interface for orchestrating and executing geospatial processing tasks. It abstracts the complexity of distributed workflows and connects the user environment to processing backends.
- **STAC API:** Provides metadata discovery, search, and access for geospatial datasets, ensuring compliance with the **SpatioTemporal Asset Catalog** standard.

This layer enables seamless communication between the user interface and back-end components.

3. Back-end Services

The **Back-end Services** handle job management, orchestration, and system coordination:

- **Training Set Importers:** Import training datasets and ancillary features for machine learning models into the system.
- **Registry:** Manages the catalog of processors, their versions, and availability for job execution.
- **RabbitMQ:** Acts as a message broker to orchestrate and distribute processing jobs to the available processors.
- **Status Consumer:** Monitors the status of processing tasks, collecting feedback and ensuring reliable job execution.

These services enable scalable, fault-tolerant, and efficient management of processing workflows.

4. Data Persistence Layer

The **Data Persistence Layer** provides long-term and efficient storage for input data, metadata, and intermediate results:

- **S3:** Cloud-based object storage for handling large volumes of geospatial data (e.g., input imagery, processed outputs, and temporary results).
- **PG DB (PostgreSQL Database):** Stores metadata, including processing parameters, feature sets, and STAC-compliant metadata. The integration of pgSTAC ensures efficient querying and retrieval of geospatial records.

This layer ensures durability, traceability, and accessibility of system data.

5. Processors Cluster & Deploy [HRLC-TR-12]
This layer handles the execution and deployment of processing workflows:

- **Processor 1 to Processor N:** A scalable cluster of containerized processors that execute tasks such as optical preprocessing, SAR preprocessing, classification, and data fusion. These are orchestrated by RabbitMQ.
- **Git/Docker Registry:** Stores versioned container images for all processing components, ensuring reproducibility and portability of workflows.

The processors are deployed in a scalable, cloud-native environment (e.g., Kubernetes), enabling distributed execution of geospatial processing tasks.

3.1.2 Dynamic View

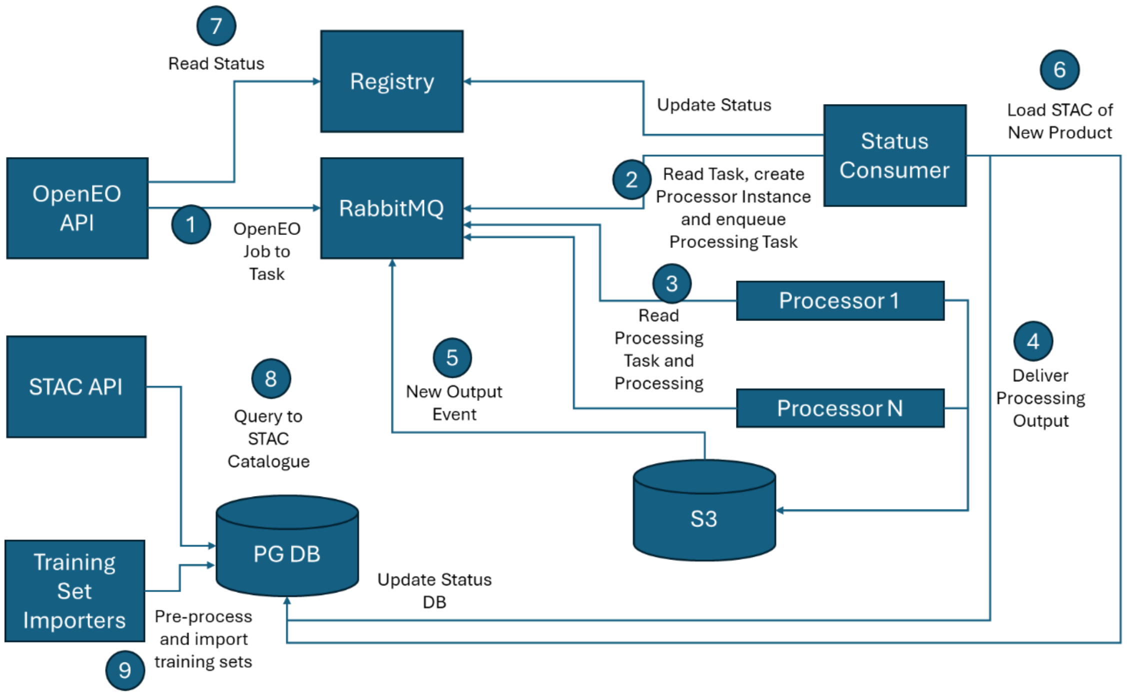


Figure 2: Logical Elements of the processing chain as they are organized in the CCI-HRLC system and their dynamic behaviour

1. **Notebook-Based User Interaction:** A Jupyter Notebook (or similar environment) serves as the primary interface for researchers and analysts. Through the notebook, users:
    - Invoke data queries and job submissions via the OpenEO API.
    - Discover and query datasets through the STAC API.
    - Initiate new processing tasks (e.g., running a land cover classification model).
    - Monitor job status and retrieve outputs.
    - Import the training set using specific back-end importers
2. **APIs:**
    - **OpenEO API:** Provides a standardized interface for initiating, managing, and retrieving the results of geospatial processing jobs. Users can compose workflows using a high-level Python client in the notebook.
    - **STAC API:** Allows the notebook user to search and discover geospatial datasets, browse metadata, and query time-series or spatial subsets. The STAC API's responses inform the user about available data suitable for processing tasks.

3. **Backend Orchestration and Messaging (RabbitMQ):**
   - **RabbitMQ** acts as a task broker, receiving processing job requests dispatched by the OpenEO API. It balances workloads and ensures distributed processors can handle tasks in parallel.
   - The **Registry** service keeps track of available processors and their versions. When a job is submitted, the OpenEO API consults the Registry (implicitly or explicitly) to identify which processors are available for specific tasks.
4. **Data Processing Components:**
   - **Processors** (e.g., Processor 1, Processor N) are containerized algorithms that consume tasks from RabbitMQ. These processors:
     - Fetch input data from S3 (or other object storage).
     - Process the data (e.g., apply land cover classification algorithms).
     - Write results (intermediate and final) back to S3.
   - **Training Set Importers** ingest reference and training datasets into the system's storage and databases. They can be triggered separately or integrated into the workflow when re-training or model tuning is required.
5. **Data Storage and Databases:**
   - **PostgreSQL (PG DB):**
     Stores metadata related to datasets, job parameters, processing results, and references to outputs stored in S3. It may also store training metadata and model versions.
   - **S3 Object Storage:**
     Holds large datasets, intermediate products, and final outputs. Processors read from and write to S3, ensuring a scalable storage layer that can handle large volumes of geospatial data.
6. **Status and Monitoring:**
   - **Status Consumer:** Listens to RabbitMQ for job status updates published by processors. It updates the Registry and PostgreSQL database with the current state of each job.
   - The notebook can periodically query the OpenEO API to obtain real-time status updates and logs.

Step-by-Step Interaction Flow:

1. **Data Discovery in the Notebook:**
   - The user runs Python code in the Jupyter Notebook to query the STAC API.
   - The STAC API returns a list of relevant datasets (e.g., Sentinel-2 imagery for a given AOI and time range).
   - The user selects specific STAC Items or Collections to process.
2. **Job Submission via OpenEO:**
   - Using the OpenEO client library in the notebook, the user composes a processing graph or workflow (e.g., cloud masking, spectral index calculation, classification).
   - The user submits the workflow to the OpenEO API, specifying input datasets (found via STAC), processing parameters, and desired outputs.
3. **Task Dispatch to RabbitMQ:**
   - The OpenEO API receives the request and places a corresponding job message into RabbitMQ.
   - The message contains references (STAC links, parameters) to input data in S3 and instructions on which processor(s) to use.
4. **Processor Execution:**
   - A suitable processor (Processor 1 … Processor N), registered in the Registry, fetches the job message from RabbitMQ.
   - The processor retrieves necessary data from S3 (as indicated by STAC references stored in PG DB).
   - The processor executes the job (e.g., land cover classification) and writes intermediate/final results back to S3.
5. **Status Updates and Completion:**
   - During processing, the processor sends status updates back to RabbitMQ, which the Status Consumer picks up.
   - The Status Consumer updates job state in the PG DB and possibly the Registry.
   - On completion, the final output references (e.g., STAC Items of the resulting products) are stored in the PG DB and accessible via STAC API.
6. **Result Retrieval in the Notebook:**

   - The user periodically queries the OpenEO API from the notebook to check job status.
   - Once the job is complete, the user retrieves the results (e.g., a classified raster in GeoTIFF format) from S3, or downloads them locally via the OpenEO API.
   - The user can also query the STAC API to integrate newly generated products into subsequent analyses.
7. **Retraining and Model Updates (Optional):**
   - If the user needs to improve the classification model, they can run Training Set Importers from the notebook to load new training samples.
   - The updated training data is stored in PG DB and S3.
   - The notebook triggers a re-training job (also dispatched via OpenEO → RabbitMQ → Processor), resulting in a new model version.
   - The Registry updates model references for subsequent processing tasks.

3.1.3 Security and Authentication Management

The system implements authentication and authorization mechanisms for integration point such as APIs. Users of the system interact with the system primarily through a Jupyter Notebook interface and S3 clients. In particular, the Jupyter Notebook must first authenticate against the identity provider before invoking the openEO and STAC APIs. Authentication tokens (e.g., OAuth 2.0 bearer tokens) are obtained and securely stored within the Notebook session, ensuring that all subsequent requests made to the openEO API are verified and authorized at the time of execution. Similarly, the STAC API enforces authenticated access to metadata catalogs and uses presigned URLs for data downloads from S3 storage. These presigned URLs are generated server-side, are short-lived, and provide read-only access to specific objects, preventing unauthorized data retrieval. All communications are transmitted over HTTPS.

3.1.4 Technological Stack

The proposed system architecture builds on established geospatial data standards and modern processing technologies. At the core, it leverages **STAC (SpatioTemporal Asset Catalog)** for consistent data discovery and metadata management and **openEO** to standardize the interface for executing and managing large-scale Earth Observation processing workflows. It integrates **Jupyter** notebooks for user-friendly, interactive development and analysis; **stac-fastapi** for implementing a performant, standards-compliant STAC API; **pgstac** for efficient metadata storage and retrieval in a PostGIS-enabled PostgreSQL database; and scalable object storage (e.g., **S3**) for handling large volumes of input and output data. Data are stored and processed using **Cloud Optimized GeoTIFF** (COG) formats for efficient, cloud-native reading and analysis. The processing environment is encapsulated in **conda-based processors**, ensuring reproducible and portable runtime environments with common geospatial Python libraries (such as GDAL, Rasterio, xarray, and geopandas). This technology stack, combined with robust orchestration and messaging frameworks, provides a flexible, scalable, and interoperable platform for Earth Observation data processing and analytics.

STAC (SpatioTemporal Asset Catalog): According to the STAC specification website, STAC provides a common language to describe geospatial datasets, making them easy to find, use, and integrate. STAC defines a flexible, JSON-based structure that can be read by both humans and machines, enabling indexing, discovery, and interoperability of spatial-temporal assets. By standardizing metadata and catalog organization, STAC ensures that satellite imagery, aerial photography, and other geospatial resources can be consistently searched, accessed, and combined, regardless of origin or platform.

openEO: As described on the openEO project website, openEO is an open API and corresponding ecosystem designed to connect Earth observation data processing platforms in a simple and interoperable manner. It allows users to process geospatial datasets using a unified interface and workflow, independent of the underlying data storage and processing infrastructure. Through openEO, developers and researchers can build workflows in familiar languages (such as Python), transparently execute them on diverse backends, and easily compare different processing environments.

Jupyter: According to Jupyter.org, Project Jupyter provides a web-based, interactive computing environment that supports over 40 programming languages. Its Notebook, Lab, and Hub tools enable data scientists and researchers to create, share, and reproduce data-driven narratives that combine live code, visualizations, and explanatory text. Jupyter's extensible architecture and strong community ecosystem make it a cornerstone in the modern data science workflow, promoting transparency, reproducibility, and collaborative research.

stac-fastapi: Described in its GitHub repository, stac-fastapi is a flexible and performant implementation of the STAC API specification. It leverages the modern FastAPI Python framework to provide a developer-friendly, high-performance, and scalable API for searching, browsing, and retrieving STAC-compliant catalogs and items. The project aims to simplify the deployment of STAC services, supporting various backends and extending STAC's core capabilities while maintaining compatibility with the STAC specification.

pgstac: Based on its GitHub page, pgstac is a PostgreSQL/PostGIS extension that optimizes storage and querying of STAC metadata within a relational database. It uses a JSONB-based schema to efficiently store and retrieve STAC Items, leveraging PostGIS for spatial queries and PostgreSQL's indexing capabilities for fast searching. pgstac enables robust filtering, faceted searches, and dynamic queries, making it an essential component for large-scale STAC-based data catalogs.

S3 (Amazon Simple Storage Service): From the AWS product page, Amazon S3 is a scalable, secure, and highly durable object storage service designed for modern cloud applications. It provides easy-to-use management features and an API-driven architecture, supporting a virtually unlimited amount of data storage. With low latency and high throughput, S3 allows developers, enterprises, and researchers to store and retrieve any amount of data from anywhere on the web, making it ideal for large geospatial datasets.

Cloud Optimized GeoTIFF (COG): As explained on the COG website, a Cloud Optimized GeoTIFF is a regular GeoTIFF file structured so it can be accessed efficiently over HTTP. By organizing internal data blocks and including an overviews structure, a COG allows clients to request only the required portions of the file instead of downloading the entire dataset. This structure makes streaming and visualization of large geospatial rasters possible directly from cloud storage, significantly improving performance and user experience.

Conda-Based Processors and Common Geospatial Python Libraries: Conda is a package, dependency, and environment management system that simplifies the creation of reproducible software environments. By using conda-based processors, developers encapsulate all dependencies and libraries into isolated, portable environments, ensuring consistent results across machines. Common geospatial Python libraries, such as GDAL, Rasterio, xarray, and GeoPandas, provide robust functionality for reading, writing, transforming, and analyzing geospatial data. These libraries support a wide range of operations—from file handling and coordinate transformations to complex geospatial analytics—empowering data scientists to build advanced Earth observation workflows.

3.2 Operational Processing Chain Component

3.2.1 Land Cover Chain

Taking into account the description in the ATBD document, the design of the Land Cover operational processing chain has the aim to describe the logical elements that will be developed and their integration. Here is a scheme that identify the Logical Elements of the processing chain as they are organized in the CCI-HRLC system.

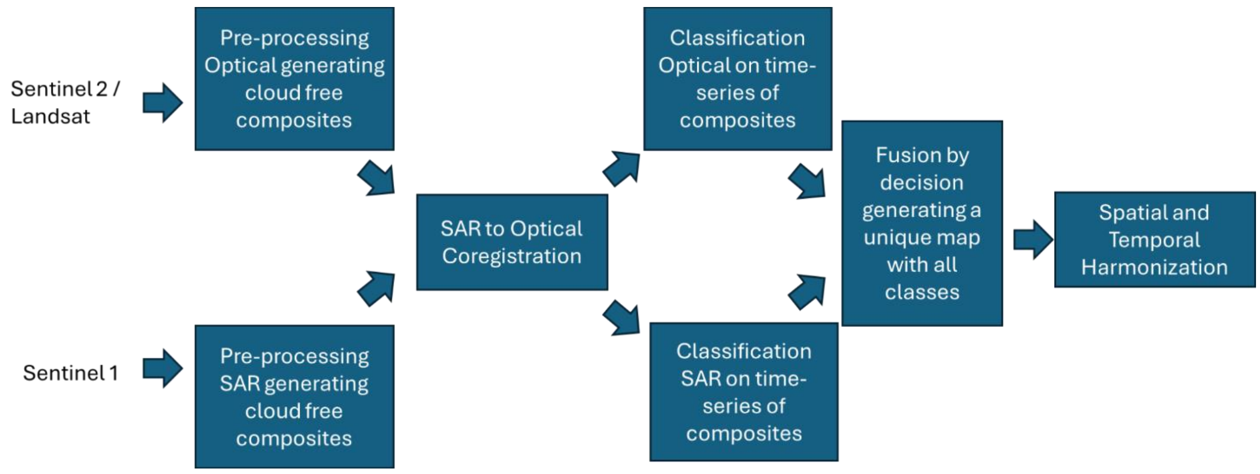

Sentinel 2 / Landsat
Pre-processing Optical generating cloud free composites
Sentinel 1
Pre-processing SAR generating cloud free composites
SAR to Optical Coregistration
Classification Optical on time-series of composites
Classification SAR on time-series of composites
Fusion by decision generating a unique map with all classes
Spatial and Temporal Harmonization

Figure 3: Logical Elements of the processing chain as they are organized in the CCI-HRLC system for the LC chain

Here is a detailed description of each step:

Optical Preprocessing (FORCE EO):

- **Description:** The optical preprocessing step leverages the FORCE Earth Observation (EO) processing framework to perform atmospheric corrections, surface reflectance retrievals, cloud masking, and radiometric calibrations. By running FORCE EO operations on raw optical satellite imagery (e.g., Sentinel-2, Landsat), this processor ensures that the resulting data is analysis-ready, with standardized reflectance values and consistent metadata. Throughout the process, intermediate data—such as cloud masks, pixel quality matrices —are generated and stored in S3.
- **Inputs (according to ATBD indications):**
    - Optical data from Landsat (e.g., Landsat 8) sourced from STAC catalogs, aggregated seasonally (e.g., 4 seasonal composites per year).
    - Optical data from Sentinel-2 retrieved from STAC catalogs, aggregated monthly (12 monthly composites per year).
- **Intermediate Data Structures:**
    - **Preprocessed Reflectance Composites (Cloud-Free):**
        - **Format:** Cloud Optimized GeoTIFF (COG)
        - **Naming Convention:** optical_{sensor}_{year}_{period}.tif (e.g., optical_sentinel2_2021_01.tif for January 2021 monthly composite, optical_landsat_2018_season1.tif for a pre-2020 seasonal composite)
        - **Metadata:** Accompanying STAC Items or STAC Item-level metadata in PostgreSQL/pgstac including sensor type, temporal period (monthly or seasonal), spatial extent, and processing parameters (cloud mask ratio, atmospheric correction method).
    - **Ancillary Layers (e.g., Cloud Masks, Quality Flags):**
        - Additional COGs (e.g., optical_sentinel2_2021_01_cloudmask.tif)
        - Linked via STAC Item properties and referenced in the Metadata DB.

SAR Preprocessing (ESA SNAP + Python Pipelines):

- **Description:** This step focuses on Synthetic Aperture Radar (SAR) data, employing the **ESA SNAP toolbox** for initial calibration, speckle noise filtering, and terrain correction. After these baseline operations, Python-based feature extraction pipelines produce advanced SAR metrics (e.g., backscatter intensity composites, coherence layers) that help characterize the target surfaces.
- **Inputs (according to ATBD indications):**
    - Sentinel-1 data from STAC catalogs, processed to normalized backscatter composites for the same monthly periods as Sentinel-2.
- **Intermediate Data Structures:**
    - **Preprocessed SAR Composites:**
        - **Format:** COG
        - **Naming Convention:** sar_sentinel1_{year}_{month}.tif (e.g., sar_sentinel1_2021_01.tif)
        - **Metadata:** STAC Items with properties describing polarization modes (e.g., VV, VH), incidence angles, and terrain correction flags.
    - **Derived Feature Layers (e.g., Coherence, Texture):**
        - Stored as separate COGs (e.g., sar_sentinel1_2021_01_coherence.tif), linked to main SAR composite via STAC relationships and referencing the same spatial and temporal extent.

Coregistration (Python Scripts):

- **Description:** The coregistration processor is responsible for precisely aligning optical and SAR datasets to ensure spatial consistency across different sensor modalities. Implemented as a Python script, this step employs robust feature-matching algorithms and geometric transformations to achieve pixel-level alignment of multisensor imagery.
- **Inputs (according to ATBD indications):**
    - Monthly Sentinel-2 composites and Sentinel-1 composites for the same time periods.
- **Intermediate Data Structures:**
    - **Coregistration Parameters:**
        - **Format:** JSON files stored in S3

        - **Naming Convention:** coreg_params_{year}_{month}.json containing information such as control points, geometric transforms, and root mean square error (RMSE) metrics.
    - **Coregistered Composites:**
        - **Format:** COG
        - **Naming Convention:** coreg_optical_sar_{year}_{month}.tif
        - Metadata updates in STAC catalog, referencing applied transforms and quality metrics.

Classification (Deep Learning/ML on Time-Series):

- **Description:** This stage applies deep learning and machine learning models to time-series stacks of preprocessed optical and SAR data. Using frameworks like TensorFlow, PyTorch, or scikit-learn, the classification processor identifies and labels land cover types, taking advantage of temporal patterns and multisensor inputs. Throughout the process, intermediate products—such as extracted feature vectors, model checkpoints, inference probabilities, and preliminary classification maps—are written to S3.
- **Inputs (according to ATBD indications):**
    - Seasonal Landsat-based composites for the entire year.
    - Combined monthly time-series of Sentinel-2 (optical) and Sentinel-1 (SAR, if available) composites.
- **Intermediate Data Structures:**
    - **Feature Stacks:**
        - Temporal stacks generated per AOI and year. For pre-2020: features_optical_landsat_{year}.nc (NetCDF or Zarr) bundling all seasonal composites. For post-2020: features_optical_sar_{year}.nc bundling all monthly composites.
        - These consolidated time-series files facilitate model ingestion and can include spectral bands, backscatter coefficients, NDVI/NDWI indices, and other derived features.
    - **Model Checkpoints and Probability Maps:**
        - **Format:**
            - Model Checkpoints: PyTorch/TensorFlow model weights stored as .pt or .h5 files in S3.
            - Probability Maps: COGs (e.g., class_probs_{year}_{monthOrSeason}.tif) containing per-class probabilities.
        - **Metadata:** STAC extensions or auxiliary JSON files documenting the model version, training data sources, and classes.
    - **Final Classification Maps:**
        - **Format:** COG with integer class codes (e.g., classified_map_optical_{year}.tif for pre-2020, classified_map_optical_sar_{year}_{month}.tif for post-2020).
        - Metadata includes class label definitions, accuracy metrics, and temporal references.

Data Fusion and Harmonization (Statistical Approaches in Python):

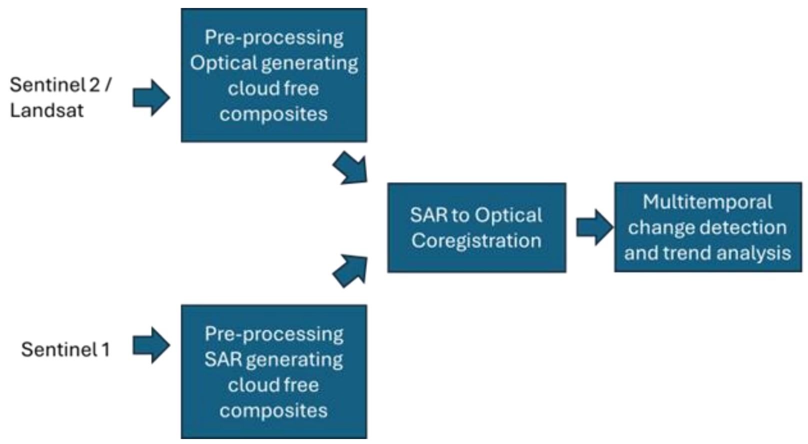
- **Description:** The data fusion step integrates the classification outputs and intermediate results from both SAR and optical domains into a unified, harmonized product. Based on statistical techniques and implemented in Python, this processor refines class boundaries, resolves conflicts between sensor-derived labels, and harmonizes spectral/temporal signatures into consistent land cover classes. During this process, intermediate fusion layers, harmonization parameters, and validation statistics are stored in S3, ensuring full transparency and providing a reference for subsequent updates. The processor also uses the Land Change Detection Maps.
- **Inputs (according to ATBD indications):**
    - After 2020: Both Optical and SAR classification maps.
    - Before 2020: Optical-only classification maps and, for harmonization, post-2020 classification layers used as temporal references.
    - Change Detection Maps
- **Intermediate Data Structures:**
    - **Fused Classification Maps:**
        - **Format:** COG
        - **Naming Convention:** fused_class_map_{year}.tif or fused_class_map_{year}_{region}.tif

        - Contains a unified class scheme derived from decision-level fusion (e.g., majority voting, statistical weighting).
    - **Harmonization Parameters and Backward Corrections:**
        - **Temporal Harmonization Tables:** JSON or CSV files (e.g., harmonization_lookup_{pastYear}_to_{refYear}.json) describing how classes from pre-2020 maps relate to post-2020 classes.
        - **Retroactive Classification Updates:** Updated classification maps for pre-2020 years that have been harmonized to match the post-2020 class definitions. Stored as COGs (e.g., harmonized_class_map_2018.tif).
        - These harmonized products ensure consistency across the entire time series, enabling long-term land cover change analysis.

All intermediate products are stored in S3 and referenced within a STAC-compatible metadata system. The PostgreSQL/pgstac database keeps track of product lineages, temporal ranges, sensor provenance, and processing metadata. This integrated approach ensures that the data pipeline remains transparent, reproducible, and easily navigable for downstream users and future reprocessing tasks.

3.2.2 Land Cover Change Detection Chain

Taking into account the description in the ATBD document, the design of the Land Cover Change Detection (LCCD) operational processing chain has the aim to describe the logical elements that will be developed and their integration. Here is a scheme that identify the Logical Elements of the processing chain as they are organized in the CCI-HRLC system.



Sentinel 2 / Landsat
Pre-processing Optical generating cloud free composites
SAR to Optical Coregistration
Multitemporal change detection and trend analysis
Sentinel 1
Pre-processing SAR generating cloud free composites

Figure 1: Logical Elements of the processing chain as they are organized in the CCI-HRLC system for LCCD chain

Here is a detailed description of each step:

Optical Preprocessing (FORCE EO):

- **Description:** The optical preprocessing step leverages the FORCE Earth Observation (EO) processing framework to perform atmospheric corrections, surface reflectance retrievals, cloud masking, and radiometric calibrations. By running FORCE EO operations on raw optical satellite imagery (e.g., Sentinel-2, Landsat), this processor ensures that the resulting data is analysis-ready, with standardized reflectance values and consistent metadata. Throughout the process, intermediate data—such as cloud masks, pixel quality matrices —are generated and stored in S3.
- **Inputs (according to ATBD indications):**
    - Optical data from Landsat (e.g., Landsat 8) sourced from STAC catalogs, aggregated seasonally (e.g., 4 seasonal composites per year).
    - Optical data from Sentinel-2 retrieved from STAC catalogs, aggregated monthly (12 monthly composites per year).
- **Intermediate Data Structures:**
    - **Preprocessed Reflectance Composites (Cloud-Free):**
        - **Format:** Cloud Optimized GeoTIFF (COG)
        - **Naming Convention:** optical_{sensor}_{year}_{period}.tif (e.g., optical_sentinel2_2021_01.tif for January 2021 monthly composite, optical_landsat_2018_season1.tif for a pre-2020 seasonal composite)
        - **Metadata:** Accompanying STAC Items or STAC Item-level metadata in

PostgreSQL/pgstac including sensor type, temporal period (monthly or seasonal), spatial extent, and processing parameters (cloud mask ratio, atmospheric correction method).

- **Ancillary Layers (e.g., Cloud Masks, Quality Flags):**
    - Additional COGs (e.g., optical_sentinel2_2021_01_cloudmask.tif)
    - Linked via STAC Item properties and referenced in the Metadata DB.

SAR Preprocessing (ESA SNAP + Python Pipelines):

- **Description:** This step focuses on Synthetic Aperture Radar (SAR) data, employing the **ESA SNAP toolbox** for initial calibration, speckle noise filtering, and terrain correction. After these baseline operations, Python-based feature extraction pipelines produce advanced SAR metrics (e.g., backscatter intensity composites, coherence layers) that help characterize the target surfaces.
- **Inputs (according to ATBD indications):**
    - To be defined
- **Intermediate Data Structures:**
    - **Preprocessed SAR Composites:**
        - **Format:** COG
        - **Naming Convention:** sar_{sensor}_{year}_{month}.tif (e.g., sar _2021_01.tif)
        - **Metadata:** STAC Items with properties describing polarization modes (e.g., VV, VH), incidence angles, and terrain correction flags.
    - **Derived Feature Layers (e.g., Coherence, Texture):**
        - Stored as separate COGs (e.g., sar_sentinel1_2021_01_coherence.tif), linked to main SAR composite via STAC relationships and referencing the same spatial and temporal extent.

Coregistration (Python Scripts):

- **Description:** The coregistration processor is responsible for precisely aligning optical and SAR datasets to ensure spatial consistency across different sensor modalities. Implemented as a Python script, this step employs robust feature-matching algorithms and geometric transformations to achieve pixel-level alignment of multisensor imagery.
- **Inputs (according to ATBD indications):**
    - Monthly Sentinel-2 composites and Sentinel-1 composites for the same time periods.
- **Intermediate Data Structures:**
    - **Coregistration Parameters:**
        - **Format:** JSON files stored in S3
        - **Naming Convention:** coreg_params_{year}_{month}.json containing information such as control points, geometric transforms, and root mean square error (RMSE) metrics.
    - **Coregistered Composites:**
        - **Format:** COG
        - **Naming Convention:** coreg_optical_sar_{year}_{month}.tif
        - Metadata updates in STAC catalog, referencing applied transforms and quality metrics.

Multitemporal Change Detection and Trend Analysis:

- **Description:** This stage focuses on detecting changes and trends over time using multitemporal analysis techniques applied to time-series of preprocessed optical and SAR data. The process identifies shifts in land cover classes, quantifies trends, and generates outputs for long-term monitoring and reporting. Advanced statistical methods, machine learning models, and geospatial analytics are utilized to compare data across multiple time steps and derive insights. Intermediate products such as change maps, trend statistics, and aggregated time-series results are stored in S3 for further use.
- **Inputs (according to ATBD indications):**
    - **Optical Data:** Preprocessed, cloud-free composites (Landsat for pre-2020, Sentinel-2 for post-2020).
    - **SAR Data:** To be defined
    - **Coregistered Data Stacks:** Combined optical and SAR datasets aligned spatially and temporally for analysis.
- **Outputs:**

- **Final Change Maps:**
  - **Format:** COG with integer codes indicating changes (e.g., final_change_map_{year1}_{year2}.tif).
  - **Metadata:** Class definitions, accuracy metrics, and temporal references documented in JSON or STAC extensions.

3.3 Metadata and Data Management Component

Metadata management is deeply connected to data management in a cloud-native approach. The implementation is based on a COTS component from robust projects that are part of the STAC ecosystem: stac-fastapi and pg-stac. This solution automates the process of ingesting geospatial data into an **S3-based Data Lake**, generating corresponding **STAC Items**, storing them in a **pgSTAC database**, and providing user access through a **STAC API**. Below is a simplified description of the behavior in the CCI-HRLC context.

---

1. **Data Ingestion to S3:**
   - Geospatial data files (e.g., satellite imagery, processed outputs) are uploaded to an **S3 bucket**, organized in a logical directory structure.
2. **Automatic STAC Item Generation:**
   - A monitoring process detects new data files added to the S3 bucket.
   - Metadata is extracted from these files, including spatial extent (bounding box), temporal information (acquisition date), and other relevant attributes.
   - **STAC Items** are generated for each file, conforming to the STAC specification, and include assets with links to the data in S3.
3. **Metadata Storage in pgSTAC:**
   - The generated STAC Items are ingested into the **pgSTAC database**.
4. **User Access via STAC API:**
   - Users access the API to search and retrieve metadata using spatial, temporal, and keyword filters.
   - The API provides links to the data assets in S3, allowing users to download or process the data as needed.

Below is the structure presented in a tabular format, designed to organize input data, final outputs and intermediate results for the described workflow. This structure is intended for an S3-like environment or a file-based storage system, with clear separation of raw inputs, intermediate steps, and final outputs. Each folder level includes a brief description of its contents.

Folder/Path	Description
/data/	Root directory for all data and processing outputs.
/data/raw/	Contains raw input data acquired directly from STAC catalogs.
/data/raw/sentinel2/{year}/{month}/	Raw Sentinel-2 imagery organized by year and month (post-2020, monthly composites).
/data/raw/landsat/{year}/{season}/	Raw Landsat imagery for pre-2020 seasonal composites.
/data/raw/sentinel1/{year}/{month}/	Raw Sentinel-1 imagery (post-2020) by year and month.
/data/preprocessed/	Outputs from optical and SAR preprocessing steps (cloud-free composites, corrected imagery).
/data/preprocessed/optical/{sensor}/{year}/{period}/	Optical preprocessed composites (COGs) and masks (e.g., optical_sentinel2_2021_01.tif, optical_landsat_2018_season1.tif).
/data/preprocessed/sar/{year}/{month}/	SAR preprocessed composites and derived features (e.g., sar_sentinel1_2021_01_coherence.tif).
/data/coregistration/{year}/{month}/	Coregistered imagery files and associated transform parameters (JSON) that align optical and SAR data.
/data/features/{year}/	Stacked feature datasets (NetCDF/Zarr) for time-series classification (e.g.,

	features_optical_landsat_2018.nc).
/data/models/	Directory for ML/DL model training artifacts.
/data/models/checkpoints/	Model checkpoint files (.pt, .h5) for classification models.
/data/models/metadata/	JSON/YAML files describing model versions, training parameters, and training datasets.
/data/classification/{year}/	Classification probability maps, preliminary classification results (COGs), and per-class metrics for the given year.
/data/classification/{year}/probabilities/	Probability map COGs (e.g., class_probs_2021_01.tif) for each period's classification.
/data/classification/{year}/final/	Final classified maps (COGs) per year and period (e.g., classified_map_optical_sar_2021_annual.tif).
/data/fusion/{year}/	Results of the data fusion step, including fused classification maps and intermediate decision metrics (COGs, JSON).
/data/changedetection/{year}/	Change detection maps
/data/harmonization/	Harmonized classification products that align pre-2020 and post-2020 class definitions.
/data/harmonization/{pastYear}/	Harmonized classification maps (COGs) and lookup tables (JSON) indicating how past classes map to modern classes.
/data/metadata/	Ancillary metadata (e.g., STAC Items, processing logs, and parameter files) linked to the PG/pgstac DB entries.
/data/metadata/stac/	STAC JSON files and catalogs referencing all products for discoverability and interoperability.
/data/logs/	Processing logs, error reports, and audit trails for debugging and reproducibility.

Table 3-1: DataLake structure to maintain the intermediate data and metadata

This structure ensures each step's outputs are logically grouped, easily referenced, and traceable through STAC metadata and naming conventions. It allows reproducible, modular, and scalable data handling across multiple years, sensors, and processing workflows.

3.4 Training Set and Ancillary Data Management Component

The Training Set and Ancillary Data Management Component is in charge of the import and archiving of the training set and charge of retrieving and archiving (if applicable) the ancillary data useful to generate new data sets. The elements of this component are:

- PostgreSQL DB Docker: it is responsible for hosting a database management system named PostgreSQL with PostGIS extension to manage geospatial data;
- Jupyter Docker: it is responsible for hosting the Python scripts to import the training set data and ancillary data and to interface with the external components.

These elements are packaged in Docker images..

The following figure shows the components diagram with each docker image and the external interface to manage the import of the Training Set and Ancillary Data into the system.

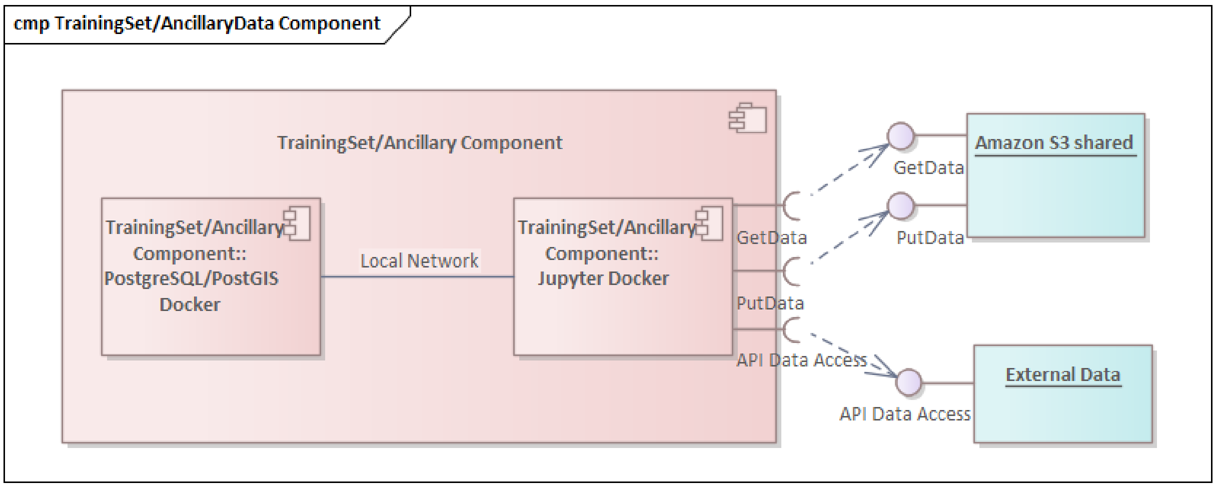

cmp TrainingSet/AncillaryData Component
TrainingSet/Ancillary Component
TrainingSet/Ancillary Component:: PostgreSQL/PostGIS Docker
Local Network
TrainingSet/Ancillary Component:: Jupyter Docker
GetData
PutData
API Data Access
GetData
PutData
Amazon S3 shared
API Data Access
External Data

Figure 4: Training Set Component

The external interfaces are the following:

- GetData: to retrieve new training set data stored in the Amazon S3. The import script will access this folder and start the import of the training set data according to the structure defined in the 3.4.2.1 sections.
- PutData: to import ancillary data (if not available API and according to the data license) in Amazon S3 storage
- API Data Access: to retrieve the ancillary data through API Data Access (if available) and store metadata into the component.

3.4.1 Ancillary Data Management

3.4.1.1 Ancillary Data

The ancillary data are data that are needed for the processing of optical and SAR data, different from the training sets and the optical and SAR data itself. Ancillary data are used in one or more tasks among training, inference, comparison, and post-processing. The majority of them are raster data.

The ancillary data can be provisionally summarised as:

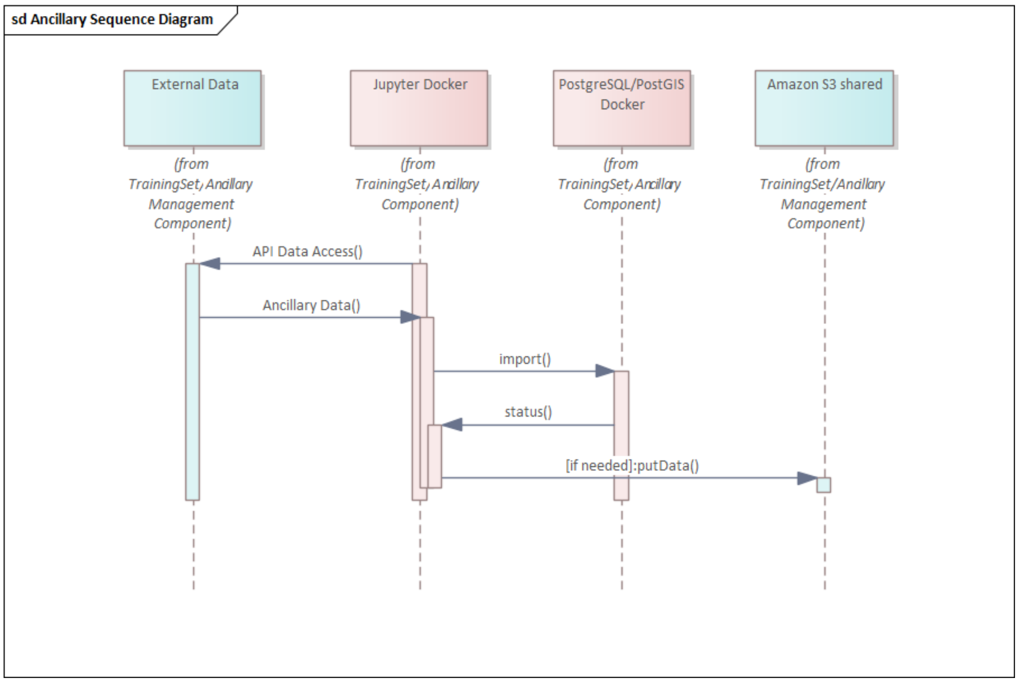
- **Digital Elevation Models (DEM)**: Copernicus DEM 30 m, raster, used in support of training/inference;
- **Land cover (LC)**:
  - ESA WorldCover, raster, used in support of comparison/post-processing, and possibly training;
  - GLC_FDS, raster, used for comparison;
  - MapBiomas Amazonia, used in support of comparison/post-processing, and possibly training;
  - CCI Circumarctic, raster, used in support of comparison/post-processing, and possibly training;
  - ESA GlobPermafrost, raster, used in support of comparison/post-processing, and possibly training;
  - CCI Land Cover , raster, used in support of comparison, and possibly training;
  - CLMS Dynamic Land Cover, used in support of comparison, and possibly training;
- **Other**:
  - Global Surface Water, raster, used in support of post processing;
  - Global Human Settlement, raster, used in support of comparison/post-processing;
  - Map of Land Cover Agreement, raster, used in support of training/comparison.
  - GLanCE, vector, used in support of training.

3.4.1.2 Ancillary Data Organisation and Integration

Two approaches are foreseen to retrieve the ancillary data:

1. For those data for which a Machine-to-Machine (M2M) approach is available, the retrieving and access is done through Application Programming Interfaces (APIs) and protocols (e.g.: S3 (Simple Storage Service) to allow the request and retrieve of the data programmatically;
2. For those data for which a M2M approach is not available (e.g.: data only accessible through static HTML pages, etc.), the data is manually retrieved and accessed once according to the data license, and stored

in a shared S3 storage.



sd Ancillary Sequence Diagram
External Data
Jupyter Docker
PostgreSQL/PostGIS Docker
Amazon S3 shared
(from TrainingSet/Ancillary Management Component)
(from TrainingSet/Ancillary Component)
(from TrainingSet/Ancillary Component)
(from TrainingSet/Ancillary Management Component)
API Data Access()
Ancillary Data()
import()
status()
[if needed]:putData()

Figure 5: Ancillary Data Sequence Diagram

The metadata will be stored in the Database and the data will be eventually stored according to a structured folder. The Data organization is TBD.

3.4.2 Training Set Management

3.4.2.1 Training Set Data

The training set data that will be imported into the component will be points and polygons. They are derived from photointerpreted and map-agreement sources.
The folder in S3 will be structured as follows:

- <area_name>
    - <area_name>_<type>_<ROI>_<year>_<source>_<provider>
        - <area_name>_<type>_<ROI>_<year>_<source>_<provider>.cpg
        - <area_name>_<type>_<ROI>_<year>_<source>_<provider>.dbf
        - <area_name>_<type>_<ROI>_<year>_<source>_<provider>.prj
        - <area_name>_<type>_<ROI>_<year>_<source>_<provider>.shp
        - <area_name>_<type>_<ROI>_<year>_<source>_<provider>.shx
        - <area_name>_<type>_<ROI>_<year>_<source>_<provider>.qml
        - <area_name>_<type>_<ROI>_<year>_<source>_<provider>.geojson

Where:

- <area_name>: indicates the toponyms name of the data like africa, amazonia and Siberia;
- <type>: indicates if the data is historic or static
- <ROI>: indicates the Region of Interest
- <year>: indicates the year of the data
- <source>: indicates if the source is photo-interpreted or map-agreement
- <provider>: indicates the provider like Polimi (Milan Polytechnic), UniPV (Padova University), UniTN (Trento University)

3.4.2.2 Database Model

The Training Set database is based on a relational database management system (PostgreSQL) with spatial extension PostGis for managing the training data defined in section 3.4.2.1. This database is composed of the following tables:

- *provider*:
- *trainingprovider*

- trainingmetadata
- traniningset
- spatialrefsys

The tables should be filled with the data coming from the import script described in section 3.4.2.3. The following figure represents the model of the relational database for the TrainingSet DB Model.

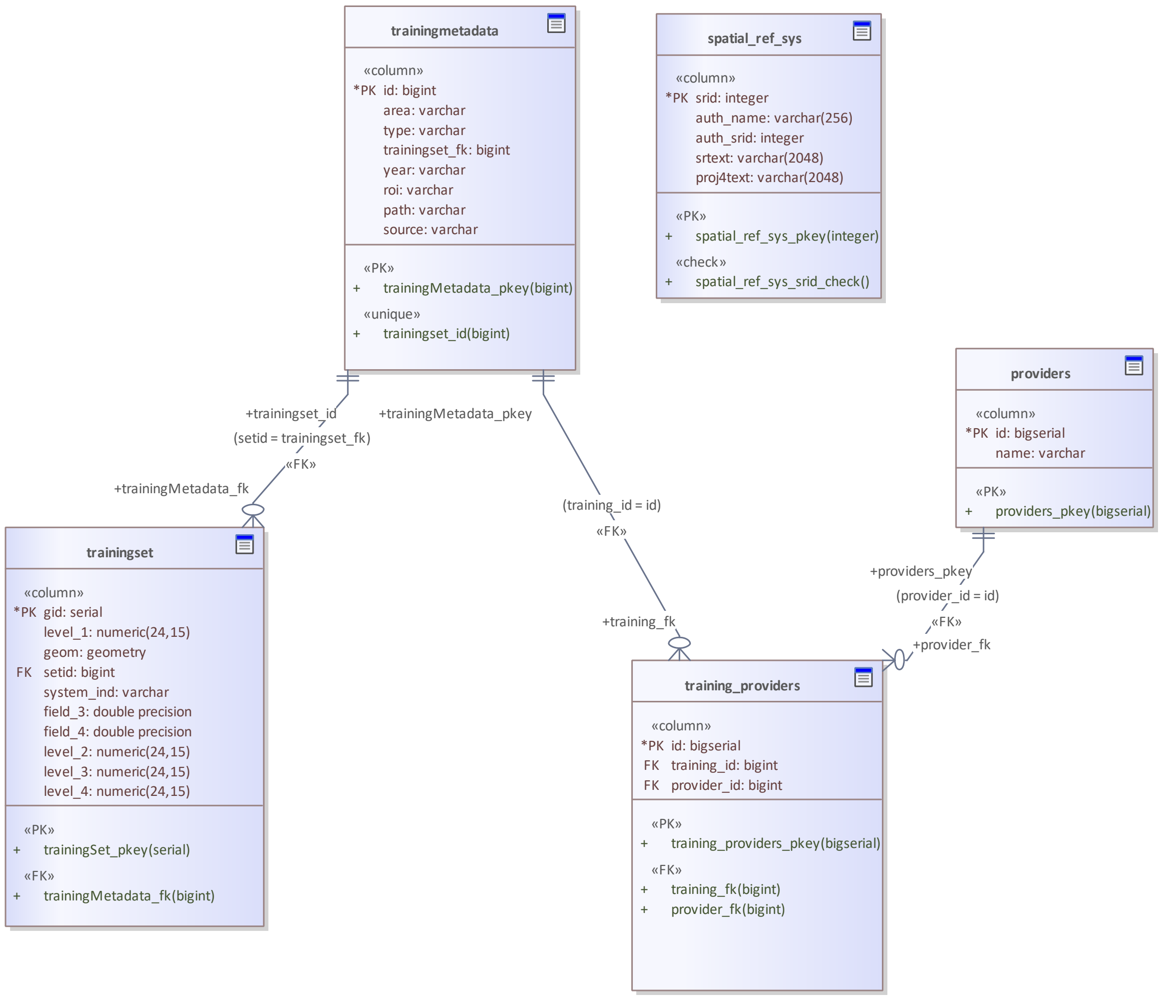


Figure 6: TrainingSet DB model

3.4.2.3 Import Data

The training data set defined in section 3.4.2.1 shall be imported into the training DB through a Python script. This Python script allows the extracting and loading of the data and metadata in the related tables.

During this phase, the Jupyter Notebook can help to check the status of the importing.
The workflow to import the data internal to the system will be the following:

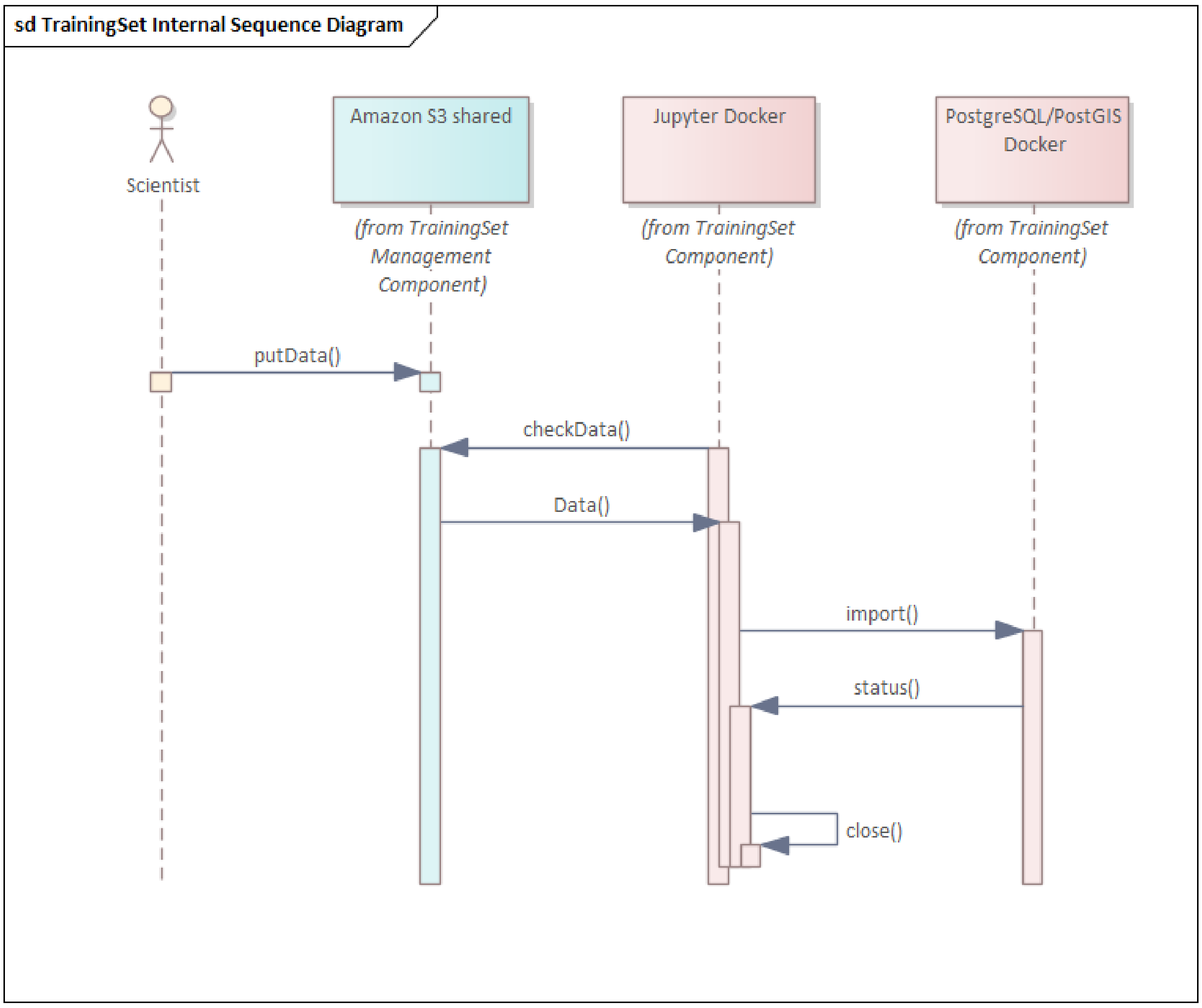

sd TrainingSet Internal Sequence Diagram
Scientist
Amazon S3 shared
(from TrainingSet Management Component)
Jupyter Docker
(from TrainingSet Component)
PostgreSQL/PostGIS Docker
(from TrainingSet Component)
putData()
checkData()
Data()
import()
status()
close()

Figure 7: Training Set Sequence Diagram

4 Interface Analysis

4.1 Internal Interfaces

The table below outlines the primary internal interfaces in the CCI-HRLC system. Each interface is identified by a unique code and describes the communication channel between components, including which source and destination it connects, the protocol used, and the data format. This ensures a clear understanding of how different parts of the system interact, facilitating maintenance, scalability, and interoperability.

Code	Source	Destination	Protocol	Format	Description
CCI-HRLC-INT-IF-000001	Jupyter Notebook	openEO API	HTTPS (REST)	JSON	Users interact with the processing system through a Jupyter Notebook. Requests to start, monitor, or stop processing tasks are sent to the openEO API in JSON.
CCI-HRLC-INT-IF-000002	Jupyter Notebook	STAC API	HTTPS (REST)	JSON (STAC)	The notebook queries the STAC API to discover and select geospatial datasets. The STAC API responds with JSON-formatted metadata conforming to the STAC standard.
CCI-HRLC-INT-IF-	Jupyter Notebook	PostgreSQL	DB Native	Binary	The notebook ingest the data from S3 to the training DB. Training data and associated labels are ingested into

000003					pgstac. The importers write dataset references, features, and annotations as JSON to facilitate model training.
CCI-HRLC-INT-IF-000004	openEO API	RabbitMQ	AMQP	JSON Messages	Once a processing job is defined, the openEO API sends job instructions to RabbitMQ message queues in JSON format for asynchronous task distribution.
CCI-HRLC-INT-IF-000005	RabbitMQ	Processors	AMQP	JSON Messages	Processors consume queued tasks from RabbitMQ, receiving job parameters and references in JSON and acknowledging or sending updates back into the queue system.
CCI-HRLC-INT-IF-000006	Processors	S3 Storage	HTTPS (S3 API)	COG, JSON	Processors read and write intermediate and final products to S3. Output data, typically Cloud Optimized GeoTIFFs (COGs) and associated metadata (JSON), are transferred over HTTPS.
CCI-HRLC-INT-IF-000007	Processors	PG DB (pgstac)	PostgreSQL Wire	SQL/JSON	Processors update, read, and write records in the PostgreSQL database with pgstac extension using SQL queries. Metadata and processing parameters are stored as JSON.
CCI-HRLC-INT-IF-000008	STAC API	PG DB (pgstac)	HTTPS (REST), PG	JSON (STAC), SQL	The STAC API interacts with pgstac to store and retrieve STAC Items and Catalogs. STAC metadata is ingested into pgstac and queried via SQL.
CCI-HRLC-INT-IF-000009	STAC API	S3 Storage	HTTPS (S3 API)	COG, JSON	The STAC API references data stored in S3, enabling direct download or streaming of geospatial products in COG format, along with STAC metadata in JSON.

Table 4-1: Internal Interfaces of the CCI-HRLC system

4.2 External Interfaces

The following table provides the list of external interfaces dedicated to the integration with Online Repositories ([HRLC-TR-14], [HRLC-TR-16]) and Data Distribution via OGC and STAC Services ([HRLC-TR-23]). The two main objectives in terms of external interfaces are:

- **Integration with Online Repositories** ([HRLC-TR-14], [HRLC-TR-16]): The system connects to external repositories either through direct integration with systems like **NASA Earthdata** and **Copernicus Dataspace** using HTTPS-based APIs and using repositories available on AWS Open Dara Registry. Data is retrieved in GeoTIFF format and ingested into the system's S3-based Data Lake, while metadata (e.g., STAC Items) is extracted and registered in the STAC API for discoverability.
- **Data Distribution via OGC and STAC Services** ([HRLC-TR-23]): Processed and harmonized products are distributed through standard interfaces. The **STAC API** serves metadata and product links in a compliant JSON format, facilitating easy access for users and integration with downstream tools. For long-term archiving and delivery, products are also integrated into the **CCI Portal**.

Code	Source	Destination	Protocol	Format	Description
CCI-HRLC-EXT-IF-000010	Online Repositories (NASA Earthdata, Copernicus SciHub, DIAS)	System Data Lake (S3 Storage)	HTTPS (REST, API)	GeoTIFF, JSON	Integration with external repositories to retrieve satellite data (Sentinel-1, Sentinel-2, Landsat) in GeoTIFF format. Metadata is fetched in JSON for STAC registration.
CCI-HRLC-	System Data Lake (S3 Storage)	STAC API	HTTPS (REST)	JSON (STAC)	STAC API serves as the interface to expose ingested geospatial

EXT-IF-000011					data and metadata for discoverability.
CCI-HRLC-EXT-IF-000012	System Data Lake (S3 Storage)	OGC Services (WMS, WCS)	HTTPS (OGC Standard)	GeoTIFF, PNG, JSON	External data distribution via OGC services, enabling visualization (WMS) and access (WCS) of processed and fused data products.
CCI-HRLC-EXT-IF-000013	STAC API	External Users / Clients	HTTPS (REST)	JSON (STAC)	Provides access to metadata, data catalogs, and links to products stored in the Data Lake for download and analysis.
CCI-HRLC-EXT-IF-000014	System Data Lake (S3 Storage)	CCI Portal (Obs4MIPs Integration)	HTTPS (REST)	NetCDF, JSON	Final harmonized products delivered to the CCI Portal in NetCDF format, following Obs4MIPs requirements for climate research integration.

Table 4-2: External Interfaces of the CCI-HRLC system

4.3 User Interaction and Configuration Management

- Management of Processor Versions, Auxiliary Data, and Parameters
- Accessibility of Data for Algorithm Developers

5 Deployment Scheme

5.1 Deployment Approach of Processing Components

The system ensures automated and consistent deployment of processing components by leveraging **Docker** for containerization, **GitLab** for code management and CI/CD pipelines, and a **Docker Registry** for storing and distributing built images. The workflow for delivering processing components is designed to package algorithms into Docker containers and seamlessly integrate them into the orchestration framework based on RabbitMQ. Below is the detailed workflow:

Algorithm Development and Dockerfile Creation

- Developers implement or update algorithms for processing components (e.g., optical preprocessing, SAR preprocessing, classification).
- Each algorithm is defined within its own repository or directory in **GitLab**, including a **Dockerfile** that specifies the environment, dependencies, and execution commands.
- The Dockerfile ensures the portability and reproducibility of the processing component by encapsulating all required libraries (e.g., geospatial Python libraries like GDAL, Rasterio) and system dependencies.

Build and Test Automation in GitLab CI/CD

- A **GitLab CI/CD pipeline** is configured to automate the building, testing, and deployment of Docker containers.
- When code is pushed to the repository, the CI/CD pipeline is triggered, performing the following steps:
    1. **Build the Docker Image:** The Dockerfile is used to create a container image for the processing component.
    2. **Run Unit and Integration Tests:** The container undergoes automated testing to validate the algorithm's functionality and compatibility.
    3. **Tag and Version the Image:** The built image is tagged with a version number (e.g., v1.0.0) and a commit SHA to ensure traceability.

Deployment to Docker Registry

- Once successfully built and tested, the Docker image is pushed to a **Docker Registry** (e.g., GitLab Container Registry, Docker Hub, or a private registry).
- The registry acts as a centralized repository for container images, enabling easy versioning and retrieval.
- Image tags (e.g., latest, v1.0.0) are managed to ensure clarity and consistency during deployment.

Integration with RabbitMQ-Based Orchestration ([HRLC-TR-12])

- After deployment, the containerized processing component becomes available for execution in the RabbitMQ-based orchestration system.
- **RabbitMQ** serves as the task broker, managing job submissions, load balancing, and message queuing for distributed processing pipelines.
- When a new job is submitted via the **openEO API**, RabbitMQ dispatches the job to the appropriate containerized processor (e.g., optical preprocessing, SAR classification) running on a worker node.
- The **worker nodes** pull the required Docker image from the registry and execute the processing task.

Workflow Example:

1. Job Submission:
   - User submits a job through Jupyter by interfacing the openEO API, specifying parameters and the required algorithm.
2. Task Queuing:
   - RabbitMQ queues the job and sends a message to the corresponding processing worker.
3. Container Execution:
   - The worker retrieves the relevant Docker image from the registry, pulls the data from S3, and runs the algorithm within the containerized environment.
4. Output Storage:
   - Results are stored back into the S3 Data Lake, and metadata updates are made via the STAC API.

Monitoring and Maintenance:

- Logs and job statuses are continuously monitored using the **Status Consumer** that tracks RabbitMQ messages.
- Docker image versions are maintained in the registry, allowing for rollbacks or updates.
- Regular updates to the Dockerfiles and GitLab CI/CD pipelines ensure that the processing components remain efficient, scalable, and up to date.

5.2 Deployment Scheme and Resource Allocation

The motivation for the selection of cloud resources is based on following criteria:

- Data Proximity: e.g. availability of Sentinel 1 and 2 and of Landsat Historical data
- Computational Resources availability On-Demand
- Computational Resources available in the form of Spot Instances

During the first phase, the success of the production was due to the availability of resources and, above all, due to data proximity.

Deployment Scheme:
The following figure describes the deployment scheme of the architecture described in par. 3.1.

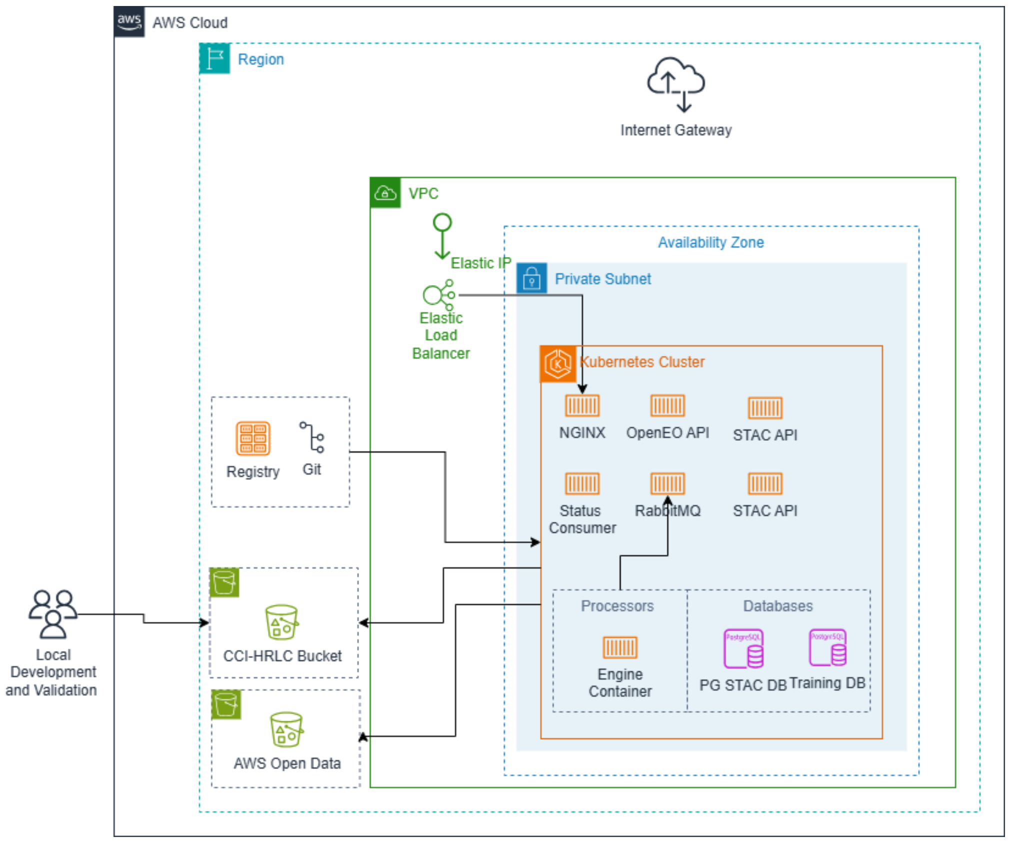


Figure 8: Deployment Scheme of the CCI-HRLC system

The cloud scheme illustrates a **cloud-native architecture** for deploying and orchestrating the High Resolution Land Cover (HRLC) system on **AWS Cloud**. The infrastructure is organized into several key layers and components within the AWS environment, as described below:

- **Region and Internet Gateway:**
  - The architecture operates within an AWS **Region** and utilizes an **Internet Gateway** for external connectivity, allowing access to public endpoints like Git repositories and registries.
- **VPC (Virtual Private Cloud):**
  - A **VPC** is provisioned to create a logically isolated network for the system. It contains all the processing and database components within a private environment.
- **Availability Zone and Private Subnet:**
  - The infrastructure is deployed in an **Availability Zone** within a **Private Subnet** to enhance security by isolating critical components from direct public access.
- **Elastic Load Balancer and Elastic IP:**
  - An **Elastic Load Balancer** distributes incoming traffic across services deployed within the **Kubernetes Cluster**, ensuring high availability and fault tolerance.
  - The **Elastic IP** provides a static address for communication with external components or users.
- **Kubernetes Cluster:**
  - Within the private subnet, the **Kubernetes Cluster** hosts all core components, including APIs (OpenEO, STAC), messaging services (RabbitMQ), status consumers, and processors. This enables container orchestration, resource scaling, and workload distribution in the cloud.
- **Databases:**
  - The **PG STAC DB** and **Training DB** are deployed within the private environment to securely store metadata, training data, and job states. These databases use PostgreSQL with pgSTAC for efficient geospatial metadata management.
- **CCI-HRLC Bucket (S3 Storage):**
  - AWS **S3 buckets** are used as storage for both raw and processed data, ensuring durability and scalability for large datasets. This bucket serves as an interface between local development environments and the cloud processing infrastructure.
- **AWS open data Bucket (S3 Storage):**
  - AWS Open Dataa **buckets** are provided as part of the Open Data strategy of Amazon Web

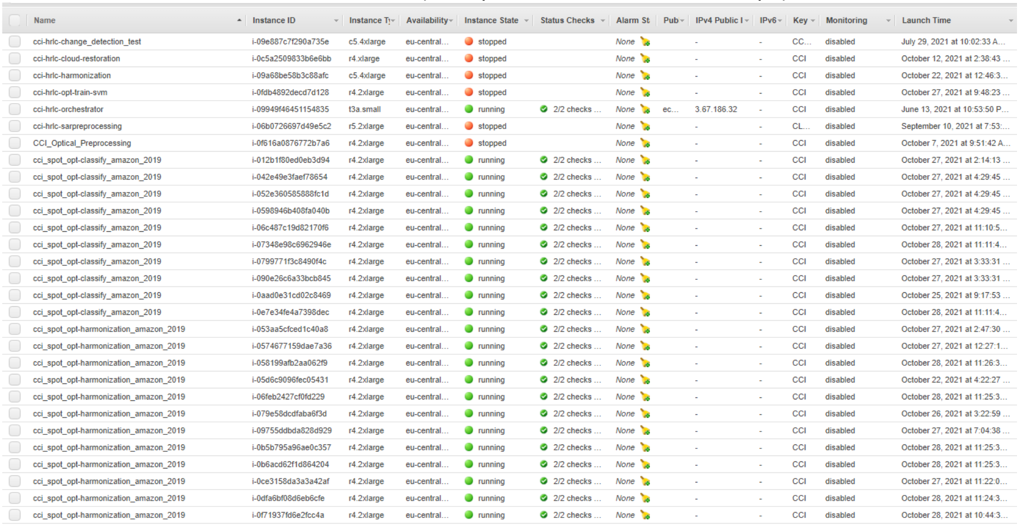
Services, described here: https://registry.opendata.aws/

- **Local Development and Validation:**
  - Users perform **local development and validation** before pushing Docker images or code updates to the **Registry** (e.g., GitLab or Docker Registry) and **CCI-HRLC Bucket**.
- **Registry and Git:**
  - A container **Registry** and Git repository serve as the source for Docker containers and application code. These components are integrated into the Kubernetes Cluster for deployment and execution.

Current Physical resources:

- **Virtual Private Networks (VPC):** 10.3.0.0 Management Network (Orchestrator Component deployed on CLEOS), 10.3.3.0 Processing Network (dynamic resources in an isolated VPC)
- **Orchestrator:** the orchestrator component deployed on CLEOS (see later the general description) is able to spin-up and spin-down EC2 resources based on specific Templates of VMs. According to the availability of specific resources, the orchestrator is able to change the Instance Type according to a predefined list (e.g. if instances of type r5.xlarge are not available in the required number, the orchestrator is able to try with similar instances like r4.xlarge or even with bigger instances like r5.2xlarge).
- **Dynamic resources:** up to AWS available resources. The project required the generation of large areas of Africa (about 1/3 of total area), South America (about 1/3 of total area) and Siberia (1/5 of total area) at 10 m resolution (related to year 2019) and at 30 m resolution (related to years 1990, 1995, 2000, 2005, 2010 and 2015). About 100000 images (from Sentinel 1, Sentinel 2, Envisat, ERS and Landsat) have been processed as source data using extreme parallelism in few days.

Here is the screenshot of the current resources (final phase of classification steps):



Name	Instance ID	Instance Ty	Availability	Instance State	Status Checks	Alarm St	Pub	IPv4 Public I	IPv6	Key	Monitoring	Launch Time
cci-hrlc-change_detection_test	i-09e887c7f290a735e	c5.4xlarge	eu-central...	stopped		None		-	-	CC...	disabled	July 29, 2021 at 10:02:33 A...
cci-hrlc-cloud-restoration	i-0c5a2509833b6e6bb	r4.xlarge	eu-central...	stopped		None		-	-	CCI	disabled	October 12, 2021 at 2:38:43 ...
cci-hrlc-harmonization	i-09a68be58b3c88afc	c5.4xlarge	eu-central...	stopped		None		-	-	CCI	disabled	October 22, 2021 at 12:46:3...
cci-hrlc-opt-train-svm	i-0fdb4892decd7d128	r4.2xlarge	eu-central...	stopped		None		-	-	CCI	disabled	October 27, 2021 at 9:48:23 ...
cci-hrlc-orchestrator	i-09949f46451154835	t3a.small	eu-central...	running	2/2 checks ...	None	ec...	3.67.186.32	-	CCI	disabled	June 13, 2021 at 10:53:50 P...
cci-hrlc-sarpreprocessing	i-06b0726697d49e5c2	r5.2xlarge	eu-central...	stopped		None		-	-	CL...	disabled	September 10, 2021 at 7:53:...
CCI_Optical_Preprocessing	i-0f616a0876772b7a6	r4.2xlarge	eu-central...	stopped		None		-	-	CCI	disabled	October 7, 2021 at 9:51:42 A...
cci_spot_opt-classify_amazon_2019	i-012b1f80ed0eb3d94	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 2:14:13 ...
cci_spot_opt-classify_amazon_2019	i-042e49e3faef78654	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 4:29:45 ...
cci_spot_opt-classify_amazon_2019	i-052e360585888fc1d	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 4:29:45 ...
cci_spot_opt-classify_amazon_2019	i-0598946b408fa040b	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 4:29:45 ...
cci_spot_opt-classify_amazon_2019	i-06c487c19d82170f6	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 11:10:5...
cci_spot_opt-classify_amazon_2019	i-07348e98c6962946e	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 28, 2021 at 11:11:4...
cci_spot_opt-classify_amazon_2019	i-0799771f3c8490f4c	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 3:33:31 ...
cci_spot_opt-classify_amazon_2019	i-090e26c6a33bcb845	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 3:33:31 ...
cci_spot_opt-classify_amazon_2019	i-0aad0e31cd02c8469	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 25, 2021 at 9:17:53 ...
cci_spot_opt-classify_amazon_2019	i-0e7e34fe4a7398dec	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 28, 2021 at 11:11:4...
cci_spot_opt-harmonization_amazon_2019	i-053aa5cfced1c40a8	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 2:47:30 ...
cci_spot_opt-harmonization_amazon_2019	i-0574677159dae7a36	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 12:27:1...
cci_spot_opt-harmonization_amazon_2019	i-058199afb2aa062f9	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 28, 2021 at 11:26:3...
cci_spot_opt-harmonization_amazon_2019	i-05d6c9096fec05431	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 22, 2021 at 4:22:27 ...
cci_spot_opt-harmonization_amazon_2019	i-06feb2427cf0fd229	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 28, 2021 at 11:25:3...
cci_spot_opt-harmonization_amazon_2019	i-079e58dcdfaba6f3d	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 26, 2021 at 3:22:59 ...
cci_spot_opt-harmonization_amazon_2019	i-09755ddbda828d929	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 7:04:38 ...
cci_spot_opt-harmonization_amazon_2019	i-0b5b795a96ae0c357	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 28, 2021 at 11:25:3...
cci_spot_opt-harmonization_amazon_2019	i-0b6acd62f1d864204	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 28, 2021 at 11:25:3...
cci_spot_opt-harmonization_amazon_2019	i-0ce3158da3a3a42af	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 27, 2021 at 11:22:0...
cci_spot_opt-harmonization_amazon_2019	i-0dfa6bf08d6eb6cfe	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 28, 2021 at 11:24:3...
cci_spot_opt-harmonization_amazon_2019	i-0f71937fd6e2fcc4a	r4.2xlarge	eu-central...	running	2/2 checks ...	None		-	-	CCI	disabled	October 28, 2021 at 10:44:3...

Figure 9: CCI HRLC current operational resources

5.3 Long-Term Archiving and Distribution

The final output package will adhere to the CCI Data Standards as described in the applicable document [AD2]. During phase 1, it has to be noted that for large coverage at 10m like in the case of CCI-HRLC, GeoTIFF is the preferred encoding format.

The final datasets will incorporate essential metadata for discoverability and compatibility with geospatial catalogs and tools. Directory structures will follow a hierarchical format, enabling users to navigate the data efficiently. This adherence ensures the seamless integration of outputs into the CCI Portal and other external climate research systems, fostering long-term usability, accessibility, and compliance with international standards for Essential Climate Variable (ECV) data.

Also, phase 2 will focus on providing tools to translate outputs to be suitable for Obs4MIPs (Observations for Model Intercomparison Projects). Obs4MIPs is an initiative designed to facilitate the use of observational data in climate model evaluation and intercomparison projects, such as those conducted under the **Coupled Model Intercomparison Project (CMIP)** framework. It provides a standardized, accessible archive of observational datasets that are formatted and organized in a way that aligns with climate model outputs, making it easier for scientists to compare observations with model simulations. Converting **land cover raster data** into a format compatible with **Obs4MIPs** will involve the adaptation of the data to meet the specific standards and requirements of Obs4MIPs, including its alignment with **CF (Climate and Forecast)** conventions, **netCDF-4 format**, and adherence to metadata and variable naming conventions used in climate modeling projects. Here is a step-by-step hypothesis for the conversion process.