HORIZON 2020
The EU Framework Programme for Research and Innovation

CURE System Design

Deliverable D4.2

DATE
31 December 2021

ISSUE
2.0

GRANT AGREEMENT
no 870337

DISSEMINATION LEVEL
PU

PROJECT WEB-SITE
http://cure-copernicus.eu/

AUTHORS
Mario Dohr (GeoVille)
Samuel Carraro (GeoVille)
Johannes Schmid (GeoVille)
Michal Opletal (GISAT)

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

App	Application
API	Application Programming Interface
CPU	Central Processing Unit
CCSI	Copernicus Core Service Interface
DB	Database
DIAS	Data and Information Access Services
DAG	Directed Acyclic Graphs
RAM	Random-Access memory
Regex	Regular expression
REST	Representational State Transfer
WP	Work Package

1 INTRODUCTION

1.1 Purpose of the document

Deliverable 4.2 is the final delivery of Task 4.2 Preparation of CURE System Design in (WP) 4 Cure System Development.

CURE Prototype System Design will describe the architectural design of the CURE Prototype in detail based on the usage of DIAS as a Service. That comprises the set-up of hardware components like processing units and storage as well as the software design including the interaction between those components. The System Design also includes the evaluation of the most suiting DIAS based on the given requirements from D4.1.

2 SYSTEM DESIGN INFRASTRUCTURE

2.1 CURE SYSTEM Architecture

The CURE SYSTEM is based on a loose microservice architecture. Table 1 lists the main system components (microservices) of the CURE SYSTEM infrastructure.

Table 1: CURE SYSTEM components

Component	Technology
Authentication & Authorization	OAuth2 Server
API Gateway	Python FLASK, Gunicorn and nginx
Interface description language	Swagger
Databases	PostgreSQL
Message queue	RabbitMQ
Job scheduling system	Apache Airflow / Celery (Flower)
Status manager	Python module
Container virtualization	Docker
Logging	Python module
Storage	WEkEO S3-Bucket
Monitoring & Tests	Grafana, API Integration tests

Figure 1 shows the CURE SYSTEM architecture and depicts how the different components are related to each other.

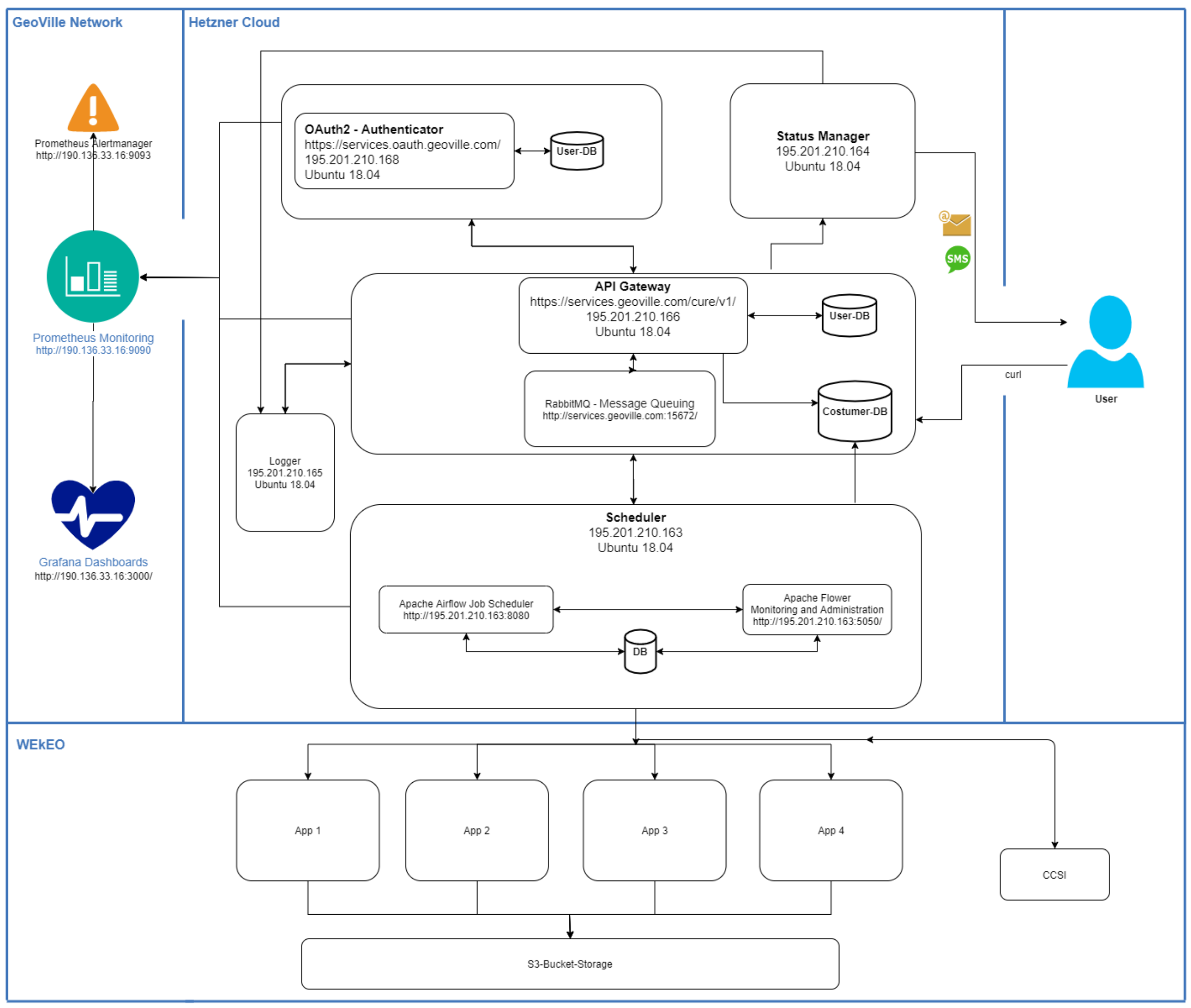


Figure 1: CURE SYSTEM architecture

2.2 CURE SYSTEM Service Submission Workflow

The aim of this section is to get a first rough overview of the CURE SYSTEM infrastructure and the workflow of a service order. In order to use a CURE SYSTEM service a user must perform the following steps:

Registration: Create a user (see Figure 2)

Authentication: Get access to the CURE SYSTEM infrastructure (see Figure 3)

Service Submission: Submit a new service order (see Figure 3)

Service Monitoring: Get the order status of the submitted service

Retrieve Results: Download the result of the service order

A more detailed workflow of a typical service submission is shown in Figure 3 and Figure 4. Figure 2 shows the activity diagram for the authentication process and the submission of a service. Figure 3 highlights the service call workflow from a component-related viewpoint.

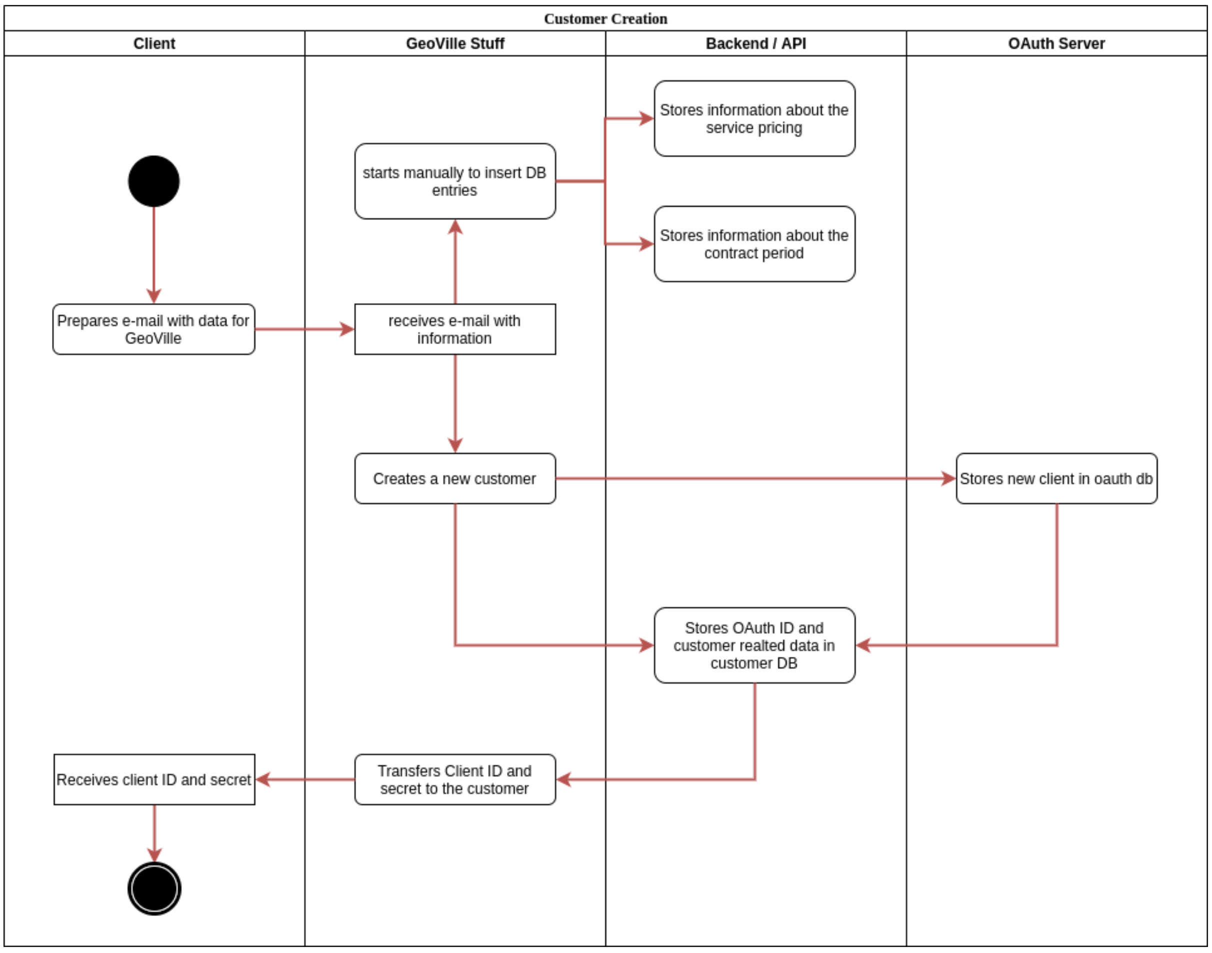

Customer Creation
Client
GeoVille Stuff
Backend / API
OAuth Server
Stores information about the service pricing
starts manually to insert DB entries
Stores information about the contract period
Prepares e-mail with data for GeoVille
receives e-mail with information
Creates a new customer
Stores new client in oauth db
Stores OAuth ID and customer realted data in customer DB
Receives client ID and secret
Transfers Client ID and secret to the customer

Figure 2: Activity diagram for the creation of a customer

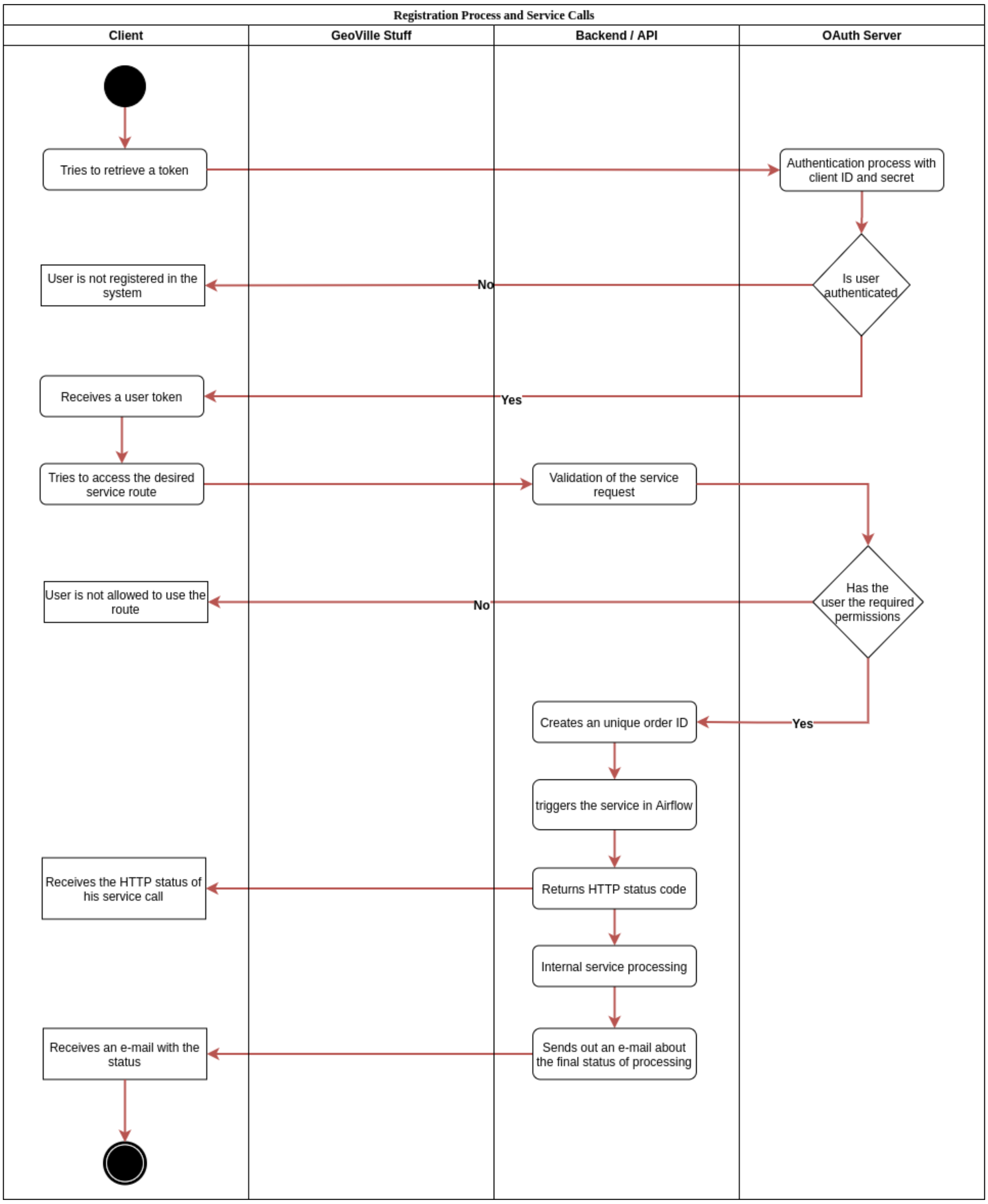
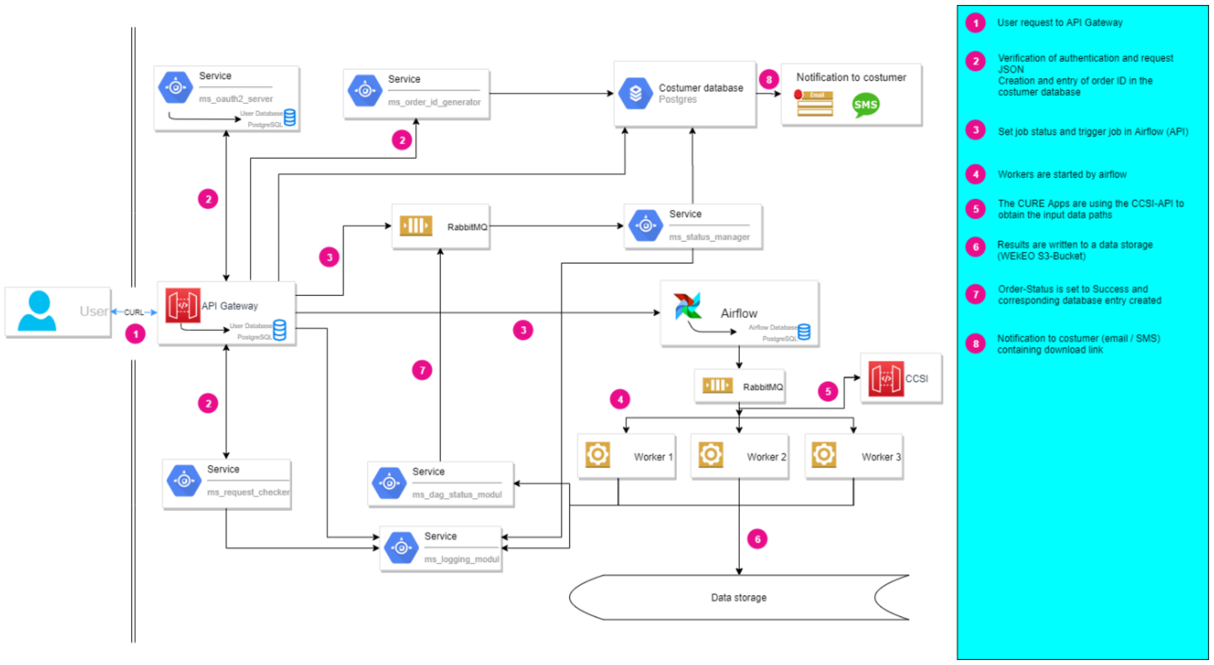

Registration Process and Service Calls
Client
GeoVille Stuff
Backend / API
OAuth Server
Tries to retrieve a token
Authentication process with client ID and secret
Is user authenticated
No
User is not registered in the system
Yes
Receives a user token
Tries to access the desired service route
Validation of the service request
Has the user the required permissions
No
User is not allowed to use the route
Yes
Creates an unique order ID
triggers the service in Airflow
Returns HTTP status code
Receives the HTTP status of his service call
Internal service processing
Sends out an e-mail about the final status of processing
Receives an e-mail with the status

Figure 3: Activity diagram for the authentication process and a service call



1. User request to API Gateway
2. Verification of authentication and request JSON
   Creation and entry of order ID in the costumer database
3. Set job status and trigger job in Airflow (API)
4. Workers are started by airflow
5. The CURE Apps are using the CCSI-API to obtain the input data paths
6. Results are written to a data storage (WEkEO S3-Bucket)
7. Order-Status is set to Success and corresponding database entry created
8. Notification to costumer (email / SMS) containing download link

Figure 4: Component based workflow diagram for a service call

2.2.1 Authentication

As a first step, a user must get access to the CURE SYSTEM infrastructure. Therefore, a user can apply the authentication API endpoint.

- Endpoint for frontend logins: https://services.geoville.com/cure/v1/auth/login
- Endpoint for registration: https://services.geoville.com/cure/v1/auth/get_bearer_token
- Endpoint for access token: https://services.geoville.com/cure/v1/crm/customers/create
- Required information: User information (name etc.), email and password
- Response: Access token, client-id and client-secret

2.2.2 Service Submission

After getting access to the infrastructure, a user can submit a new service order (e.g. *app 1*).

- Endpoint: https://services.geoville.com/cure/v1/app1
- Required information:
  - Access token if the API is directly used (using the LOG-IN Endpoint in an UI will obtain the Access Token automatically)
  - service specific parameters (region of interest, date, etc.)
- Response: Order-id

2.2.3 Service Monitoring

After successfully submitting a new order, a user can monitor the status of the service. Therefore, the CURE SYSTEM API provides the *oder_status* endpoint.

- Endpoint: https://services.geoville.com/cure/v1/services/order_status
- Required information:
  - Access token if the API is directly used (using the LOG-IN Endpoint in an UI will obtain the Access Token automatically)
  - Order-id of the individual service order
- Response: Order status and a link to the final result file if the process is successful

The endpoint supports multiple order states which are listed below:

- FAILED: An unexpected error occurred during the execution of the service
- SUCCESS: Service calculation was successful
- QUEUED: Submitted request is in a waiting position (waiting list)
- RECEIVED: API received the service request and created an order-id
- RUNNING – Service is currently running
- INVALID – Indicates missing satellite data for the requested date or tile

2.2.4 Retrieve Results

After receiving a successful state from the *oder_status* endpoint, a user can access the service result by using the download link provided by the endpoint - typically a HTTP download link. The link to the result or an error notification will be also sent to the user via email.

3 CURE SYSTEM COMPONENTS

This section describes the individual system components in detail and provides useful information to use and extend the CURE SYSTEM - typically a HTTP download link.

3.1 Authentication & Authorization

The CURE SYSTEM user authentication and user management is based on the OAuth2 standard. Therefore, the CURE SYSTEM provides an OAuth server and a database for managing users, clients, access rights and access tokens. The CURE SYSTEM API provides a set of endpoints which are required to perform common authentication and authorization operations. Among others, this includes for example:

- Creating OAuth clients
- Client login
- Access token generation
- Token validation
- Resetting passwords
- Setting scopes

To ensure appropriate authorization standard, the CURE SYSTEM supports user scopes. These scopes are used to grant access to individual CURE SYSTEM services.

3.2 API Gateway

The API Gateway is the entry point to the CURE SYSTEM infrastructure. The RESTful API provides all endpoints which are required to interact with the system. The endpoints are divided into scopes (e.g.: geo-services, custom-relation-management services, etc.)

3.3 Interface Description Language

The CURE SYSTEM API is documented with the Swagger software. Moreover, the Swagger UI supports a straightforward workflow to run, debug and test all CURE SYSTEM API endpoints. Please visit the Swagger UI for more details (https://services.geoville.com/cure/v1/).

3.4 Databases

The CURE SYSTEM infrastructure uses object-relational PostgreSQL databases. Moreover, in order to support geo-related issues, the PostGIS extension is used.

PostGIS is a spatial database extension for the PostgreSQL DBMS. The extension adds support for spatial operations and geometries such as points, lines, polygons, multi-polygons and geometry collections.

3.5 Message Queue

As already mentioned, the CURE SYSTEM infrastructure is based on a loose microservice architecture. The communication between these microservices is implemented using a message broker software. In particular, the CURE SYSTEM architecture uses the open-source message brokers RabbitMQ.

To exchange messages between the microservices, CURE SYSTEM provides a Python module which supports the two main applications of messaging – publishing messages and receiving messages.

- Publisher: Sends messages to a RabbitMQ queue
- Receiver: Reads messages from a RabbitMQ queue

3.6 Job Scheduler

The scheduling system is one of the most important components of the CURE SYSTEM infrastructure. It allows the execution of multiple parallel workflows and tasks. The CURE SYSTEM scheduler is based on the Apache Airflow workflow management platform.

The following sub-sections address some of the most important Apace Airflow components.

3.6.1 DAG

Apache Airflow supports the creation of workflows. A workflow is formulated as Directed Acyclic Graphs (DAG). Usually, such a DAG is a collection of tasks whereas each DAG is represented as a Python script.

In Figure 5 an example DAG is visually presented. Each rectangle represents a task which can be either a Python function, a Bash command or a Docker container. Therefore, before the main Docker container of a specific app runs, previous tasks can take care of obtaining the paths to the input data using GISAT's Open Search API (CCSI), preprocessing, data formatting, etc. In the end, the result can be uploaded to a WEkEO S3-Bucket and potential interim data can be deleted to save space.

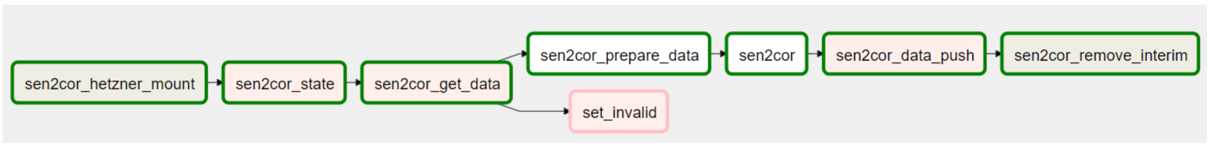


Figure 5: Graph of an example DAG

3.6.2 Operator

A DAG consists of several tasks which are also called operators. Airflow supports various types of operators. Common operators which are used in the current CURE SYSTEM installation are listed below:

Python Operator: Executes Python callable and commands.

Bash operator: Executes commands in a Bash shell.

Docker operator: Executes a command inside a docker container.

3.6.3 Worker

One advantage of Apache Airflow is the support of distributed system architectures. Therefore, so called workers are running on worker nodes. Jobs can be transformed from the main Airflow application to worker instances by using message protocols. To put it simply, each CURE app can run on a separate virtual machine and gets managed by the airflow scheduler machine.

3.6.4 Parallelism

Apache Airflow provides powerful tools and configuration options to run workflows and even DAGs in parallel. This helps to manage huge workloads.

3.6.5 Monitoring

Airflow provides a powerful monitoring tool, which helps to monitor, start, delete and debug workflows and DAGs.

3.7 Status Manager

The status manager is a Python based module which listens to events that affect the status of a service order. It updates the order status in a database accordingly. For specific status updates, additional actions are triggered (e.g.: e-mail notifications).

3.8 Container virtualization

For OS-level virtualization the CURE system uses Docker. As already mentioned, Apache Airflow provides an operator for efficiently running Docker containers. Docker images can be stored on a Docker image hub (registry.cure.geoville.com). Therefore, it is important to use correct tags.

Please consider the following recommendations:

- Small images are desirable
- In general, each piece of code should also be checked regarding the recommendations listed below:
    - Use the provided logging module (Python) or the logging API
    - Make use of only one programming language
    - Follow Coding standards (e.g. PEP8 for Python)
    - Include sufficient error handling
    - Documentation with comments
    - Avoid the creation of temporary data if possible and do not forget the final deletion of it
    - Create tests cases

  - All parameters which should be able to get set by the user need to have a respective command-line argument

3.9 Logging

The CURE system supports two different techniques for logging:

- Python based module which provides a command for logging.
- API endpoint: A REST endpoint which allows logging by applying HTTP requests.

Generally, both logging mechanisms send messages to a queue. A standalone program (*GeoVille_MS_Logging_Saver*) reads the log data from this queue and stores the messages in a database.

In order to support traceability and debugging, the logging module provides different log-levels:

- INFO: Confirmation that things are working as expected.
- WARNING: Indicates that something unexpected happened. The software is still working as expected
- ERROR: Indicates a serious problem. The software has not been able to perform some function.

3.10 Storage

The CURE system must be able to handle large amounts of geo-data. This data should be accessible (read, write) by using standard technologies (e.g.: boto3). Therefore, the CURE system uses an object storage which is easy to use and location independent (WEkEO S3 Buckets).

3.11 Monitoring & Tests

For monitoring and obtaining OS level statistics, the CURE system uses numerous Grafana dashboards. These are helpful to detect defects at an early stage. Moreover, integration tests for various stages are scheduled to continuously check the system health.

3.12 Copernicus core service interface

Copernicus Core Service Interface is a python-based service module responsible for searching input data paths across the DIAS. CCSI provides a unified OpenSearch interface across multiple collections and provides harmonized output. CCSI is intended to provide access to the products generated by the CURE system and stored on WEkEO.

3.13 Cure Portal

Cure portal is a web-based application that allows the user to interact with CURE products. The application will be based on the concept of storyline. Each storyline will represent a cure application or their combination to underline added value of the CURE products. The set of interactive maps, graphs, charts, tables, and other elements will be used to demonstrate the product's values and their benefits for the user. Storylines will be developed around the demonstration products generated for frontrunners cities. Part of the Portal will be user registrations and limited access for registered users to generate their own products by using certain CURE applications.

4 DIAS SELECTION

Smart decisions in a complex and strongly linked world need to be based on high value information extracted from multiple datasets by well suited data preparation steps. To derive all the insights two technical components are essential:

- scalable and reliable IT infrastructure and
- fast accessible, heterogeneous big datasets

For this purpose, in the context of Copernicus the EU has initiated data platforms called DIAS (Data and Information Access Services). These platforms are meant to offer All-in-One Access to satellite imagery and high values Services called Copernicus Services.

After a careful platform selection process, the consortium decided to implement its work horse for processing chains on WEkEO. WEkEO is the EU's Copernicus DIAS reference service for environmental data, virtual environments for data processing and skilled user support.

As highlighted in Figure 6, WEkEO offers a large range of pre-processed Sentinel satellite fleet data, data from Copernicus Services, and additional in-situ data. In total 235 open datasets are available ranging over many thematic and geographic areas.

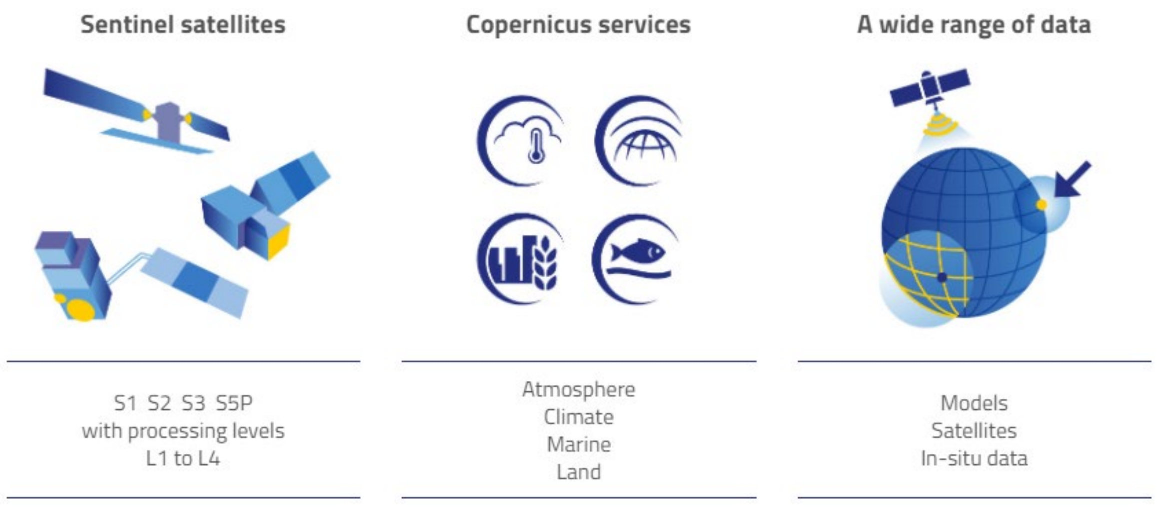


Figure 6: WEkEO Dataset

Besides the data availability, WEkEO offers additional computation infrastructure in terms of typical cloud virtualized engines. They are arranged along various virtual processing environments, suitable to serve the distributed processing system of CURE.

CURE’s backend system is based on fully virtualized containers offering:

- API based system interaction for job execution as well as data access

- central storage access components
- central job execution components
- central monitoring and logging
- user authentication
- high degree of scalability through central container orchestration

Additionally, in many projects undertaken by CURE’s consortium members WEkEO has proven service maturity, excellent stability and fast support. Therefore, WEkEO is an ideal partner for a project such as CURE.

5 Conclusion

This document presents the current version of the CURE System Design and its components and functionalities. This first version of the System will be further developed, improved, and adjusted in accordance with initial specification and planned project schedule and according to specific needs of CURE applications. As a next step the development of the CURE prototype, based on User-Requirements and System design, will be done, following an agile software development approach by iteratively build a system with step-by-step adding of functionalities (components and services). Necessary changes will be will in close iteration with the work conducted in the frame of WP3 and WP2.