D3.2 - EO4EU automated systems and services (a)

Work Package	WP3, Data Orchestration and ML
Lead Author (Org)	Babis Andreou (NKUA)
Contributing Author(s) (Org)	Kakia Panagidi (NKUA), Nektarios Deligianakis (NKUA), Kostas Kiriakos (NKUA), Vironas Anemogianis (NKUA), Konstantinos Mirtolari (NKUA), Michalis Loukeris (NKUA), Ioannis Varouxis (NKUA), Vironas Korpas (NKUA), Armagan Karatosun (ECMWF), Tolga Kaprol (ECMWF), Claudio Pisa (ECMWF), Vasileios Baousis (ECMWF), Lakis Christodoulou(EBOS), Lucia Rodriguez Munoz (CINECA), Francesco Maria Cultrera (CINECA), Piero Scrima (ENG)
Due Date	30.11.2023
Date	29.11.2023
Version	V1.0

Dissemination Level

X	PU: Public
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

Disclaimer

Versioning and contribution history

Version	Date	Author	Notes
0.1	28.07.2023	Babis Andreou(NKUA)	TOC and V0.1
0.2	30.10.2023	All partners	Input
0.3	05.11.2023	Babis (Andreou)	Refinements
0.4	15.11.2023	Kakia Panagidi (NKUA)	Review
0.5	22.11.2023	Babis Andreou, Kakia Panagidi (NKUA)	Refinements
1.0	27.11.2023	Simone Mantovani (MEEO)	Review

Terminology

Terminology/Acronym	Description
CSA	Coordination and Support Action
DoA	Description of Action
EC	European Commission
GA	Grant Agreement to the project
WFE	WorkFlow Editor
PRP	Pre-Processor
PP	Post-Processor
PS	Provision Service
FE	Fusion Engine
OSS	Operations Support System

Table of Contents

List of Figures

List of Tables

Executive Summary

This deliverable reports the progress made in Task T3.2 of the WP 3 during the first implementation cycle of the EO4EU project. Firstly, the document provides an insight of the Initial Development Infrastructure by presenting the cloud infrastructures that EO4EU platform will use. Following, in the document is presented the EO4EU software components and functionalities describing the initial development process. Furthermore, the document provides integration, validation and testing description, in a component level analysis, based in the methodology defined in D4.7. Finally, this deliverable presents a technical overview, usage and foreseen improvements for each component that was developed during this cycle and includes a summary of foreseen components.

1 Introduction

1.1 Scope of D3.2

This deliverable describes the implementation of the EO4EU components involved in the first implementation cycle. The document presents:

- The general technical approach
- EO4EU hardware and software environment
- Technical description and usage of EO4EU components
- Integration and validation in component level
- Foreseen refinements

This document is structured as follows:

- Chapter 2 presents the development status of the PaaS tier
- Chapter 3 contains the integration and validation process concerning each specific component
- Chapter 4 contains the conclusion

1.2 Relation to other deliverables

A detailed description of requirements, design and specification of EO4EU platform was carried out in D2.2 and D2.4. The D2.2 includes the first version of user requirements of EO4EU and the baseline of all business processes, which will take place through a flowchart methodology. Furthermore, the D2.4 presents the system specifications for all different components of the EO4EU solution and a technical overview of the architecture of the proposed solution. In D4.5 there is a fully description and repository reference of the implementation of the serverless FaaS architecture of EO4EU platform.

2 Development of EO4EU Systems and Services

2.1 Initial Development Infrastructure

The EO4EU platform leverages on CINECA ADA Cloud and ECMWF/WEkEO DPI (Distributed Partner Infrastructure) cloud infrastructures, constituting a multi-cloud system. The description of these infrastructures and their interconnection is presented in detail in deliverable D4.3.

The ADA Cloud HPC infrastructure, co-funded by the European ICEI project, is a Tier-1 system designed for scientific research. It offers high performance and flexibility by integrating cloud computing into the HPC ecosystem. This flexibility allows adaptation to diverse user workloads while providing powerful computing capabilities. Additionally, other top-tier HPC systems like GALILEO100 and Leonardo can be integrated into the workflow as computing needs grow. Data can be stored in dedicated areas (DRES) accessible by all HPC systems, enhancing collaborative research.

WEkEO, a part of the Copernicus Data and Information Access Services (DIAS), operates on a robust cloud infrastructure comprising high-performance servers, extensive storage solutions, and high-speed networking. Load balancers optimize network traffic for service stability. WEkEO utilizes OpenStack, offering Infrastructure as a Service (IaaS) functionalities and orchestrates virtual machines, object and block storage systems, and networking capabilities, providing a flexible and scalable environment for Earth observation data processing and geospatial analysis tasks.

2.2 Software Components and Functionalities

In the figure below there is a general overview of the EO4EU platform architecture. Data, IaaS, Paas, ML and Frontend tier are the architectural layers that are presented in the D2.4 “Technical, Operational and Interoperability Specifications and Architecture” deliverable. In this document is presented in detail the backend System and Services of the EO4EU platform PaaS tier that consists of the Platform Controller, Fusion Engine, and Platform Orchestrator.

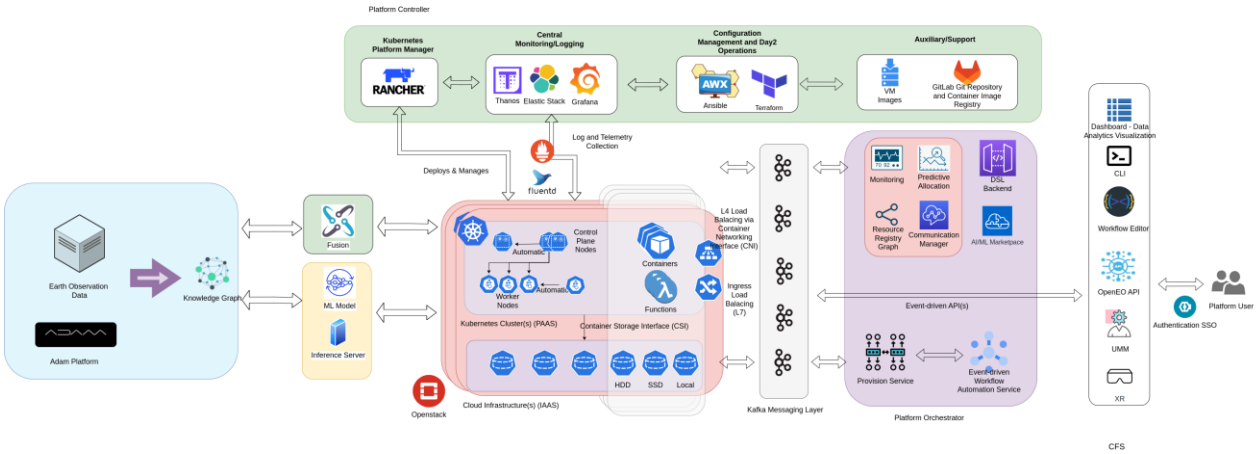

Figure 1: EO4EU architecture (first version) and current integration coverage

2.2.1 Systems and Services

2.2.1.1 Platform CONTROLLER

2.2.1.1.1 Platform Manager

Component	**Platform Manager**
Responsible partner	ECMWF
Participant partners	CINECA
Parent Component	Platform CONTROLLER
Technical description	Platform manager offers a powerful platform for the deployment, administration and orchestration of Kubernetes clusters. It seamlessly integrates with cloud infrastructure through Cluster API, facilitating native compatibility. Platform manager streamlines cluster access by providing both a user-friendly graphical user interface and an API for efficient resource management across multiple clusters.
Background	
Technologies/Frameworks used	• OpenStack • Kubernetes • Rancher • Terraform and Ansible for the deployment
Input	-
Output	• Orchestration of the workloads on Kubernetes clusters across different cloud providers

2.2.1.1.2 Monitoring

Component	**Central Monitoring**
Responsible partner	CINECA
Participant partners	ECMWF, NKUA
Parent Component	Platform CONTROLLER
Technical description	The Monitoring component provides insightful information on the status of the platform by collecting metrics from various targets, such as applications and servers, to ensure, for example, their reliability and performance. Metrics are scraped and persisted using Prometheus [1]and Thanos [2]and visualized with Grafana[3]. They provide a scalable and efficient monitoring solution for cloud-native environments, helping optimize cloud services and infrastructure. SOTA: Prometheus and Thanos are community-driven projects with an open-source license hosted by the Cloud Native Computing Foundation. Prometheus is an open-source toolkit that scrapes and

[1] https://prometheus.io/
[2] https://thanos.io/
[3] https://grafana.com/

	stores metrics in a time-series database and it supports service discovery and alerting based on predefined rules. Thanos extends Prometheus with global scalability and long-term data retention capabilities, enabling a global view of metrics data and high availability. Grafana is a visual-analytics software which provides tools to turn time series data into graphs organized into custom dashboards. It is capable also of managing alerts from Prometheus AlertManager[4]. The whole architecture is divided into an observer and multiple observe clusters: • Observe clusters collect data directly from cluster local services. • Observer cluster gathers metrics from each observee cluster and provides a centralized database for querying metrics via APIs and for visualization. Following the Infrastructure as Code (IaC) and GitOPS principles, we use Terraform, GitLab and Fleet [5] for deploying each component in Kubernetes clusters. In particular, each observee cluster and the observer cluster use kube-prometheus operator to configure both the Prometheus instances and the Thanos Sidecars to store the collected metrics into a central S3 Bucket and make them accessible to the remaining Thanos components, which are deployed in the observer cluster using bitnami/thanos Helm chart[6]. These components can automatically scale horizontally. Finally, all the metrics are rearranged in visualization dashboards in Grafana, which is managed by grafana-operator.
Background	On the one hand, Prometheus and Thanos are community-driven projects with an open-source license and they are hosted by the Cloud Native Computing Foundation. On the other hand, Grafana is an open-source tool extensively used for data visualization. These technologies are currently widely adopted throughout the cloud panorama, and therefore, they are the optimal choices for EO4EU platform. In addition, several products like Rancher support them by default, which makes their use even more advantageous.
Technologies/Frameworks used	• Prometheus (kube-prometheus). • Thanos (bitnami/thanos). • Grafana (grafana-operator). • Terraform and GitLab CI/CD for the deployment.
Input	• Metrics data exposed by application services in Prometheus format.

[4] https://prometheus.io/docs/alerting/latest/alertmanager/
[5] https://fleet.rancher.io/
[6] https://artifacthub.io/packages/helm/bitnami/thanos

Output	• Collected and deduplicated metrics data exposed to Orchestrator applications and visualized as Grafana dashboards.

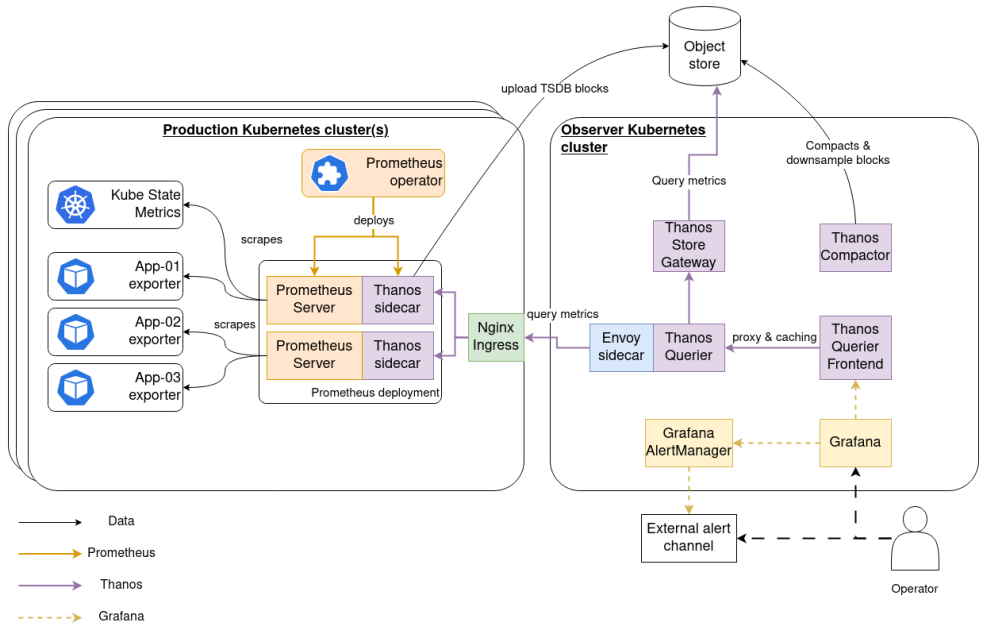

Figure 2: Schematic representation of Monitoring component architecture

2.2.1.1.3 Logging

Component	**Central Logging**
Responsible partner	CINECA
Participant partners	ECMWF
Parent Component	Platform CONTROLLER
Technical description	This component allows us to gather logs directly from the output of every application on the platform and extract meaningful information from them. In the observability platform, this enables navigating from an identified issue to its root cause. The data are aggregated and correlated using different tools: Fluentd [7]and Fluentbit [8]are used to create a unified layer to capture event logs from a diverse source range. Fluent technologies are not used for detailed analytics, but rather to filter these data and extract information from them. OpenSearch is a community driven project, forked by Elasticsearch and Kibana, which is used for searching and analytics. It integrates both a full-text search engine and a dashboard component.

[7] https://www.fluentd.org/
[8] https://fluentbit.io/

	The whole architecture is divided into an observer and multiple observe clusters: • Observe clusters collect data directly from cluster local services. • Observer cluster gathers logs from each observee cluster. Following the Infrastructure as Code (IaC) and GitOPS principles, we use Terraform, GitLab and Fleet for deploying each component in Kubernetes clusters. In particular, each observe cluster and the observer cluster use the logging-operator from Banzai Cloud to configure both the Fluentbit agents (one per node) and the Fluentd instance to centralize the collection of the cluster logs. The collected data are forwarded to an OpenSearch installation (managed by OpenSearch operator[9]) which stores them on block storage and rearranges them in visualization dashboards.
Background	Fluentd and Fluent-bit are community-driven projects with an open-source license, and they are hosted by the Cloud Native Computing Foundation as graduated projects. They are packaged in an open-source Kubernetes operator by Banzai cloud, which is supported by default by Rancher. This allows us to easily operate the logging data retrieval from every cluster of the EO4EU Platform. OpenSearch is a very flexible tool which allows us to collect and visualize logging data. Furthermore, it is fully compatible with the aforementioned tools and its use is attractive due to its open-source license.
Technologies/Frameworks used	• Fluentd and Fluentbit (logging-operator). • OpenSearch (OpenSearch operator). • Terraform and GitLab CI/CD for the deployment.
Input	• Log data from applications.
Output	1 Collected and rearranged log data visualized as OpenSearch dashboards.

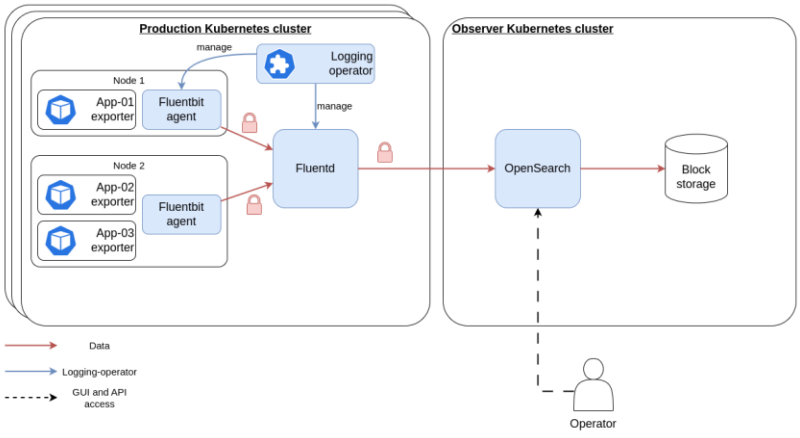

Figure 3: Schematic representation of Logging component architecture.

2.2.1.1.4 Configuration Management and Day2 Operations

[9] https://opensearch.org/docs/2.1/clients/k8s-operator/

Component	Configuration Management and Day2 Operations
Responsible partner	ENG
Participant partners	CINECA
Parent Component	Platform CONTROLLER
Technical description	In the context of EO4EU, Configuration Management and Day2 Operations refers to managing the configurations of cloud resources and applications, ensuring that they are always aligned with project objectives and that environments are consistent and replicable. Including all ongoing management activities that occur after the initial deployment of resources, in the development phase. These activities need to continue even once the system is available in the production environment. In this context it is always necessary to keep the environment updated with all the updates that are part of the normal maintenance and life cycle of the software. First in the EO4EU the system needs to be able to provide the infrastructure by the use of infrastructure configuration, Terraform automatically provisions the necessary resources. Once Terraform[10] has built the infrastructure, Ansible[11] can be used to configure and manage machines, both virtual and physical. Ansible uses playbooks, written in YAML, to describe automation tasks that configure systems to run software, apply patches, manage users, and many other configuration management functions. All this configuration can be stored in a code repository such GitLab[12], that is also available in EO4EU infrastructure. GitLab contains a tool for Continuous Integration and Continuous Delivery (CI/CD) to get the system synchronized to configuration uploaded. These tools help to automate the process of developing, testing, and deploying applications and configurations. The integration of GitLab CI/CD into EO4EU greatly improves the automation of development, testing and deployment processes, ensuring that changes are validated accurately, and that assets and applications are reliably deployed in replicable cloud environments and coherent.
Background	Terraform is an Infrastructure as Code (IaC) tool developed by HashiCorp as shown in Figure 4. It is used to define and provision an IT infrastructure through a high-level language. Terraform allows developers to write code in a declarative format (using HCL - HashiCorp Configuration Language syntax) to describe the infrastructure needed to launch an application or service. It supports multiple cloud providers (such as AWS, Azure, Google Cloud, etc.), as well as on-premise systems. Ansible is an open-source IT automation tool that simplifies configuration management and the automation of deployment processes.

[10] https://www.terraform.io/
[11] https://www.ansible.com/
[12] https://about.gitlab.com/

	GitLab is an open-source project with possible enterprise subscription. GitLab is a complete web platform for managing the software development lifecycle, providing a managed Git repository, along with CI/CD (Continuous Integration/Continuous Deployment), issue management, code review, and many more capabilities tools for collaboration and tracking of software development. GitLab CI/CD is a complete Continuous Integration and Continuous Delivery solution integrated into the GitLab platform. This tool offers the following features: • Pipeline Configuration: GitLab CI/CD allows you to define pipelines that automate your development and deployment processes using a YAML configuration file. This file defines the steps to be performed, including testing, releasing, and monitoring. • Automated Testing: GitLab CI/CD facilitates the automated execution of unit, integration and performance tests, ensuring that configuration and application changes are reliably validated. • Continuous Deployment: With GitLab CI/CD, you can automate the continuous release of changes to your infrastructure and applications, ensuring rapid and controlled deployment. • Integration with Version Control: GitLab CI/CD is integrated directly with version control systems like Git, allowing pipelines to be automatically detected and started in response to changes in the repository.
Technologies/Frameworks used	• Terraform • Ansible • GitLab CI/CD
Input	1 Ansible: YAML config file that defines the activity to run; these files are called playbooks. 2 Terraform requires configuration files in HashiCorp Configuration Language (HCL) format that describe the desired infrastructure, including cloud providers, resources, and dependencies. 3 GitLab CI/CD: GitLab runs a CI/CD pipeline based on the .gitlab-ci.yml configuration file.
Output	1 The results are the infrastructure created for terraform, the machine configuration for ansible and the pipeline execution for GitLab, which may include docker images.

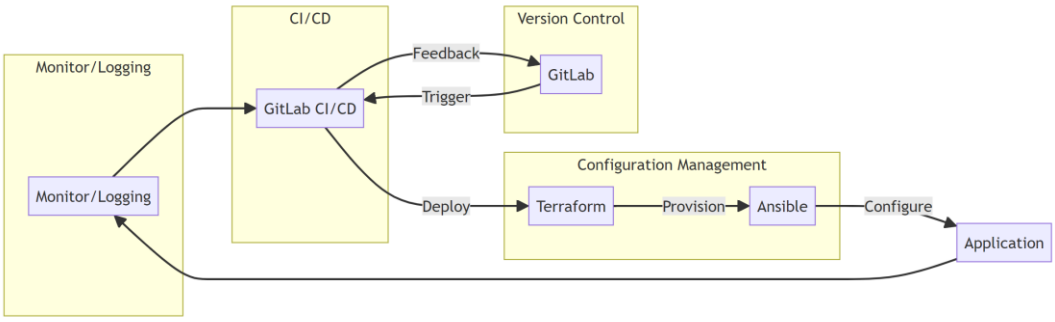

Figure 4: Configuration Management and Day2 Operations.

2.2.1.1.5 Scratch Storage

Component	**Scratch Storage**
Responsible partner	CINECA
Participant partners	ECMWF, NKUA
Parent Component	Platform CONTROLLER
Technical description	Scratch Storage, being a temporary storage area, is optimized for short-term data retention, emphasizing quick access and data purging capabilities. It ensures a clean and efficient storage environment by automatically removing data after a predefined expiration period or upon completion of relevant tasks. It is designed to be a versatile and temporary storage solution for various data needs within our system architecture.
Background	This component will be implemented using MinIO, an open-source, high-performance object storage service that is API-compatible with Amazon S3. MinIO's deployment in this context allows to offer a scalable, secure, and efficient means to store temporary data, such as intermediate file processing results, temporary backups, or data awaiting further processing or transfer.
Technologies/Frameworks used	1 MinIO
Input	• EO and meteorological data
Output	• Post processed data

2.2.1.2 Auxiliary and support

2.2.1.2.1 Resource Registry

Component	**Resource Registry**
Responsible partner	NKUA
Participant partners	ECMWF, CINECA
Parent Component	Platform Orchestrator

Technical description	The resource registry component aims to create a representation of the current instance of the Kubernetes cluster as a graph. Neo4j graph database will be used for this purpose. When a new Kubernetes object is deployed to the cluster, it will be instantiated in the graph database, using the Kubernetes manifest. Having this parallel representation allows us to store information, like resources in use, connections, and dependencies between Kubernetes objects, in an accessible way. Components like monitoring, provision service and predictive allocation will be able to query and update the graph database without burden to the Rancher API. Finally, the graph database will help ensure that a Kubernetes object about to be deployed has all the required resources and the needed Kubernetes objects (dependencies) to run.
Background	For the creation of this component, we considered KubeView, a Kubernetes cluster visualizer and visual explorer. KubeView is deployed in the cluster and provides a mapping of the API objects in real-time from the Kubernetes API. We chose to create our own resource registry component using Neo4j in order to decrease calls to the Kubernetes API to only, when necessary, by using the monitoring component and the Provision Handler. In addition, our approach allows easy querying from the various components and is more versatile in allowing us to visualize more objects in detail and save additional information like the overall health status that is not provided by the Kubernetes API.
Technologies/Frameworks used	• Neo4j, Kubernetes API
Input	• Kubernetes API
Output	• No output. Updating Kubernetes Cluster Graph.

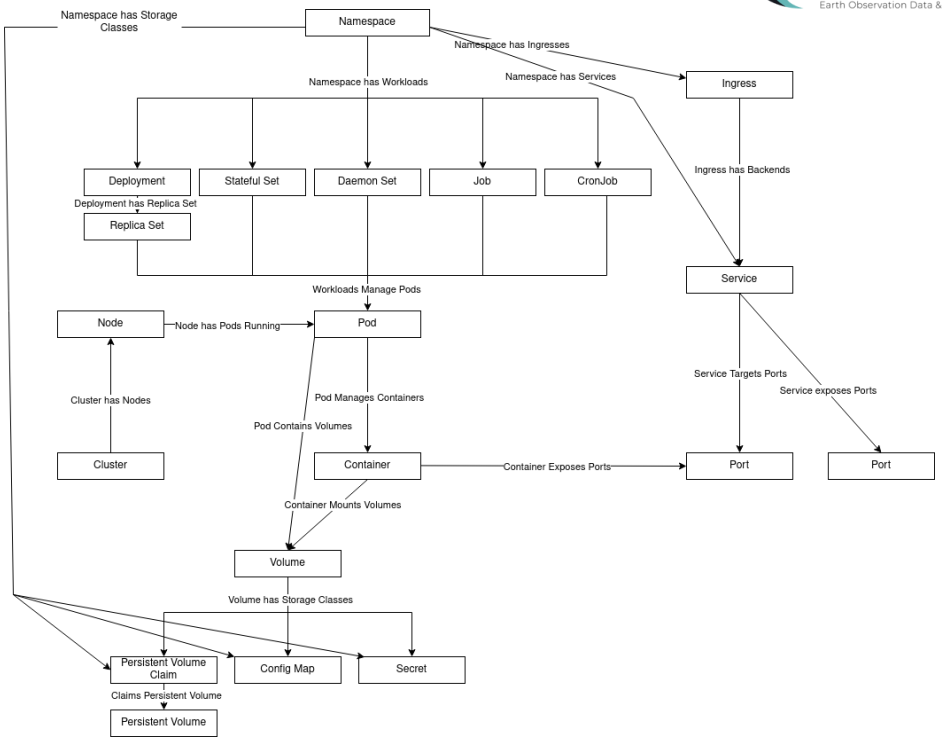

Figure 5: Resource Registry

2.2.1.2.2 Repository Registry

Component	**Container Image Registry**
Responsible partner	ENG
Participant partners	ECMWF
Parent Component	Platform Controller
Technical description	Container Image Registry is a centralized system that allows users to store and manage container images, such as Docker images, within a specific environment. These repositories serve as a secure and organized way to keep track of different versions of container images, ensuring efficient access and version control. In our project, we have chosen to implement this crucial component using GitLab Container Registry for several reasons. GitLab Container Registry seamlessly integrates into an existing Continuous Integration/Continuous Deployment (CI/CD) system, offering a smooth and efficient workflow. This integration means that Docker images created during the CI/CD pipelines can be automatically stored in the GitLab Registry, streamlining the deployment process. The container registry fits perfectly into the operational flows of EO4EU. After the codes of the components to be built have been pushed to GitLab, the latter starts the building which, after having created the image, takes care of uploading it to the registry. Furthermore, once the image has been created and uploaded to the registry, the CI/CD component can trigger other pipelines that deploy it in different environments or that start other processes of creating more complex infrastructures.
Background	GitLab's container registry system is open source, integrated with Docker Registry, and allows to store Docker container images in a private registry managed user or organizations, giving more control

	over privacy and access to images. The main difference between the GitLab container registry and the Docker Hub (which is one of the most well-known public Docker registries) is that the GitLab container registry is a private registry owned and managed by users or your organizations. This means that the container images stored in GitLab are private and accessible only to authorized people. In contrast, the Docker Hub is primarily a public registry where images are generally accessible to anyone, although access control can be set on specific images.
Technologies/Frameworks used	• GitLab Container Image Registry
Input	• Container Images • Helm Charts
Output	• Container Images

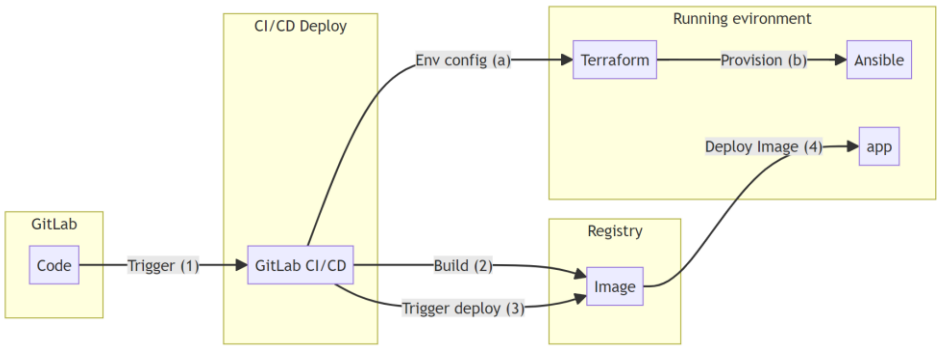

Figure 6: Container Image Registry.

2.2.1.3 Platform Orchestrator

2.2.1.3.1 Communication Manager

Component	**Communication Manager**
Responsible partner	NKUA
Participant partners	ECMWF
Parent Component	Platform Controller
Technical description	The Communication Manager's task is to manipulate the persistent communication flow of EO4EU pipelines and ensuring continuous operations. Efficiently orchestrating and directing messages between services, applications, and subsystems is crucial for maintaining operational coherence for the software procedures. The Communication Manager deftly ensures optimal data delivery by interpreting the source and destination needs while attuned to the fluctuating demands of contemporary systems and can swiftly expand or contract its capacity in response to varying workloads. Such flexibility ensures resource optimization and service efficiency. Furthermore, constructed with a fail-safe design, the module assists

	in the formulation, modification, and termination of communication topics and brokers. The underlying technology that powers and supports the Communication Manager within the Kubernetes ecosystem is the Strimzi operator. Strimzi provides a way to run an Apache Kafka cluster on Kubernetes simply and robustly, making it a perfect choice for our Communication Manager's needs with an interface that exposes the utilization of Kubernetes. Strimzi simplifies the deployment, maintenance, and scaling of Kafka clusters, making it a practical choice for complex communication systems. Also, it uses Kubernetes Custom Resources to control Kafka resources, enabling a declarative approach to configure topics, users, and other Kafka components. This integration allows the Communication Manager to leverage Kafka's robust messaging capabilities for handling high-throughput, distributed messaging systems while benefiting from Kubernetes' scalability and self-healing features. Finally incorporating Cruise Control into the Communication Manager's architecture marks a significant advancement in performance management. Cruise Control's role in continuously monitoring operational metrics and identifying performance bottlenecks is vital. It ensures that the Communication Manager maintains peak operational efficiency, automatically adjusting resources and balancing workloads to meet the dynamic needs of the system.
Background	EO4EU Communication manager will build upon the current standards of the Strimzi operator on Kubernetes. The deployment of Apache Kafka brokers within Kubernetes environments has been significantly streamlined and optimized through the use of Strimzi, an open-source project that provides tooling and operator support for Kafka on Kubernetes. Strimzi leverages the Kubernetes Operator pattern to automate the deployment, management, and scaling of Kafka clusters. It simplifies the process by handling complex Kafka operations, such as configuration, provisioning, maintenance, and upgrades, in a Kubernetes-native way. Strimzi includes custom resource definitions (CRDs) for Kafka clusters, Kafka Connect, Kafka MirrorMaker, and Kafka Bridge, allowing for a declarative approach to configuring these components. The integration of Strimzi with Kubernetes ensures that Kafka brokers are efficiently managed and can dynamically scale to meet workload demands. Strimzi integrates with Kubernetes' self-healing mechanisms, such as pod restarts and auto-replacements, to ensure high availability. It also supports rolling updates for Kafka brokers, which minimizes downtime during upgrades or configuration changes. For monitoring, Strimzi can be configured to expose Kafka metrics which can be collected and visualized using tools like Prometheus and Grafana. This enables real-time monitoring of key metrics such as throughput, latency, and broker health. Additionally, Strimzi's integration with Kubernetes' logging and monitoring infrastructure allows for comprehensive logging and observability, facilitating efficient troubleshooting and performance tuning. The combination of Strimzi

	and Kubernetes offers a powerful and flexible platform for running Apache Kafka in a cloud-native environment, ensuring scalability, reliability, and ease of management.
Technologies/Frameworks used	• Strimzi Operator • Strimzi Cruise Control • Kubernetes Custom Resource Definitions (CRDs)
Input	• Directives for topic and broker lifecycle
Output	• Characteristics and description of the kafka elements

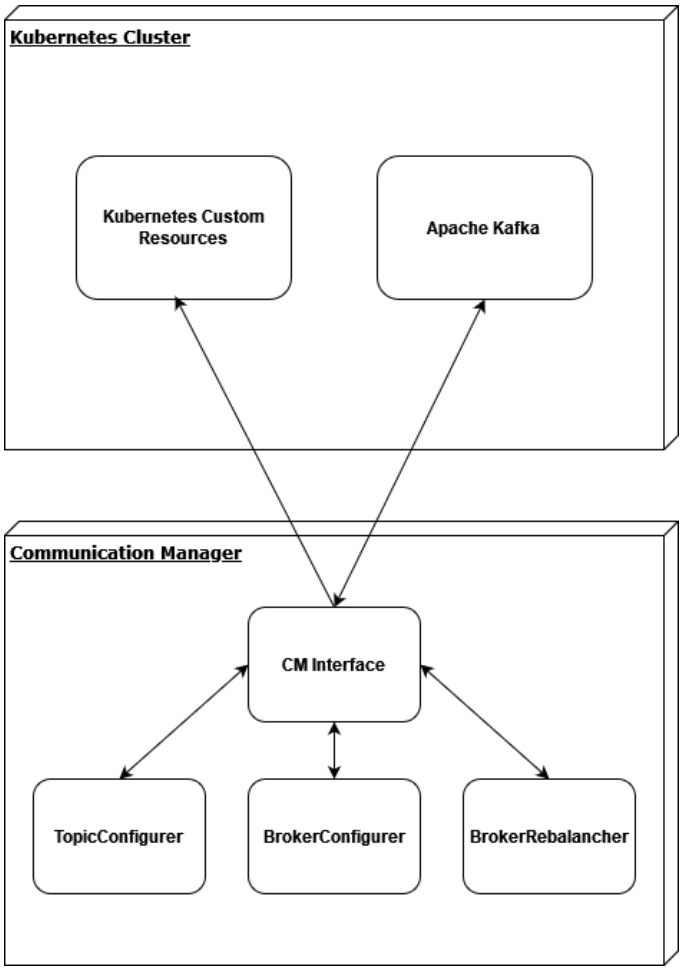

Figure 7: Communication Manager.

2.2.1.3.2 Provision Manager

2.2.1.3.2.1 Pre-Processor

Component	**Pre-Processor**
Responsible partner	NKUA
Participant partners	
Parent Component	Provision Service
Technical description	The Pre-Processor is responsible for downloading the user specified datasets from the corresponding services (CDS, ADS, ADAM etc). The component is fed with the Python script which downloads the specified files (datasets). Pre-process expects the body of the Python script to be encoded in B64.

	After decoding, the script is executed, and dataset files are downloaded. If the downloaded files are packed, the component will unpack them in a temporary directory. Afterwards, the pre-processor will upload all files (downloaded packed file and extracted files) in the corresponding S3 bucket. Upon successful S3 upload, a Kafka message is posted to appropriate topic, to notify the next component in the workflow chain. Finally, the cleanup process runs, which removes all created / downloaded files.
Technologies/Frameworks used	• S3 bucket • Kafka
Input	• Python script to be executed (in Base 64 encoding)
Output	• Upload files to S3 bucket • Kafka message

2.2.1.3.2.2 Post-Processor

Component	**Post-Processor**
Responsible partner	NKUA
Participant partners	
Parent Component	Provision Service
Technical description	The Post-Processor is the final stage of each workflow. To begin, the component listens to a topic for the previous component to finish. Then it iterates all the files (decompressing any compressed tar files) on the S3 bucket uploading any files that are compatible meaning they can be indexed to the elastic search instance for visualization purposes.
Technologies/Frameworks used	• S3 bucket • Kafka • Elastic Search • Pandas/GeoPandas
Input	• Kafka Message
Output	• Elastic Search Upload

2.2.1.3.2.3 Function as a Service (FaaS)

Component	**FaaS Proxy**
Responsible partner	NKUA
Participant partners	ENG,
Parent Component	Provision Service
Technical description	FaaS is a cloud computing service that allows users to run code in response to events without managing the complex infrastructure typically associated with building and launching microservices applications. In this context, the FaaS component receives code snippets from the DSL component and initiates the processing immediately upon receiving a message from Kafka. After the FaaS operation is successfully completed, a notification is sent via Kafka to ensure that all pertinent components are informed of the results.

Technologies/Frameworks used	• S3 bucket • Kafka • OpenFaaS
Input	• DSL output • Kafka message
Output	• CSV, Shapefiles, tiff, tar.xz • Kafka message

2.2.1.3.2.4 Provision Service

Component	**Provision Service**
Responsible partner	NKUA
Participant partners	
Parent Component	Provision Manager
Technical description	The Provision Service is the backbone of the provision manager. The component is responsible for the dynamic instantiation of each user defined workflow. The Provision Service in order to instantiate each user defined workflow, has to receive input from the DSL engine (see section 2.2.3). All the required components and procedures the user has opted for are inputted to the provision manager which are processed by the provision service. This input is parsed and processed in order to extract the required resources and components deployments which will be created on the infrastructure, specifically the dedicated clusters for this project. All the resources to be created are unique, dedicated and oriented to each separate workflow. Kubernetes resources are created to be available for the deployed components. These resources are specific configuration and authentication files storing necessary data and authentication information for storage platforms and tokens for other services. The components require this information in order to authenticate themselves to other services. Also, for all the various components to be coordinated together, the provision service decides and dictates several configuration options for every single component. All the components are assigned input and output communication queues for communicating in a specified order. The components communication sequence is given to the Provision Service from DSL Engine which also is part of the EO4EU workflow's description. Additionally, the provision service provides the needed configuration of the corresponding communications in the workflow for each component. This functionality takes strongly into account any relation and dependency between the various components (the output of one component may be the input for another). After the above steps, the provision service initiates the deployments of all the components (and their corresponding subcomponents, if any). At this stage, the user defined workflow does not yet begin its execution, but all the components have to be deployed and initialized. Finally, the provision service also dispatches messages to the components via predefined communication queues for initialization

	purposes only. Any component is informed of its assigned input and output queues, any selected algorithms or procedures the user opted for in the editor, etc. After the workflow is executed and is bearing results, the DSL engine dispatches a message verifying that the workflow is completed. The provision service begins to free up the allocated resources for the workflow.
Background	Several works can be found studying real time scheduling on distributed core environment (mostly by using CPUs) while other A suggested solution for scheduling any deployment is the Argo CD[13] tool for Kubernetes[14]. However, to avoid additional intermediaries in our workflow, we opted to use the included Kubernetes API (KAPI), which provides all required API calls for scheduling any application on the cluster. Also, Argo CD does not offer out of the box functionalities for creating Kubernetes resources (e.g. secrets, configmaps, etc). It is achievable but requires great effort to accomplish and set up compared to the KAPI which requires only one API call for each such task
Technologies/Frameworks used	• Kubernetes API • Python libraries • YAML • Kafka
Input	• DSL output
Output	• Creation of Workflow dedicated kubernetes namespace • Creation of kubernetes resources (configmaps, volumes, secrets) containing Kafka topic names, S3 bucket name and access details, scripts for downloading EO datasets, configuration details and specifiacations for all deployments. • Deployment of Workflow components (Pre-Processor, Post-Processor, FaaS proxy and OpenFaaS function deployment) and attaching volumes and references to kubernetes resources to be accessed. • Dispatching initialization messages to Fusion Engine and ML component containing specifications for algorithms and operations to be executed on the EO datasets. • Informing the WFE the workflow has been created and is executing.

[13] "The argo cd project," [Last Accessed; 01-November-2012]. Available: https://github.com/argoproj/argo-cd

[14] The kubernetes authors," [Last Accessed; 01-November-2012]. Available: https://kubernetes.io/

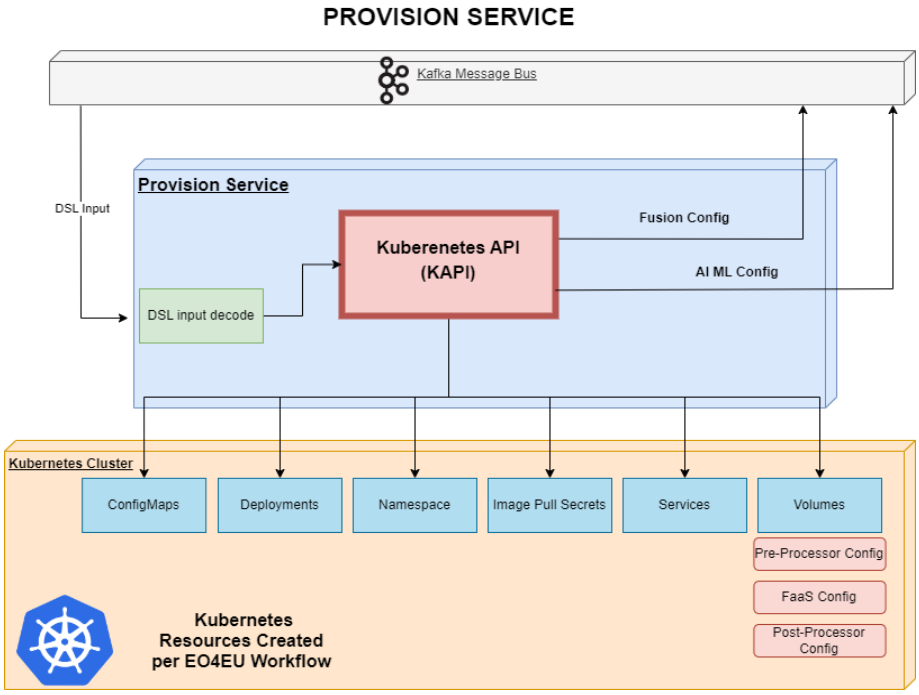

Figure 8: The provision service architecture.

2.2.1.4 Authentication SSO

Component	Authentication SSO
Responsible partner	EBOS
Participant partners	EBOS
Parent Component	Keycloak
Technical description	The SSO (Single Sign On) functionality allows users to sign in once and get access to a set of different applications. Such functionality requires the use of cookies and therefore the access is granted per browser. The Single sign-on (SSO) is an identification and an authentication method that enables users to log into the EO4EU Software platform system that provides access onto multiple applications and services with one set of credentials. SSO streamlines the checking authentication process for users. The Authentication SSO mechanism is being integrated into the EO4EU Software Platform, which exploits a wide spectrum of AI/ML functionalities and EO Services to the end-user in a versatile cloud security framework. The advantage is to allow ubiquitous access to EO4EU services and data offerings. Furthermore, the Authentication SSO mechanism provides an easy and accelerated in time user data engagement with a checking and secure authentication. In particular the user authentication and registration of the EO4EU will be based on ASP.NET Core, containing features for managing authentication, authorization, data protection, HTTPS enforcement, app secrets, XSRF/CSRF prevention, and CORS management. Furthermore, a session mechanism will be applied to assist users interact with the EO4EU framework. Currently, the assessment of the cloud security and authentication mechanism that is being designed, developed and integrated into the platform, is being programmed by the technology specialists in CINECA and ECMWF.

	In addition, from the technical side of development view, a robust security-authentication-encryption mechanism is being integrated into the EO4EU Software Platform System to allow only authorized entities to access system data and functionality subject to specific arrangements and approval previously granted.
Background	The SSO[15], functionality, as a fundamental component of modern authentication and authorization systems, has garnered significant attention and refinement in recent years. Notably, the integration of SSO within the EO4EU Platform hinges on the utilization of cookies, thereby providing access on a per-browser basis. This approach aligns with best practices in contemporary web security. Here, the Keycloak and the User Data Interface are designed-developed-and integrated in the back-end of the User Management Model providing also the software functionality of the Authentication SSO (Single Sign On). Another key-advantage here, is that we are designing the Authentication SSO based on the OAuth 2.0 and OpenID Connect[16], that are fundamental protocols used for authentication and authorization in modern web applications. In particular, OAuth 2.0, is an authorization framework that allows third-party applications to access resources on behalf of a user without exposing the user's credentials. It provides a secure and standardized way for users to grant limited access to their resources to other applications. The use of ASP.NET Core[17], for user authentication and registration underscores a commitment to robust and scalable frameworks. ASP.NET Core encompasses a rich feature set for managing authentication, authorization, data protection, HTTPS enforcement, app secrets, XSRF/CSRF prevention, and CORS management. This ensures a comprehensive and secure environment for user interaction within the EO4EU framework. In this project we also follow the standards of the OpenID Foundation, "OpenID Connect Core 1.0"[18], which is a supporting tool for understanding the technical details and standards of OpenID Connect, including the ID token, authorization request, token endpoint, and other key components of the protocol.

[15] From a technical standpoint, the integration of a robust security-authentication-encryption mechanism is central to the EO4EU Software Platform's commitment to safeguarding sensitive data and functionalities. This mechanism operates on the principle of allowing access solely to authorized entities, contingent upon specific arrangements and pre-granted approvals. This multi-layered approach ensures that only authenticated and approved users gain entry to the system, fortifying the platform's overall security posture.
In conclusion, the integration of the Authentication SSO functionality within the EO4EU Software Platform represents a significant stride towards providing users with streamlined and secure access to a diverse range of applications and services. This development, underpinned by cutting-edge technologies and robust security measures, stands poised to revolutionize the user experience in the realm of Earth observation data analysis, processing, and visualization.

[16] Nat Sakimura, Edmund Jay, and Brian Campbell, "OAuth 2.0 and OpenID Connect (in Plain English!)" ,Nomura Research Institute, Ltd., 2018.

[17] Microsoft Docs, "ASP.NET Core Identity", "Introduction to Identity on ASP.NET Core", Introduction to Identity on ASP.NET Core | Microsoft Learn , Article 12/01/2022.

[18] "OpenID Connect Core 1.0" by OpenID Foundation, 2014.

	Moreover, a session management mechanism is slated for implementation, augmenting user engagement and interaction with the EO4EU platform. This feature will enhance the overall user experience by facilitating seamless interactions with the platform's resources.
Technologies/Frameworks used	• Keycloak • ReactJS • C#, ASP.NET Core
Input	• User Credentials + OTP
Output	• Access Granted to the requested application

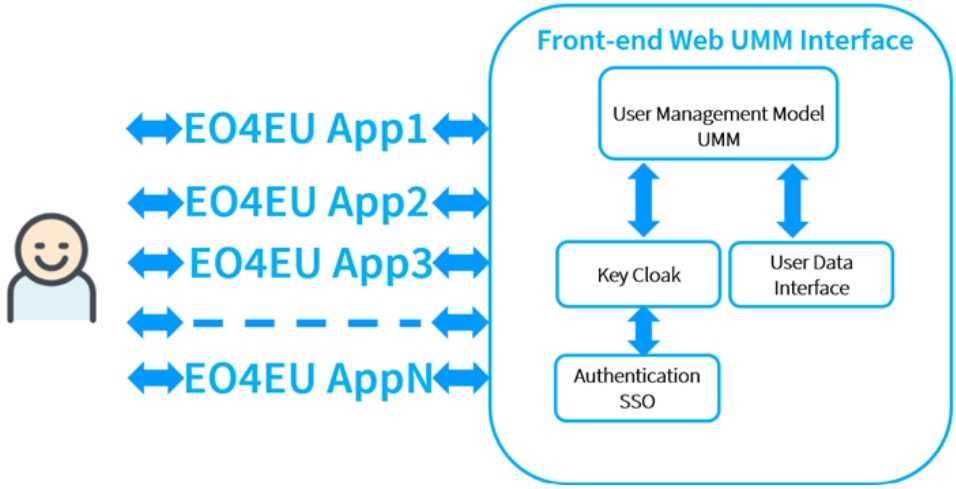

Figure 9: Authentication SSO and End-User Log-in.

2.2.2 Fusion Engine

Component	**Fusion Engine**
Responsible partner	NKUA
Participant partners	MEEO, CMCC
Parent Component	-
Technical description	Fusion (FE) enables context awareness by combining the data readings and leading to situation awareness. FE has two main functionalities: i) create fusion models and pipelines to provide spatiotemporal fusion functionalities of the data, and ii) execute multiple workflows on parallel coming as requests from the user in a dynamic way. Fusion functionalities are "black boxes" to the user, in which are translated to different chains of several algorithms. Each pipeline is tested in Jupyter notebooks and developed in Kubeflow pipelines. Kubeflow builds on Kubernetes as a system for deploying, scaling, and managing complex systems compatible to EO4EU installations. Fusion pipelines have access to data in the S3 bucket of the user as presented in Figure 10 . The configuration of the fusion as long as the access credentials to S3 bucket are consumed by kafka message bus. Fusion pipelines are published in AI/Marketplace in order to be accessible to the users in WFE. Each pipeline is AI/Marketplace has an icon, name, input requests, output requests and a functionality description for user. A Fusion Proxy is also added as the main

	orchestrator to serve multiple requests coming to the FE in parallel from users as shown in Figure 11. When a new workflow is created by WF Editor, a request arrives to Fusion Proxy in order to create a specific workflow in Kubeflow environment, i.e. create a consumer triggering the start of the execution, initialize the namespace, run the pipeline and then publish the results in data repository and inform next component via Kafka. All computations will be investigated to be performed by using HPC or GPU environments and always in regards to high productivity rates.
Background	Spatiotemporal fusion techniques have been gained a great amount of interest during the last decades[19]. The use of EO data combined with multiple sensor sources, mobile or fixed, maximizes the quality of information that arrives to the user as a combination of several interpolated layers. Researchers have shown that maximizing the integrated amount of information through spatial, spectral, and temporal attributes can lead to accurate stable predictions and enhance the final output [20] [21] [22]. Spatiotemporal fusion can be applied within local and global fusion frameworks, where locally it can be performed using weighted functions and local windows around all pixels [23] [24], and globally using optimization approaches[25]. Additionally, spatiotemporal fusion can be performed on various data processing levels depending on the desired tools and applications to be used[26]. It also can depend on the type of data used, for instance, per-pixel operations are well suited for images acquired from the same imaging system (i.e. same sensor) since they undergo a similar calibration process and minimum spectral differences in terms of having the same number of bands and bandwidths ranges in the spectrum, whereas feature or decision level fusion are more flexible and able to handle heterogeneous data such as combing elevation data (e.g. LiDAR) with satellite images[27]. Fusion levels include: i) Pixel-

[19] Bandara, W.G.C., Valanarasu, J.M.J., Patel, V.M., 2022. Hyperspectral pansharpening based on improved deep image prior and residual reconstruction. IEEE Transactions on Geoscience and Remote Sensing 60, 1{16. doi:10.1109/TGRS.2021.3139292.

[20] Zhu, X., Zhan, W., Zhou, J., Chen, X., Liang, Z., Xu, S., Chen, J., 2022. A novel framework to assess all-round performances of spatiotemporal fusion models. Remote Sensing of Environment 274. doi:10.1016/j.rse.2022.113002.

[21] Albanwan H, Qin R. A Novel Spectrum Enhancement Technique for Multi-Temporal, Multi-Spectral Data Using Spatial-Temporal Filtering. ISPRS J Photogramm Remote Sens 2018; 142: 51–63.

[22] Gómez C, White JC, Wulder MA. Optical Remotely Sensed Time Series Data for Land Cover Classification: A review. ISPRS J Photogramm Remote Sens 2016; 116: 55–72.

[23] Feng Gao, Masek J, Schwaller M, et al. On the Blending of the Landsat and MODIS Surface Reflectance: Predicting Daily Landsat Surface Reflectance. IEEE Trans Geosci Remote Sens 2006; 44: 2207–2218.

[24] Wu, X., Hong, D., Chanussot, J., 2022. Convolutional neural networks for multimodal remote sensing data classification. IEEE Transactions on Geoscience and Remote Sensing 60, 1{10. doi:10.1109/TGRS.2021.3124913.

[25] Liu, J., Shen, D., Wu, Z., Xiao, L., Sun, J., Yan, H., 2022a. Patch-aware deep hyperspectral and multispectral image fusion by unfolding subspace-based optimization model. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 15, 1024{1038. doi:10.1109/JSTARS.2022.3140211.

[26] Li, Jiaxin, et al. "Deep learning in multimodal remote sensing data fusion: A comprehensive review." International Journal of Applied Earth Observation and Geoinformation 112 (2022): 102926.

[27] Reiche J, Souzax CM, Hoekman DH, et al. Feature Level Fusion of Multi-Temporal ALOS PALSAR and Landsat Data for Mapping and Monitoring of Tropical Deforestation and Forest Degradation. IEEE J Sel Top Appl Earth Obs Remote Sens 2013; 6: 2159–2173.

	level image fusion [24 25] ii) Feature-level image fusion [28] [29] [30] and Decision-level image fusion [31] [32].
Technologies/Frameworks used	• Kubeflow • Jupyter notebooks • Python libraries • Docker • YAML
Input	• Dataset repository/ S3 bucket
Output	• CSV, Shapefiles, tiff, tar.xz • Kafka message

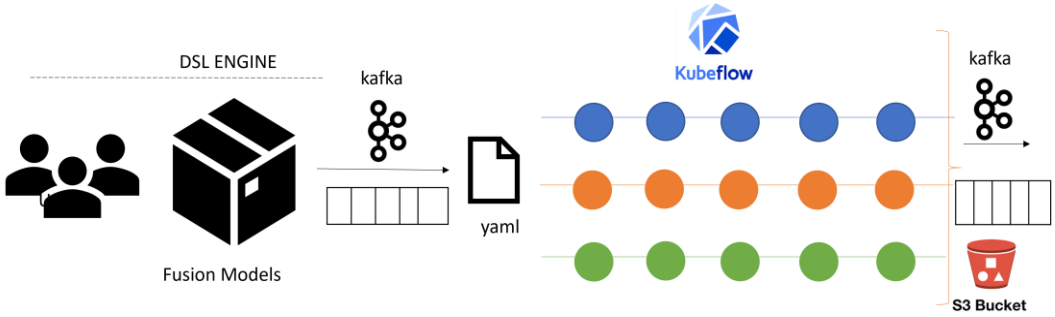

Figure 10: FE lifecycle

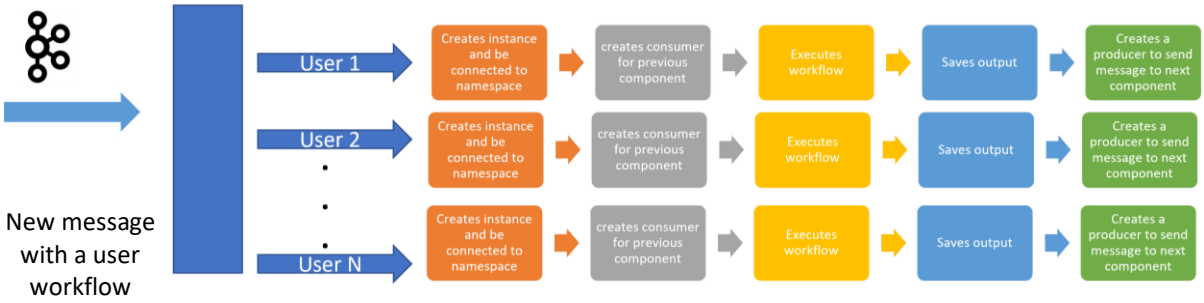

Figure 11: Fusion proxy and parallel processing

2.2.3 DSL Engine

Component	**DSL Engine**
Responsible partner	NKUA
Participant partners	
Parent Component	Workflow Editor

[28] Palsson F, Sveinsson JR, Ulfarsson MO. Multispectral and Hyperspectral Image Fusion Using a 3-D-Convolutional Neural Network. IEEE Geosci Remote Sens Lett 2017; 14: 639–643.

[29] Uezato, T., Hong, D., Yokoya, N., He, W., 2020. Guided deep decoder: Unsupervised image pair fusion, in: European Conference on Computer Vision, Springer. pp. 87{102. doi:10.1007/ 978-3-030-58539-6_6

[30] Wei, W., Nie, J., Li, Y., Zhang, L., Zhang, Y., 2020. Deep recursive network for hyperspectral image super-resolution. IEEE Transactions on Computational Imaging 6, 1233{1244. doi:10. 1109/TCI.2020.3014451.

[31] Hang, R., Li, Z., Ghamisi, P., Hong, D., Xia, G., Liu, Q., 2020. Classification of hyperspectral and lidar data using coupled cnns. IEEE Transactions on Geoscience and Remote Sensing 58, 4939{4950. doi:10.1109/TGRS.2020.2969024.

[32] Albanwan H, Qin R, Lu X, et al. 3D Iterative Spatiotemporal Filtering for Classification of Multitemporal Satellite Data Sets. Photogramm Eng Remote Sens 2020; 86: 23–31.

Technical description	The DSL Engine's role is to act as a validation and control schema for the System Workflows. A DSL (Domain Specific Language) is developed for the needs and characteristics of the Workflow Editor named GDL (Graph Description Language). This language is mainly comprised of structures containing the information of system nodes, their characteristics, metadata and relation other system nodes. The DSL Engine while being a standalone component is tightly integrated with the Workflow Editor (WFE) through the WFE's Auxiliary Service. When a workflow is about to be deployed from the WFE ,it is first sent to the DSL Engine for compilation and validation, where in the case of it being valid will get compiled to YAML format and sent back to the AUX Service for deployments to the Systems ,or in case of it not being valid an error report will be sent to the AUX Service.
Background	
Technologies/Frameworks used	• Xtext • Java Frameworks • YAML • Docker
Input	• Workflow Editor
Output	• Kafka messages

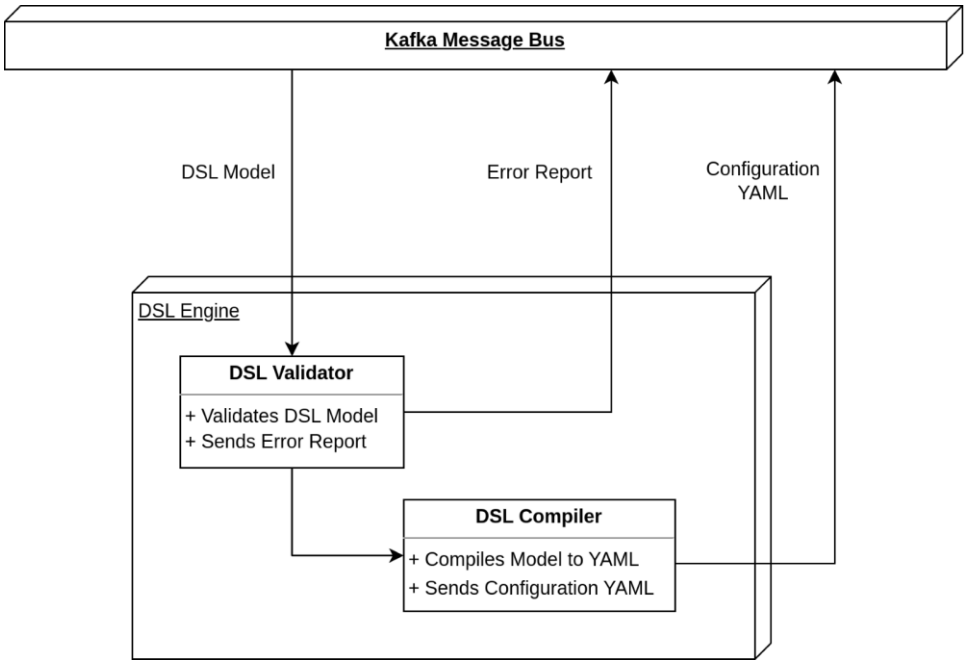

Figure 12: DSL Engine Class Diagram.

2.2.4 AI ML Marketplace

Component	**AI ML Marketplace**
Responsible partner	EBOS
Participant partners	NKUA
Parent Component	Customer Facing Services (Dashboard)

Technical description	The introduced AI/ML Marketplace instance is actually a Software Library-List contained in the Dashboard of the CFS Software package. The AI/ML Marketplace is a software library that operates within the dashboard, offering a collection of machine learning algorithms and models for users to access and utilize on a software platform system. It facilitates seamless integration of various ML and not only capabilities and empowers end-users to leverage these algorithms to solve specific problems. **Technical Features-Technologies:** **Algorithm Repository:** The marketplace contains a repository of pre-trained machine learning algorithms and models that cover a wide range of tasks. More AI/ML algorithms will be added such as the Data Fusion, and other algorithms developed in the project by the assigned partner, will be added. **Model Details and Metrics:** Each algorithm will be defined with detailed information by the corresponding partner-developer, including its purpose, accuracy metrics, input requirements, and output format, allowing users to make informed decisions. **Model Selection:** The end-users can select the ML algorithm that best fits their needs and integrates it into their workflow. **Personalization:** The marketplace allows users to customize their algorithm preferences based on their past selections and feedback, since the dashboard will provide a history events-actions list of the end-user. **Versioning and Updates:** The library maintains version control for algorithms, enabling users to select specific versions based on their requirements. It also notifies users of updates and improvements to the algorithms. **Technologies and Development:** **Backend Framework:** The library's backend is developed using languages like Python, which provide robustness and scalability. **Database:** A database stores algorithm metadata, user preferences, and versioning information. **API Development:** The library exposes APIs to communicate with the ML algorithms and retrieve information about them. **Model Wrapping:** Algorithms initially installed on the HES-SO inference server are wrapped in a standard format, allowing seamless integration into the library. **Interfaces:** User Interface (Dashboard): The AI/ML marketplace is accessed through a user-friendly dashboard, offering an intuitive interface for users to browse, search, and select ML algorithms. **Visual Workflow Editor:** The platform features a visual workflow editor, allowing users to design and execute complex ML workflows by combining multiple algorithms in a drag-and-drop manner. **Technical Features:** Icons and Descriptions: Provide intuitive icons and concise descriptions for each AI/ML model and workflow to help users quickly understand their functionality. Input and Output Specifications: Clearly define the expected input data formats and output data structures for each model or workflow, ensuring compatibility with user data.

	Restrictions and Recommendations: Highlight any limitations or prerequisites for each model or workflow, as well as provide recommendations for optimal usage. **Re-usability to the end-users of the pre-defined WF Models.** By incorporating these trends, security measures, and user-friendly features, the AI/ML Marketplace will empower users to make informed decisions and effectively leverage the full potential of AI/ML models in processing and visualizing EO data and metadata. Each prototyped AI-ML Algorithm-Technique-Model has to be well defined and to keep certain standards in order to be able to accept the input data that the end-user will select and to process the data for further AI Data Analytics or Data Graphs/Visualizations. Data pre-processing will be needed earlier the Data Fusion and ML pipeline processing phase, but a data pre-processing mechanism maybe also needed before it enters the AI/ML Marketplace. The AI/ML section will allow users to get (or download) workflow files published by other users in the S3 buckets. It will also give metadata about different workflow files. Overall, the AI/ML Marketplace software library streamlines the process of accessing and utilizing ML algorithms on the platform system, providing end-users with a specific AI/ML models' array of options for solving data-driven challenges. Its seamless integration with the visual workflow editor and data storage services enhances the platform's capabilities and facilitates efficient data processing and analysis through the Dashboard.
Background	The AI/ML Marketplace concept builds upon the latest advancements in AI/ML integration, data access, and user interface design. Some notable trends and technologies in this area are included in the following key-innovations that are developed in this project. Recent research advancements in Microservice-based Architecture and API services[33] [34] [35], are highly demanded as they are leveraging microservices for building modular, scalable, and secure APIs for exclusive and dedicated work tasks. This allows for easier management and deployment of individual services, enhancing flexibility and scalability. L. Lechner has proposed Integrating Machine Learning Models into Existing Applications[36]. In this area, we are heavily designing and developing prototyped AI/ML models and pre-defined WF models based on a seamless integration and communication of a PostgreSQL database server and the work flow editor. Introducing the availability of AI/ML models inside the front-end web interface of the dashboard, provides a significant availability and capacity of AI/ML utilities to the end user to apply innovative workflows and application. J. Hill et al. has introduced methods of securing APIs[37], thus our aim here was to implement specific authentication and security measurements,

[33]] R. Ranjan et al., "Microservices Architectural Patterns and API Gateway for Containerized Cognitive Computing Systems," in IEEE Transactions on Services Computing, 2018.
[34]] D. Faria et al., "A Secure Microservice Architecture for Healthcare," in Procedia Computer Science, 2018.
[35]] M. M. Rahman et al., "A Review of Microservices Architecture: Key Challenges and Solutions," in IEEE Access, 2020.
[36] L. Lechner, "Integrating Machine Learning Models into Existing Applications," in AI & Society, 2018.
[37]] J. Hill et al., "Securing APIs," in O'Reilly Media, Inc., 2016

	such as: the focus on security, especially in communication with databases, is crucial. Technologies like OAuth 2.0, JWT (JSON Web Tokens), and HTTPS are commonly used to secure API endpoints and data communication. Additional key-technological innovations in this project related with the integration of the AI/ML Marketplace, is the incorporation and data interface with the following software components: I. Workflow Editors: Providing a user-friendly interface for creating, editing, and managing workflows is an essential feature. This empowers users to design their own data processing pipelines. II. Data Visualization Integration: Integration with data visualization libraries and tools ensures that the processed EO data and metadata can be presented in a user-friendly and insightful manner. III. User Management and Access Control: Implementing robust user authentication, authorization, and access control mechanisms is crucial for ensuring that users only have access to the resources and functionality they are authorized to use.
Technologies/Frameworks used	The following Technologies/Frameworks have been used: • ReactJS • C#, ASP.NET Core • Kafka
Input	• Yaml representations of workflows, AI/ML algorithms and their configuration.
Output	• Yaml representations of workflows, AI/ML algorithms and their configuration towards the workflow editor.

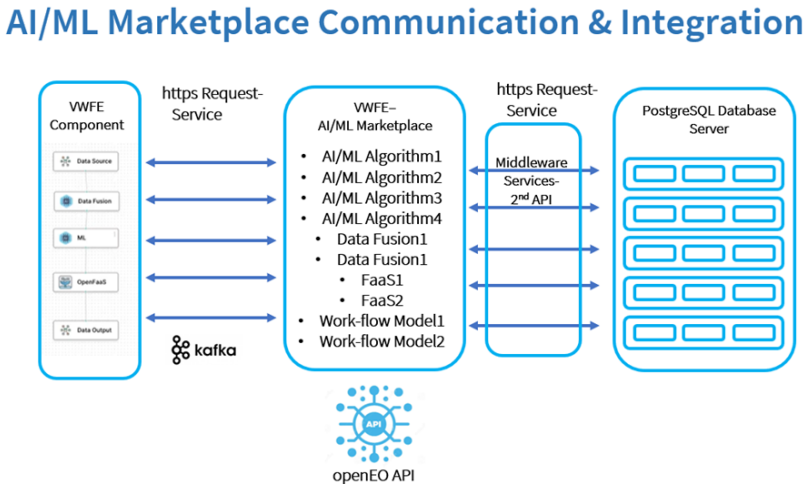

Figure 13: AI/ML Marketplace Communication and Integration – System Architecture

2.2.5 *Infrastructure as a Code*

Component	**Infrastructure as a Code**
Responsible partner	ECMWF
Participant partners	CINECA, ENG
Parent Component	Platform CONTROLLER
Technical description	Infrastructure as a Code is a tool developed by HashiCorp, used to define and provision IT infrastructure through a high-level language.

	Terraform, a part of this tool, allows developers to write code in a declarative format using HashiCorp Configuration Language (HCL) syntax. This format is utilized to describe the infrastructure needed to launch applications or services. Ansible, an open-source IT automation tool, is used for simplifying configuration management and automating deployment processes. The EO4EU project also utilizes GitLab, an open-source platform for managing the software development lifecycle. This includes a managed Git repository, CI/CD (Continuous Integration/Continuous Deployment), issue management, code review for collaboration and software development tracking.
Background	
Technologies/Frameworks used	• Terraform • Ansible • Gitlab
Input	• Terraform Templates • HCL Configuration Files for Terraform • Ansible Playbooks • Git Repositories
Output	• Provisioned Infrastructure • Deployment Artifacts • Infrastructure State Files

3 Integration and Testing of EO4EU Systems and Services

3.1 Approach

The objective of this activity is to produce an end-to-end operational prototype of the EO4EU platform that is used in testing pilots in the context of this specific task and, ultimately in test cases as described in D5.1. The integration process started at the very beginning of the project inception and in its associated description of work, in which numerous design choices have guided the initial steps of the project execution.

To ensure that the developed software operates as expected and is of utmost quality, performing tests is of essence. The purpose of testing is to isolate and identify defects before the software is available on the market and, as a result, improve its quality and ensure high performance. Testing can be done in two ways: automatically or manually. Due to the alignment of the EO4EU Systems and Services integration plan with CI/CD and development and operations (DevOps) methodologies, introducing automation in testing is vital to avoid bottlenecks and ensure integration targets are met without major issues. Apart from the time saved when conducting tests in an automated manner, automated tests are particularly useful to ensure that the software updates do not retroactively introduce problems in the existing code (regression testing), therefore ensuring a much faster delivery cycle compared to manual testing. Hence, EO4EU will align its testing lifecycle approach to have as many automated tests as possible, and consequently to minimize the human factor and reduce the overall test effort and time consumed. The current chosen strategy for testing within EO4EU systems is based

on manual tests, specifically by running user-acceptance tests which involve human operators. We will be including automation testing as soon as possible.

3.2 Methodology

Integration testing includes activities where individual software modules are combined and tested as a group. It precedes validation testing and generally applies tests defined in an integration test plan to aggregates or groups of unit-tested modules with the aim to deliver as its output an integrated system ready for validation testing.

Integration activities follow the individual / unit testing activities performed (mainly in the context of WP4) on the various components defined in the architecture deliverables (WP2 D2.4), and are based on the integration testing plan (verification scenarios) defined in D4.7. They aim to provide sufficient proof of correctness of functionality for combinations of platform components and identify possible bugs and inefficiencies in the foreseen workflow of EO4EU platform services usage. The methodology adopted in EO4EU for integration testing generally follows a bottom-up approach, in the sense that integration activities are performed initially pairwise with test cases involving 2 components that directly communicate either synchronously or asynchronously (via message bus) and then proceeding with more extensive test scenarios involving interactions of multiple components that implement part or complete EO4EU workflows.

The integration tests involved the following major categories:

1. **Testing of components interfaces (black box testing):** This kind of black box testing should be performed for all components implemented in the 1st iteration cycle that provide an interface or are capable to send/receive data from Message Bus.
   An interaction matrix has been created (see Table 1) which provides a quick reference of all the interacting components (including the type of interaction) independent of the tier they exist. Based on this matrix a detailed report was compiled (see section 3.4) which elaborates on the exact interface or message exchange that was tested during integration activities.
2. **Execution/Testing of verification scenarios (1st level of white box testing):** This step involved the execution of all the applicable (since some components were not considered for the 1st iteration) integration and validation tests defined mainly in D4.7 section 4. Although these verification scenarios aim mainly to verify individual components' functionality in most cases, they have as pre-requisite the existence of other components (tools or services). Therefore, despite the individual component testing performed during implementation activities in WP4, the (re)execution of all these verification scenarios was deemed necessary.
3. **Execution of end-to-end scenarios (1st level of system testing):** This step involved the execution of scenarios that address multiple components and verify the behaviour of the system for its expected 'real' usage (i.e. the Provision of workflow and consequent execution and completion of a scenario). No such tests were prescribed/foreseen for integration testing activities during the first iteration cycle. As a consequence, this step will be done in the next cycles. It is however mentioned at this point because it is an important part of the methodology, which should not be overlooked.

Table 1 – Interface Interaction matrix of components.

Components	Platform Manager	Monitoring	Logging	Configuration Management and Day2 Operations	Scratch Storage	Resource Registry	Container Image Registry		Provision Service	Pre-Processor	Post-Processor	FaaS	Authentication SSO	Fusion Engine	DSL Engine	AI ML Marketplace
Platform Manager	x	x														
Monitoring	x	x				x							x			x
Logging	x	x				x							x			x
Configuration Management and Day2 Operations							x					x	x			x
Scratch Storage									x							
Resource Registry	x	x				x			x							
Container Image Registry				x								x				
Communication Manager	x	x							x				x			x
Provision Service		x				x	x			x	x			x	x	
Pre-Processor																
Post-Processor	x	x	x							x		x				
FaaS									x							
Authentication SSO	x	x											x			x
Fusion Engine		x	x		x	x		x	x	x	x	x				x
DSL Engine																
AI ML Marketplace	x	x														

Note: Performance tests and tests involving non-functional aspects of the EO4EU system were not considered as part of the integration activities and will not be included in the present report.

Note: Because some components were not present in the first iteration, to complete the integration testing activities mentioned above, certain assumptions/simplifications were made to meet the prerequisites needed in each test scenario. These assumptions mainly have to do with:

The pre-existence of certain data in the EO4EU database due to the fact that the tool/service that was responsible for inserting/updating these data was not implemented or partly implemented

The fact that a limited number of components involved in the core test workflow were not considered for implementation in the 1st iteration cycle. Thus, these components (involving mainly interactions via message bus) had either to be skipped during integration testing or considered to provide a default functionality

More precise information on the assumptions/simplifications made will be provided on a per test case basis in sections 3.4 and 3.5 that provide details on the testing activities.

Acceptance tests are carried out by end-users to check if the developed system meets the goals as defined in the business requirements. The adoption of the software by target stakeholders is determined by the level of success of acceptance tests. During the development phase, EO4EU partners involved in the development of a particular component will act as end-users themselves and run use case trials.

3.2.1 *Test framework*

Integration of components is performed in stages:

1. *Intra-tier*: addressing activities needed to integrate and test components in the same tier (e.g. platform controller, platform orchestrator, etc);
2. *Inter-tier*: addressing activities needed to integrate and test components belonging to 2 different tiers;
3. *System wide*: addressing activities needed for verifying end to end interaction flows (all tiers, end-to-end integration). testbed

Inter-tier and Intra-tier stages involved both interface testing and functional (white box) testing while the System wide stage focused only on functional aspects.

In order to allow for a common and concise way of representing the results of all kinds of integration tests, two templates were used, that are shown in Table 2 and Table 3:

Table 2 - Template for reporting interface test results.

Component: <***Component Name***>		Conducted by: <**Partner ID**>		Date: **Feb 2016**	Test Category: **Interface testing**
Preconditions		Describe any general precondition that must be present (if any)			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	<Component Name>	R	<Method Name>	Not applicable	E.g. component does not yet exists
2	<Component Name>	M-c	<Message Name>	Partial success	Message was consumed by Resource controller since Experiment Controller does not yet exists Message successfully received by receiving component
		M-c	<Message Name>	Not tested	E.g. functionality not yet supported
3	Message Bus	M-p	<Message Name>	Success	E.g. connection to database succeeded Retrieval/update/insert of information succeeded
4	<Component Name>	JDBC	<Method Name>	Fail	Describe reason of failure e.g. connection to database fail

Regarding the above template:

- For message-oriented communications (where the message bus acts as intermediate) since we have producers and consumers, in the interface template we depict both of them using the convention M-c, M-p so that it is clear that the producing component sends to MessageBus and the consuming component receives the message
- For other types of synchronous interactions like REST, SOAP/ RPC, JDBC, S3, configMap etc. it is obvious that the interface template will refer to component that initiates the communication (caller).
- Allowed status include *Success / Partial success / Fail / Not tested / Not applicable*
- *Success* status is highlighted in green color, *Partial Success* in orange, while *Not tested / Not* applicable are identified in grey

Generally, we include information regarding interactions with the message bus by both producers and consumers components. Interface of type M-p (that is the case the component acts as producer) should not include any related component (or only “Message Bus”). This message may be received by multiple consumers and this interaction is shown in the interface table of each receiver component including information for the exact producer. Therefore, there is no need to replicate this for the producer by including several similar rows.

The rationale of not specifying a related component when type of communication is M-p is that this kind of communication is quite loosely coupled and in general it is not easy for the producing component to know which target component will consume the message. There can be one or many components but there is no reason i.e. to create 10 rows in the producer component because the message will be consumed by 10 components.

This information is shown to the related component that acts as consumer (has type M-c).

In the case of interface testing that refers to communication between components, there are no steps here, but only *Success, Partial success, Fail* or *Not tested* with a possible remark.

Table 3 - template for reporting integration scenarios test results

Test ID**: MB02**	Conducted by**:** **<Partner ID>**	Date**: Feb 2016**	Test Category**: Verification Tests (XX tier)**
Hardware Configuration	See section 2.3.1		
Software Configuration	See section 2.3.1		
Test Name:	***Receive resource provision notification***		
Preconditions	• The user must have a registered email account belonging to the platform • A long-term selection must be ….		
Related Requirements	(may not be present in integration tests)		
Tools Used	*list any special or extra tools used beside code tests*		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Book any resource in order to carry on a certain experiment in the near future	Reservation data entries are added to the DB	Success / Partial success / Failed / Not tested / Not applicable	*list here any divergence from initial foreseen action*
2	Wait till the established date and time to be executed	-		
3	Verify that user has received the corresponding notification regarding the ...	An email is send to the user		
4				

Regarding the above template:

- HW and SW configuration may refer to EO4EU Platform and/or infrastructure. For the platform case a common configuration was used in all integration activities which is listed in section 3.3.2. Information for the infrastructure can be found in section 3.3.1.
- The field related to requirements may be omitted in this first iteration report. The rationale is that integration tests generally are component level specific activities. However, during the integration period (May-November 2023) the only available requirements were the ones of D2.4 which were mainly high-level system requirements that aim to outline the overall behaviour, services and performance characteristics that the EO4EU platform architecture should adhere to.

- Although the *action* field usually refers to a step that must be user initiated in certain cases (to better illustrate the flow of activities) it is possible to include their activities that are performed by a component (once or on a periodic basis) as a result of previous resultField *expected result* might include a single or multiple outcome(s). In the latter case the outcomes should be numbered accordingly in order to easily distinguish them.
- In the verification test, we use the nearly the same status labels *Success / Partial success / Failed / Not tested / Not Applicable* (keeping in mind that partial success can apply only in situation where a single step entails multiple results).

This addresses the verification of the component and system beyond the syntactical and static analysis of the correct combination and matching of inter-component interfaces, initial requirements and pre-conditions.

3.3 Integration Environment Setup

Detailed descriptions of the infrastructure and the platform are provided in D4.3. In the following subsections we summarize the main information.

3.3.1 Infrastructure

For the multi-cluster orchestration at CINECA, resources are allocated for computationally intensive workloads related to EO4EU services. The initial allocation includes 200 vCPU, 1.5 TB of RAM, 5120 GB of block storage, and 40 public IPs.

In the case of WekEO's infrastructure, resources are primarily allocated for computationally intensive tasks related to EO4EU services. The initial allocation includes 100 virtual machines with 384 vCPUs, 768 GB of memory, 300 volumes, and 64 TB of volume storage.

Any need for additional resources will be evaluated after the integrated platform undergoes its initial operational testing.

3.3.2 Platform

The EO4EU multi-cloud platform implementation employs multiple Kubernetes deployments to leverage different cloud benefits, minimize vendor lock-in risks, and enhance resilience. Using cloud-native tools like Rancher, Kubernetes clusters are deployed with uniform configuration management, ensuring consistent application delivery through containerization, CI/CD, and automated deployment processes. Monitoring, logging, and alerting are unified for visibility, and security and compliance are managed through unified Identity and Access Management (IAM) and policy enforcement. Gitlab serves as the Version Control System, supporting source code management, software development collaboration, and CI/CD pipelines, enhancing development workflows within the platform.

Rancher, chosen for deployment, enables the installation and configuration of Kubernetes clusters, streamlining multi-cluster organization. This approach ensures scalability, high availability, and optimal resource allocation, essential for integrating various tools and services required for EO4EU data processing and analysis within the platform ecosystem.

3.4 Integration Test Results

Table 5 - Interface types used in interface testing

Type	Description
M-c	Message bus consumer (receives messages from the message bus)
M-p	Message bus producer (sends messages to the message bus)
REST or R	REST (via HTTP) web service
gRPC	gRPC protocol
VPC	S3 protocol interface
configMap	Configuration files mounted directly to the components
EO sources or EOsrc	Interface with the EO data sources to retrieve data
KAPI	Kubernetes cluster API through kubernetes framework for python

3.4.1 Platform Integration

3.4.1.1 Provision Manager

Table 4: Provision Manager interface test results

Component***: Provision Manager***	Conducted by**: ECMWF**	Date**: September 2023**	Test Category**: Interface testing**
Preconditions	• Cloud platforms up and running • Rancher deployed (through Terraform) • Kubernetes cluster created (through Terraform) • Rancher and Kubernetes credentials obtained		

	Related Component	Type	Message or API Call	Status	Remarks/comments
1	Rancher	REST API	List Kubernetes Clusters	Success	
2	Kubernetes cluster	REST API	Cluster Health	Success	

3.4.1.2 Monitoring

Table 5 - Monitoring interface test results.

Component***: Monitoring***	Conducted by**: CINECA**	Date**: September 2023**	Test Category**: Interface testing**
Preconditions	• Rancher-monitoring properly configured, up and running • S3 bucket to be mounted correctly • Kubernetes monitoring service and ingress up and running		

	Related Component	Type	Message or API Call	Status	Remarks/comments
1	Workflow Namespace	gRPC	Get platform information	Success	Intra-cluster retrieval of metrics from the monitoring component
2	OSS Platform	KAPI	Gather platform information	Success	Collect metrics from all the system components of the platform
3	S3	VPC	Upload	Success	Upload the Prometheus/Thanos data in a S3 bucket

4	Identity management	ConfigMap and REST	Authenticate users	Success	Connect Grafana to central KeyCloack to authenticate and authorize users to access metrics dashboards
5	GitLab CI/CD	ConfigMap, KAPI and REST	Deploy cluster components	Success	Deploy at creation time and update monitoring components on the Kubernetes clusters by leveraging on the EO4EU GitLab CI/CD component

3.4.1.3 Logging

Table 6 - Logging interface test results.

Component***: Logging***	Conducted by**: CINECA**	Date**: September 2023**	Test Category**: Interface testing**
Preconditions	• Logging-operator and OpenSearch tested with initial configuration • Logging operator services up and running		

	Related Component	Type	Message or API Call	Status	Remarks/comments
1	OSS Platform	KAPI	Get logging data	Success	Retrieve logging data from any system component of the main observability cluster
2	OpenStack Cinder CSI	REST, KAPI	Create volumes	Success	Use block storage volumes to save and retrieve logging data
3	GitLab CI/CD	ConfigMap, KAPI and REST	Deploy cluster components	Not ready yet	Deploy at creation time and update logging components on the Kubernetes clusters by leveraging on the EO4EU GitLab CI/CD component

3.4.2 Auxiliary and Support Integration

Table 7 - Monitoring interface test results.

Component***: Container Registry***	Conducted by**: Eng**	Date**: September 2023**	Test Category**: compotesting**
Preconditions	• GitLab properly configured, up and running • GitLab CI/CD properly configured • GitLab runners configured and available • GitLab CI/CD pipeline configured in the same repository		

	Related Component	Type	Message or API Call	Status	Remarks/comments
1	GitLab CI/CD	configMap	Push Image	Success	Image stored
2	GitLab CI/CD	R	Image retrieve	Success	Image retrieved

3.4.3 Platform Integration

3.4.3.1 Provision Manager

3.4.3.1.1 Pre-Processor

Table 8 - Pre-Processor interface test results.

Component: ***Pre-Processor***	Conducted by: **NKUA**	Date: **May 2023**	Test Category: **Interface testing**
Preconditions	• Apache Kafka properly configured, up and running • Related components must be up and running • S3 bucket to be mounted correctly • Provision service must be up and running		

	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Workflow Namespace	configMaps	GetConfigurationDetails	Success	Retrieve S3 config, kafka topic in-out, communication credentials, [Nekt] data sources
2		EOsrc	Execute data request scrpt	Success	Execute the request scripts towards the data sources.
3	S3	VPC	Upload	Success	Upload the raw data in the workflow S3 bucket. (archived and unarchived files)
4	Message Bus	M-p	SendNextComponentStatus	Success	Send to the next component that execution was completed, the folder and the form that the data are stored in S3 bucket

3.4.3.1.2 Post-Processor

Table 9 - Post-Processor interface test results.

Component: ***Post-Processor***	Conducted by: **NKUA**	Date: **May 203**	Test Category: **Interface testing**
Preconditions	• Apache Kafka properly configured, up and running • Related components must be up and running • S3 bucket to be mounted correctly • Provision service must be up and running • Elastic Search service must be up and running		

	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Workflow Namespace	configMaps	GetConfigurationDetails	Success	Retrieve S3 bucket name, kafka topic in-out, communication credentials, Elastiic Search credentials.
2	Kafka	M-C	GetPreviousComponentStatus	Success	Receive that the previous component has ended its job, what kind of data are produced and where they are stored.
3	S3	VPC	IterateBucket	Success	Iterate S3 bucket to find file that can be uploaded in the Elastic Search
4	Elsaticsearch	Rest	UploadData	Success	Upload data to the Elastic Search

5	Message Bus	M-p	SendNextComponentStatus	Success	Send to the next component that execution was completed, the folder and the form that the data are stored in S3 bucket

3.4.3.1.3 FaaS

Table 10 - FaaS interface test results.

Component***: FaaS***	Conducted by**: NKUA**	Date**: May 2023**	Test Category**: Interface testing**
Preconditions	• Apache Kafka properly configured, up and running • Related components must be up and running • S3 bucket to be mounted correctly • Provision service must be up and running		

	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Workflow Namespace	configMaps	GetConfigurationDetails	Success	Retrieve S3 bucket name, kafka topic in-out, communication credentials.
2	Kafka	M-C	GetPreviousComponentStatus	Success	Receive that the previous component has ended its job, what kind of data are produced and where they are stored.
3	S3	VPC	GetData	Success	Download data from S3 bucket
4	FaaS	REST	SendData	Success	Send the data to the OpenFaaS function service for processing
5	Message Bus	M-p	SendNextComponentStatus	Success	Send to the next component that execution was completed, the folder and the form that the data are stored in S3 bucket

3.4.3.1.4 Provision Service

Table 11 - Provision Service interface test results.

Component***: Provision Service***	Conducted by**: NKUA**	Date**: Feb 2016**	Test Category**: Interface testing**
Preconditions	• Apache Kafka properly configured, up and running • Related components must be up and running • S3 bucket to be mounted correctly		

	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Kafka	M-C	GetDirectivesFormWFE	Success	Receive the user defined yaml files and the needed configurations for the

					deployment of the workflow.
2	Kubernetes cluster	KAPI	CreateUniqueNamespace	Success	Message was consumed by Resource controller since Experiment Controller does not yet exists Message successfully received by receiving component
3	Kubernetes cluster	KAPI	CreateNamespaceSecrets	Success	Create the needed secrets in the above namespace.
4	Kubernetes cluster	KAPI	CreateNamespaceConfigMaps	Success	Create the needed configuration files (configMaps) in the above namespace.
5	Kubernetes cluster	KAPI	CreateNamespaceDemployments	Success	Deploy the components designed in the WFE
6	Kubernetes cluster	KAPI	CreateNamespaceServices	Success	Deploy the services required or designed for the workflow
7	Kubernetes cluster	KAPI	CreateNamespaceIngress	Success	Deploy the ingress services required for the workflow
8	Message Bus	M-P	SendConfig	Success	Send configuration information to the Fusion component(s)
9	Message Bus	M-P	SendConfig	Success	Send configuration information to the ML component(s)
10	Message Bus	M-P	DepoolymentStatus	Not tested	Report message for the deployment of the workflow.
11	Message Bus	M-C	DeleteWorkflow	Not tested	Control message from WFE to destroy workflow after processing.
12	Kubernetes cluster	KAPI	DeleteUniqueNamespace	Not tested	Delete Namespace
13	Kubernetes cluster	KAPI	DeleteSecrets	Not tested	DeleteSecrets
14	Kubernetes cluster	KAPI	Delete configMaps	Not tested	Delete configMaps
15	Message Bus	M-P	DeletionStatus		Report message for the deletion of the workflow.

3.4.4 Authentication SSO Integration

Table 12 - Authentication SSO interface test results.

Component: ***Authentication SSO***		Conducted by: **EBOS**		Date: **September 2023**	Test Category: **Interface testing**
Preconditions		• Have an account created on Keycloak instance			
	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	User Management Module (UMM)	HTTP	GetSsoLoginPage	Success	Redirects the current SSO login page when we try to open the app url.
2	UMM	API	ViewUsers	Success	View all users and details

4	UMM	API	CreateUser	Success	Creates new user (disabled)
5	UMM	API	EditUser	Success	Edit User details
6	UMM	API	DeleteUser	Success	Delete User
7	UMM	API	ViewApplications	Success	Shows all applications/clients registered so far
8	UMM	API	CreateApplication	Success	
9	UMM	API	EditApplication	Success	
10	UMM	API	DeleteApplication	Success	
11	UMM	API	ViewOpenIDC	Success	Shows Client OpenID Connect details
12	UMM	API	ManageGroups	Success	Edit, Create, Read, delete group and assign users to group
13	OpenEO API	API	/Auth/token	Success	Using the password grant flow to get access token from registered user credentials
14	OpenEO API	API	/Auth/userinfo	Success	Returns User details if the Bearer token is still valid
15	UMM	API	Manage User Resources	Not ready yet	
16	UMM	API	Manage User Roles	Not ready yet	

3.4.5 Fusion Engine Integration

Table 13 - Fusion Engine interface test results.

Component***: Fusion Engine***	Conducted by**: NKUA**	Date**: May2023**	Test Category**: interface testing**
Preconditions	• Apache Kafka properly configured, up and running Related components must be up and running S3 bucket to be mounted correctly		

Related Component		**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Message Bus	M-C	GetWorkflowConfiguration	Success	Receive user configuration, S3 bucket configuration, algorithms and next component to notify
		M-C	GetPreviousComponentStatus	Success	Receive that the previous component has ended its job and fusion can start
		O	UpdateS3Bucket	Success	Write in the S3 bucket the generated/trasformed data
		M-p	SendNextComponentStatus	Success	Send to the next component that pipeline was completed, the folder and the form that the data from fusion are stored in S3 bucket

3.4.6 DSL Engine Integration

Table 14 - DSL Engine interface test results.

Component***: DSL Engine***	Conducted by**: NKUA**	Date**: May2023**	Test Category**: interface testing**
Preconditions	• Apache Kafka properly configured, up and running. • Workflow Editor is up and running.		

Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1 Message Bus	M-C	GetDslModel	Success	Receive DSL model and compile it
	M-C	SendConfigurationYaml	Success	If received DSL model is valid send configuration YAML to Systems

3.4.7 AI ML Marketplace Integration

Table 15 - AI/ML Marketplace interface test results.

Component***: Provision Service***	Conducted by**: EBOS**	Date**: N/A yet**	Test Category**: Interface testing**
Preconditions	• PostgreSQL Database Server Set-Up and Programming (under Technical Development, Phase-B) • Work Flow Editor running and fully deployed • API Microservices Middleware API development and deployment (under Technical Development, Phase-B) • AI/ML Models prototyped and well-defined (HES-SO) • Work Flow Models prototyped and well-defined (NKUA) • OpenEO API to be well defined and fully integrated with the rest of the components (EBOS) • Cluster Network, and HTTPS URL Communication to be well-established		

	Related Component	**Type**	**Message or API Call**	**Status**	**Remarks/comments**
1	Work Flow Editor	N/A	API Call1	N/A	Phase B-Technical Development
2	AI/ML Models	N/A	API Call2	N/A	Phase B-Technical Development
3	WF Models	N/A	API Call3	N/A	Phase B-Technical Development
4	Microservices API	N/A	API Call4	N/A	Phase B-Technical Development
5	OpenEO API	N/A	API Call5	N/A	Phase B-Technical Development
6	Dashboard-Data Query	N/A	API Call6	N/A	Phase B-Technical Development
7	Dashboard – Data Visualization	N/A	API Call7	N/A	Phase B-Technical Development

3.4.8 Infrastructure as a Code Integration

Table 16 - IaC interface test results.

Component: ***Infrastructure as a Code***	Conducted by: **ECMWF**	Date: **September 2023**	Test Category: **Interface testing**
Software Configuration	• Terraform • Ansible • Gitlab		
Test Name:	***Infrastructure as a Code Verification***		
Preconditions	• Gitlab Version Control System Deployed • Gitlab Runners Deployed • Gitlab CI/CD Configuration Defined • Terraform Environment Created		
Related Requirements			
Tools Used	• Terraform • Rancher • Kubernetes • Gitlab • OpenStack		

Step	Action	Expected Result	Status	Remarks
1	Setup Gitlab Repository	Repository created	Success	
1	Add Terraform configuration files to Gitlab	Configurations onboarded	Success	
2	Trigger Gitlab CI/CD pipeline	Resources deployed	Success	

3.5 Verification scenarios results

In this section, the results of the executed verification scenarios of DX.X (chapter X) are explained. The template table, given and explained in section 3.2.1, was extended to better visualize the scenario steps and the results of them.

3.5.1 Platform Controller

3.5.1.1 Platform Manager

Table 17 - Platform Manager Verification test results.

Test ID: **SS-PM-T-001**	Conducted by: **ECMWF**	Date: **September 2023**	Test Category: **Verification Tests**
Hardware Configuration			
Software Configuration	• Terraform • Ansible • Rancher		

Test Name:		***Platform Manager up and running***		
Preconditions		• Gitlab CI/CD up and running • Cloud resources allocated		
Related Requirements				
Tools Used		• Terraform • Rancher • Kubernetes • Gitlab		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Add Terraform and Rancher configuration files to Gitlab	Configurations onboarded	Success	
2	Trigger Gitlab CI/CD pipeline	Rancher server deployed	Success	
3	Provision a Kubernetes cluster through Rancher's UI	Kubernetes cluster deployed	Success	
4	Deploy a NGINX Kubernetes pod with a persistent volume claim attached an associated public service and ingress	NGINX web service reachable	Success	

3.5.1.2 Monitoring

Table 18 - Monitoring Verification test results.

Test ID**: SS-MON-T-001**	Conducted by**: CINECA**	Date**: September 2023**	Test Category**: Verification Tests**
Hardware Configuration			
Software Configuration	• Kubernetes cluster		
Test Name:	***Monitoring up and running***		
Preconditions	• Kubernetes clusters up and running • S3 service up and running • Rancher server up and running • GitLab CI/CD up and running		
Related Requirements			
Tools Used	• S3 • Kubernetes • Rancher server • GitLab		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Add configuration in GitLab Repo	Configuration details successfully deployed	Success	Triggers subsequent deployments
2	Deploy Kubernetes Custom Resource Definitions in the target Kubernetes cluster	Monitoring Custom Resource Definitions correctly deployed	Success	Done with Rancher Helm charts
3	Deploy monitoring operators	Kubernetes monitoring operators up and running	Success	Done with Rancher Helm charts

4	Deploy monitoring component configuration	Components up and running and metrics flowing to the observer cluster and correctly visualized in Grafana dashboards with authentication enabled. Historical data stored and retrieved in/from S3 bucket	Success	

3.5.1.3 Logging

Table 19 - Logging Verification test results.

Test ID**: SS-LOG-T-001**	Conducted by**: CINECA**	Date**: September 2023**	Test Category**: Verification Tests**
Hardware Configuration			
Software Configuration	• Kubernetes cluster		
Test Name:	***Logging up and running***		
Preconditions	• Kubernetes clusters up and running • Cinder CSI plugin up and running • Rancher server up and running		
Related Requirements			
Tools Used	• S3 • Kubernetes • Rancher server		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Deploy Kubernetes Custom Resource Definitions in the target Kubernetes cluster	Monitoring Custom Resource Definitions correctly deployed	Success	Done with Rancher Helm charts
2	Deploy logging operators (rancher-logging and OpenSearch operator)	Kubernetes logging operators up and running	Success	Done with Rancher Helm charts
3	Deploy logging component configuration	Components up and running and logs flowing to the OpenSearch cluster and correctly visualized in OpenSearch dashboards with basic authentication enabled. Log data stored and retrieved in/from Cinder volumes.	Success	

3.5.2 Auxiliary and Support

3.5.2.1 Container Registry

Table 20 - Container Registry Verification test results.

Test ID: **SS-CR-T-001**	Conducted by: **ENG**	Date: **September 2023**	Test Category: **Verification Tests**
Hardware Configuration			
Software Configuration	• GitLab CI/CD		
Test Name:	***Container Registry up and running***		
Preconditions	• Gitlab up and running • GitLab CI/CD configured • GitLab runners configured and available • GitLab project available • Create a gitlab-ci yml file that execute dockerfile		
Related Requirements			
Tools Used	• GitLab CI/CD		

Step	Action	Expected Result	Status	Remarks
1	Push gitlab-ci.yml file in GitLab project	Image created	Success	

3.5.3 Platform Orchestrator

3.5.3.1 Provision Manager

3.5.3.1.1 Pre-Processor

Table 21 - Pre-Processor Verification test results.

Test ID: **SS-PRP-T-001**	Conducted by: **NKUA**	Date: **May 2023**	Test Category: **Verification Tests**
Hardware Configuration			
Software Configuration	• Kubernetes cluster		
Test Name:	***Analysis Tool will be able to query available data schemas***		
Preconditions	• Working message bus • Working S3 Bucket • WFE up and running • Provision Service up and running		
Related Requirements			
Tools Used	• S3 • Kubernetes		

Step	Action	Expected Result	Status	Remarks
1	PRP retrieves information from mounted configMaps and secrets	Configuration details successfully imported	Success	
2	PRP mounts S3 bucket	Dataset is accessible	Success	
3	PRP configures producer based on topic_out	Connection established	Success	

4	PRP executes requests scriots towards EO data sources	Data sources response with the requested datasets	Success	
5	PRP uploads data in archived and unarchived form in S3 bucket	Upload successful	Success	
6	Send message to the next components with information about the end of the PRP process and details for the data uploaded in the S3 bucket	Producer successfully send message	Success	

3.5.3.1.2 Post-Processor

Table 22 - Post-Processor Verification test results.

Test ID**: SS-PP-T-001**	Conducted by**: NKUA**	Date**: May 2023**	Test Category**: Verification Tests**
Hardware Configuration			
Software Configuration	• Kubernetes cluster		
Test Name:	***Analysis Tool will be able to query available data schemas***		
Preconditions	• Working message bus • Working S3 Bucket • WFE up and running • Provision Service up and running • Elastic Search Service up and running		
Related Requirements			
Tools Used	• S3 • Kubernetes • Elastic Search		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	PP retrieves information from mounted configMaps and secrets	Configuration details successfully imported	Success	
2	PP configures consumer and producer based on topic_in and topic_out	Connection established	Success	
3	PP mounts S3 bucket	Dataset is accessible	Success	
4	Iterate in S3 bucket for appropriate file	# Files found	Success	
5	Push files in Elastic Search	Push successful	Success	

3.5.3.1.3 FaaS

Table 23 - FaaS Verification test results.

Test ID**: SS-F-T-001**	Conducted by**: NKUA**	Date**: May 2023**	Test Category**: Verification Tests**
Hardware Configuration			
Software Configuration	• Kubernetes cluster		
Test Name:	***Analysis Tool will be able to query available data schemas***		

Preconditions	• Working message bus • Working S3 Bucket • WFE up and running • Provision Service up and running • OpenFaaS up and running			
Related Requirements				
Tools Used	• S3 • Kubernetes • OpenFaaS			
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	FaaS proxy component retrieves information from mounted configMaps and secrets	Configuration details successfully imported	Success	
2	FaaS proxy configures consumer and producer based on topic_in and topic_out	Connection established	Success	
3	FaaS proxy mounts S3 bucket	Dataset is accessible	Success	
4	Proxy sends distributed requests in FaaS function service	Successful processing with response	Success	
5	Proxy uploads response data in S3 bucket	Upload successful	Success	
6	Send message to the next components with information about the end of the FaaS process and details for the data uploaded in the S3 bucket	Producer successfully send message	Success	

3.5.3.1.4 Provision Service

Table 24 - Provision Service Verification test results.

Test ID**: SS-PS-T-001**	Conducted by**: NKUA**	Date**: May 2023**	Test Category**: Verification Tests**
Hardware Configuration			
Software Configuration	• Kubernetes cluster		
Test Name:	***Analysis Tool will be able to query available data schemas***		
Preconditions	• Working message bus • Working S3 Bucket • WFE up and running • Provision service defined configuration details configMaps		
Related Requirements			
Tools Used	• S3 • Kubernetes		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
	Loading the contents of the Provision Service configMap and initialize the KAPI	KAPI successfully initialised and the service has admin roles for the cluster (necessary for creating resources)	Success	

	Provision Service awaits input from the DSL engine.	Input received describing a user defined workflow.	Success	Initiating the workflow's creation
	Provision Service creates a general configmap containing configurations for all components to be deployed.	All deployed components successfully access this file.	Success	Message bus configuration is accessed by all in cluster components
	Creating secrets for S3 bucket authentication and access.	S3 Login successful for all in cluster components	Success	
	Creating additional configmap and secret for the Pre-Processor to authenticate and access the datasources.	Successful login and dowload the datasources (datasets)	Success	
	Creating additional configmap and secret for the Ppst-Processor to authenticate and access the ELS.	ELS login and send text data.	Success	
	Provision service sends configuration to system fusion topic	S3 Login successful	Success	
	FE mounts S3 bucket (per workflow)	Dataset is accessible	Success	
	FE configures consumer and producer based on topic_in and topic_out	Communication between the previous and the next component is established, and data streams are created.	Success	
	Provision service sends configuration to system ml topic	AI ML components initiated (with the corresponding inputted algorithms), S3 Login successful	Success	The AI ML component(s) initiate the various ML algorithms (this is relayed via the provision service from the WFE to the AI ML).
	AI ML mounts the S3 bucket (per workflow)	FE output stored on S3 bucket available	Success	
	AI ML configures consumer and producer based on topic_in and topic_out	Communication between the previous and the next component is established, and data streams are created.	Success	

3.5.4 Authentication SSO

Table 25 - Authentication SSO Verification test results.

Test ID**: SS-ASSO-T-001**	Conducted by**: EBOS**	Date**: September 2023**	Test Category**: Verification Tests**
Hardware Configuration	Cluster Network		
Software Configuration	Key Cloak		
Test Name:			

Preconditions		UMM, Key Cloak, openEO API		
Related Requirements				
Tools Used		UMM		
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Create and configure properly your client/application on UMM	SSO page appears when you attempt to open your client URL	Success	
2	Store or record all user operations with results	Every API call is recorded into a database	Success	This logs only UMM operations.
3	Manage Users, Applications and groups with the UMM User Interface	View at least users, applications and groups	Success	

3.5.5 Fusion Engine

Table 26 - Fusion Engine Verification test results.

Test ID**: SS-FE-T-001**	Conducted by**: NKUA**	Date**: May 2023**	Test Category**: Verification Tests**
Hardware Configuration			
Software Configuration	Kfp 1.8.22 Boto3 1.26.139 Python 3 Rasterio 1.3.7		
Test Name:	***Analysis Tool will be able to query available data schemas***		
Preconditions	• Working message bus • Working S3 Bucket • Working Data Analysis Tool		
Related Requirements			
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Provision service sends configuration to system fusion topic	Login successful	Success	
2	FE mounts S3 bucket	Dataset is accesible	Success	
3	FE configures consumer and producer based on topic_in and topic_out			

Table 27 - Fusion Engine in workflow Verification test results.

Test ID**: SS-FE-T-002**	Conducted by**: NKUA**	Date**: May 2023**	Test Category**: Verification Tests**
Hardware Configuration			

Software Configuration	Kfp 1.8.22 Boto3 1.26.139 Python 3 Rasterio 1.3.7			
Test Name:	***Analysis Tool will be able to query available data schemas***			
Preconditions	• Working message bus • Working S3 Bucket • Working Data Analysis Tool			
Related Requirements				
Tools Used				
Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	FE conume specific message from previous components	Login successful	Success	
2	FE configures the Kubeflow pipeline	Dataset is accessible	Success	
3	FE cruns user pipeline			
4	FE sends status in the Logger		Not tested	
5	When experiment is completed the data are uploded in S3 bucket			
5	FE produceskafka message to the next components			
6	FE terminates the specific workflow			

3.5.6 DSL Engine

Table 28 - DSL Engine Verification test results.

Test ID**: SS-DSLE-T-001**	Conducted by**: NKUA**	Date**: May 2023**	Test Category**: Verification Tests**
Hardware Configuration			
Software Configuration	• Java 18 • Xtext 2.30 • Docker		
Test Name:	***DSL Engine Input / Output functionality***		
Preconditions	• Working message bus • Workflow Editor is running		
Related Requirements			
Tools Used			

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	Workflow Editor sends DSL model	DSL model is compiled and configuration YAML gets dispatched	Success	
2	Workflow Editor sends invalid DSL model	DSL model is compiled and error message is sent	Success	

3.5.7 AI ML Marketplace

Table 29 - AI/ML marketplace Verification test results.

Test ID: **SS-Mark-T-001**	Conducted by: **EBOS**	Date: **N/A**	Test Category: **Verification Tests**
Hardware Configuration	• Local Premises Network Servers • Cluster Network • PostgreSQL Database Server		
Software Configuration	• Kubernetes cluster • Middleware-Microservices API • PostgreSQL Programming Set-up		
Test Name:	***Analysis Tool will be able to query available data schemas***		
Preconditions	• Working message bus • Working S3 Bucket • WFE up and running • Provision Service up and running		
Related Requirements			
Tools Used	• S3 • Kubernetes		

Step	**Action**	**Expected Result**	**Status**	**Remarks**
1	API Call1	N/A	N/A	Phase B - Development
2	API Call2	N/A	N/A	Phase B - Development
3	API Call3	N/A	N/A	Phase B - Development
4	API Call4	N/A	N/A	Phase B - Development
5	API Call5	N/A	N/A	Phase B - Development
6	API Call6	N/A	N/A	Phase B - Development

3.5.8 Infrastructure as a Code

Table 30 - Infrastructure as a code Verification test results.

Test ID: **SS-IaaC-T-001**	Conducted by: **ECMWF**	Date: **September 2023**	Test Category: **Verification Tests**
Hardware Configuration			
Software Configuration	• Gitlab CI/CD configuration • Gitlab CI/CD configuration file in terraform repository		
Test Name:	***Infrastructure as Code verification***		
Preconditions	• Gitlab up and running • Gitlab CI/CD pipeline set up • Cloud infrastructure up and running • Terraform repository ready to deploy Rancher		
Related Requirements			
Tools Used	Gitlab, Terraform, Rancher		

Step	Action	Expected Result	Status	Remarks
1	Git clone Terraform repository	Terraform repository cloned locally	Success	
2	Add small commit to repository (e.g. README file)	Gitlab pipeline triggered, Rancher server deployed on cloud	Success	
3	Check Gitlab pipeline logs for errors	No errors	Success	
4	Check Rancher reachability	Rancher reachable	Success	
5	Change terraform configuration in repository (e.g. virtual machine flavor), make a new commit and push	Gitlab pipeline triggered, Rancher server redeployed reflecting changes	Success	
6	Check Gitlab pipeline logs for errors	No errors	Success	
7	Check Rancher reachability	Rancher reachable	Success	
8	Check that Rancher has been redeployed with the changes in the latest commit	Rancher deployment reflects latest commit changes	Success	

4 Conclusion

This document contains the development of the components and functionalities of the System and Services of the EO4EU platform and details about their development process and roadmap. D3.2 provides a technical description, technologies and frameworks used for the development, input and output of each component but also a state-of-the-art section concerning the specific area of interest according to the component. Furthermore, elaborating with D4.7 in this document is presented the Integration, Validation, and Testing (IVT) process according to the development integration phase that is described in that document. This way each component justifies the IVT steps towards the staging phase of the platform integration.