TEADAL

D6.2 INTEGRATION REPORT

Revision: v.1.0

Work package	WP6
Task	T6.3, T6.4
Due date	31/08/2024
Submission date	31/08/2024
Deliverable lead	ALMAVIVA
Version	1.0
Authors	Bruno Feitas (UBIWHERE), Andrea Falconi (MARTEL), Samantha Hine (ALMAVIVA), Vincenzo Cirillo (ALMAVIVA), Sergio Sestili (ALMAVIVA)
Reviewers	Pierluigi Plebani (POLIMI), Alessio Carenini (CEFRIEL), Katherine Barabash (IBM)
Abstract	The deliverable focuses on reporting the work associated with the whole integration process of the platform with the testbed done during the project. It will describe the main phases of the integration process, highlighting the evidence from the outcomes of trials.
Keywords	TEADAL, Report, Integration, Deployment, Node, Cluster, Testbed, Trial, CI/CD, Pipeline

Grant Agreement No.: 101070186
Call: HORIZON-CL4-2021-DATA-01
Topic: HORIZON-CL4-2021-DATA-01-01
Type of action: HORIZON-RIA

Document Revision History

Version	Date	Description of change	List of contributor(s)
V 0.1	09/05/2024	ToC	Samantha Hine (*ALMAVIVA*)
V 0.2	27/06/2024	First draft	Bruno Feitas (*UW*), Samantha Hine (*ALMAVIVA*), Andrea Falconi (*MARTEL*)
V 0.3	25/07/2024	Complete contents, ready for review	Bruno Feitas (*UW*), Samantha Hine (*ALMAVIVA*), Andrea Falconi (*MARTEL*), Vincenzo Cirillo (ALMAVIVA), Sergio Sestili (ALMAVIVA)
V 1.0	31/08/2024	Final version	Bruno Feitas (*UW*), Samantha Hine (*ALMAVIVA*), Andrea Falconi (*MARTEL*), Vincenzo Cirillo (ALMAVIVA), Sergio Sestili (ALMAVIVA), Pierluigi Plebani (POLIMI), Alessio Carenini (CEFRIEL), Kathrine Barabash (IBM)

DISCLAIMER

Funded by the European Union

Funded by the European Union (TEADAL, 101070186). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

COPYRIGHT NOTICE

Project funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	**R**	
Dissemination Level		
PU	*Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)*	✓
SEN	*Sensitive, limited under the conditions of the Grant Agreement*	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

EXECUTIVE SUMMARY

This document presents the main phases of the integration process and the work associated with the integration of the TEADAL software with the testbed, also reporting the evidence of the trials that have been put in place on a pilot per pilot base.

As a key aspect of the TEADAL project, a software contribution process has been put in place, to integrate technological components into a unified platform. The process leverages a Git repository hosted on a Gitlab service to maintain the master code of the platform. The Kubernetes-based TEADAL platform articulates in a TEADAL cluster that organises service mesh and core services components. The Linux Operating System and Kubernetes provide the integration layer among the software components and the hardware resources, reducing complexity both in distribution and runtime processes.

The TEADAL CI/CD tools are based on Git and Kubernetes and leverage ArgoCD to easily distribute and update the TEADAL cluster on the testbeds. The CI/CD process envisages cloning the master “TEADAL.Node” repository to easily customise it according to the use case requirements and then deploying the customised TEADAL cluster on the testbed. Thanks to this configuration, once Kubernetes and ArgoCD are installed on the testbed, the TEADAL cluster deployment is fully automated and takes a few minutes to complete, increasing efficiency and facilitating the adoption of the TEADAL platform.

The whole process has been carefully tested on a pilot-by-pilot basis. A complete set of CI/CD pipelines has been defined for each pilot case. As each pilot case envisages a number of TEADAL Nodes featuring different organisations, multiple cloning of the master “TEADAL.Node” is required along with the related pipelines. Dedicated pipelines are required for the distribution of software on edge resources. For each pilot case, a trial has been carried out with a single pilot node deployment. The TEADAL platform integration has been demonstrated by loading a small dataset on the pilot node and then deploying and running a REST service exposing a single Federated Data Product. The current deliverable reports the evidence of each trial.

The complex of the trials that have been performed confirms that the deployment of the TEADAL platform is a matter of minutes and requires basic Linux and DevOps skills. The TEADAL platform usage is based on Federated Data Products, requiring programming skills with no language constraints.

Funded by the European Union

TABLE OF CONTENTS

Funded by the European Union

LIST OF FIGURES

Funded by the European Union

LIST OF TABLES

Funded by the European Union

ABBREVIATIONS

API	Application Programming Interface
CD	Continuous Delivery/Deployment
CI	Continuous Integration
FDP	Federated Data Product
GitLab	DevOps Platform and Services
GitOps	Git-based DevOps approach
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IaC	Infrastructure as Code
K8s	Kubernetes
OPA	Open Policy Agent
OpenAPI	Standard specification for REST APIs
OTA	Over-The-Air
sFDP	shared Federated Data Product
TEE	Trusted Execution Environment
ToC	Table of Contents
VM	Virtual Machine

Funded by the European Union

1 INTRODUCTION

The TEADAL (“Trustworthy, Energy-Aware federated Data Lakes along the computing continuum”) project mission is to provide key technologies to enable sharing data and computation among the cloud-edge continuum (gravity) managed by a single organisation and in a data lake federation environment (friction) where the nodes of such federation are managed by different entities. All this by enabling private, confidential, and energy-efficient data management.

The main goal of the project is to develop a software toolset for data lake technologies to provide trusted, verifiable, and energy-efficient data flows, both in a stretched data lake and across a trustworthy mediator-less federation of data lakes, based on a shared approach for defining, enforcing, and tracking privacy/confidentiality requirements balanced with the need for energy reduction.

The main phases of the integration of the platform with the testbed done during the project are described in this deliverable, reporting the work associated with the integration process and highlighting the evidence from trials.

The document consists of 7 sections, including this short Introduction.

The 2nd section describes the TEADAL Node Integration and Runtime. Starting from the definition of Testbed Site, TEADAL Cluster and TEADAL Node, the section provides an overview of the runtime architecture of a TEADAL Node, of the deployment process and the TEADAL repository. Finally, a reference to the guides that have been produced to help with the integration of TEADAL software with pilot testbeds is given.

The 3rd section describes TEADAL CI/CD Tools that leverages GitLab, Kubernetes and ArgoCD to put in place simple and efficient integration and delivery processes.

The 4th section details TEADAL CI/CD Processes, describing the structure of the software repositories and the software update and deploy processes.

The 5th section provides a pilot per pilot description of CI/CD Pipelines.

The 6th section describes the Pilot Nodes Deployment, reporting the evidence of the trials that have been performed to demonstrate the integration among TEADAL software and pilot testbeds for the deployment of a pilot node.

The 7th section reports the Conclusions.

To provide references and evidence, the document links external contents related to the TEADAL project operations. At the present, to access linked contents a registration could be required.

Funded by the European Union

2 TEADAL NODE INTEGRATION AND RUNTIME

The TEADAL Node is the building block used to implement the different Pilot Testbeds in their specific topology[1]. This section describes how the TEADAL Node has been realised through the integration of many components, representing the TEADAL Node different abstraction layers.

2.1 TESTBED SITE

A TEADAL Testbed Site encompasses the hardware and infrastructure on which a TEADAL Cluster is deployed. This hardware includes the physical machines or virtualized environments necessary to run the TEADAL data lake services. The Testbed Site is where the provisioning of machines occurs, forming the foundational layer of the TEADAL architecture.

2.2 TEADAL NODE

The TEADAL Node integrates essential components such as ArgoCD and Istio within a containerized environment. The TEADAL Node Baseline includes all the TEADAL tools and serves as the foundation for further development and customization specific to each cluster's needs.

Each TEADAL node includes a backbone software stack on which cluster-specific data services run. The backbone stack includes general-purpose data lake components (Kubernetes, Istio, ArgoCD, etc.) as well as TEADAL-specific components (Catalogue, Advocate, TEE, Pipelines, Policies, etc.) that allow producers and consumers to share data in a trustworthy and secure way, according to agreed-upon governance, privacy and energy-efficiency policies. Above this backbone sit local, cluster-specific data products and services—i.e., federated data products (FDPs), shared federated data products (SFDP), etc.

2.3 TEADAL CLUSTER

A TEADAL cluster includes both the hardware and software deployed to run an instance of a TEADAL data lake. Notably, the software implements the various TEADAL tools and services that allow multiple TEADAL clusters to be joined in a federation where producers and consumers can share data in a trustworthy and secure way, according to agreed-upon governance, privacy and energy-efficiency policies[2]. These tools and services are part of each TEADAL cluster whereas hardware, data products and corresponding data services (FDPs and SFDPs[3]) typically differ from cluster to cluster. Each TEADAL cluster is instantiated and then subsequently managed using an Infrastructure-as-Code approach.

[1] For a logical description of the TEADAL Node, please refer to Deliverable 6.1 "Testbed Design"

[2] Please refer to Deliverable 2.2 "Pilot Cases' Intermediate Description And Initial Architecture Of The Platform", Deliverable 4.1 "Stretched Data Lakes First Release Report" and Deliverable 5.1 "Trustworthy Data Lakes Federation First Release Report" for details

[3] Please refer to Deliverable 3.1 "Gravity And Friction-Based Data Governance" for details

Funded by the European Union

From a conceptual standpoint, a TEADAL Cluster is composed of several layers of processes and hardware, arranged hierarchically. Higher layers utilise the functionality provided by lower layers, while lower layers do not depend on higher layers[4].

Referring to Figure 1 we examine the technological specification of each layer in turn, from the bottom up.

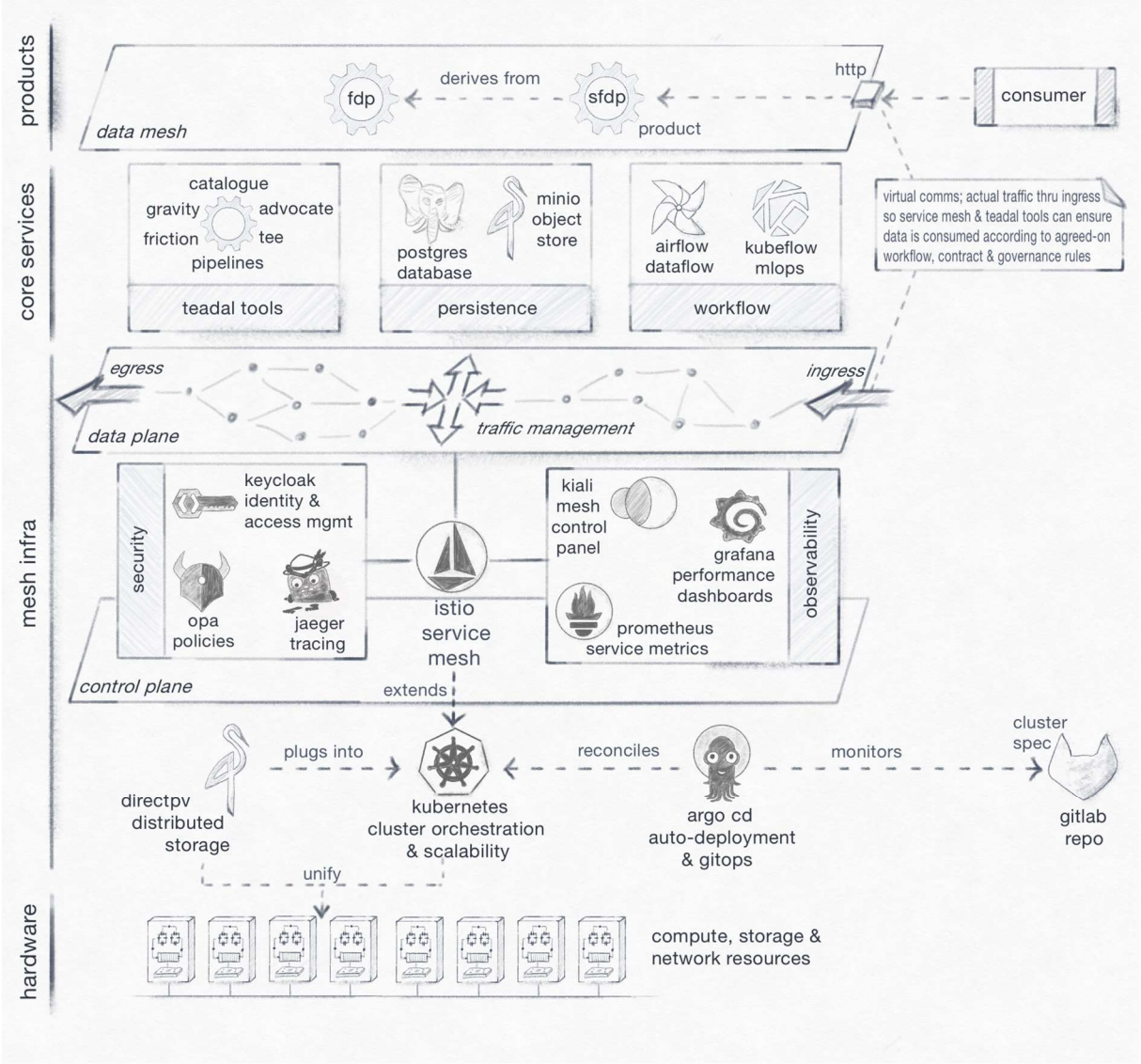

FIGURE 1: TEADAL CLUSTER TECHNOLOGIES

The *hardware layer* is the lowest layer. This is the cluster hardware (computers, network) on which all the TEADAL cluster software runs. In the case of a public cloud deployment, the hardware would typically be virtualized, whereas physical machines would be provisioned for an on-premises scenario. In the simplest case, the whole cluster can run on just one machine, whereas more computation-intensive scenarios require several machines.

The *mesh infrastructure* layer interfaces with the hardware layer to provide the service mesh functionality, with Kubernetes at its core for cluster operation, running on a Linux based operating system. This layer manages computational resources and orchestrates the

[4] For a complete description of the data and service mesh runtime architecture, please refer to Deliverable 2.3

Funded by the European Union

deployment and operation of services by means of containers. Kubernetes interfaces with DirectPV to create a distributed storage facility out of the disks attached to each node whereas Istio complements Kubernetes with mesh control and data planes. The Istio control plane manages a network of proxies that form the Istio data plane, which captures and processes service traffic. The Istio control plane is responsible for overall system management and orchestration, such as scheduling, scaling, and maintaining the desired state of the system. In contrast, the Istio data plane is responsible for the actual movement of packets through the network, handling the flow of application traffic based on the internal policies defined by the Istio control plane. This allows to augment service functionality at runtime without requiring any modifications to the services themselves. The TEADAL cluster exploits this to transparently route and balance service traffic, secure communication and access to service resources (through Keycloak and OPA), and monitor service operation (through Kiali, Prometheus and Grafana). Finally, the mesh infrastructure includes Argo CD, a GitOps continuous delivery tool for Kubernetes, to monitor the cluster Git repository in order to automatically reconcile the desired deployment state declared in the repository with the actual live state of the cluster.

Running on the *mesh infrastructure*, the *core services* layer provides TEADAL baseline functionality which enables federated data products. In this layer, PostgreSQL and MinIO provide database and object storage functionality, respectively. Workflow services are also included: Kubeflow for managing machine learning operations and Airflow for engineering data pipelines. Last but not least, the core services layer hosts the TEADAL-specific tools that allow multiple TEADAL nodes to be joined in a federation where producers and consumers can share data in a trustworthy and secure way, according to agreed-upon governance, privacy and energy-efficiency policies. Catalogue, Advocate, Trusted Execution Environment (TEE), Pipelines, and Policies (security, gravity and friction) are all examples of TEADAL services and tools in the core services layer.

Finally, *products* is the top layer, hosting cluster-specific data products and services—i.e., federated data products (FDPs), shared federated data products (SFDP), etc. As detailed in D3.1, a federated data product (FDP) extends the notion of data mesh product to cater for sharing data in a data lake federation according to the governance rules of that federation. A shared federated data product (SFDP) encapsulates a consumer-producer agreement (contract) about sharing a part of an FDP and provides the means for the consumer to process the shared data only within the bounds of the agreed-upon contract.

2.4 DEPLOYMENT

A TEADAL cluster is instantiated and then subsequently updated through a GitOps approach whereby the desired cluster runtime is declared in an online Git repository and a dedicated GitOps cluster service reconciles the desired runtime with the actual cluster state. Thus, there is a Git repository associated with every TEADAL cluster and, as briefly mentioned earlier, there is an ArgoCD service in that cluster which monitors the Git repository in order to automatically reconcile the desired deployment state with the actual live state of the cluster.

The deployment state in the Git repository is declared through a set of YAML files which Kustomize[5] can process. Each of these files declares a desired instantiation and runtime configuration for some of the components in the TEADAL cluster. Collectively, the files at a given Git revision describe the deployment state of the entire TEADAL cluster at a point in time. Changes to the live system are triggered through an automated workflow which the cluster administrator initiates by creating a new revision of some configuration files (in YAML

[5] Kustomize (https://kustomize.io/) is a tool to create Kubernetes cluster configuration resources modularly by assembling and extending Kubernetes resource definitions in YAML files.

Funded by the European Union

format) in the Git repository. On detecting a new revision, ArgoCD transitions the cluster to the new desired state. The diagram in Figure 2 exemplifies the GitOps workflow.

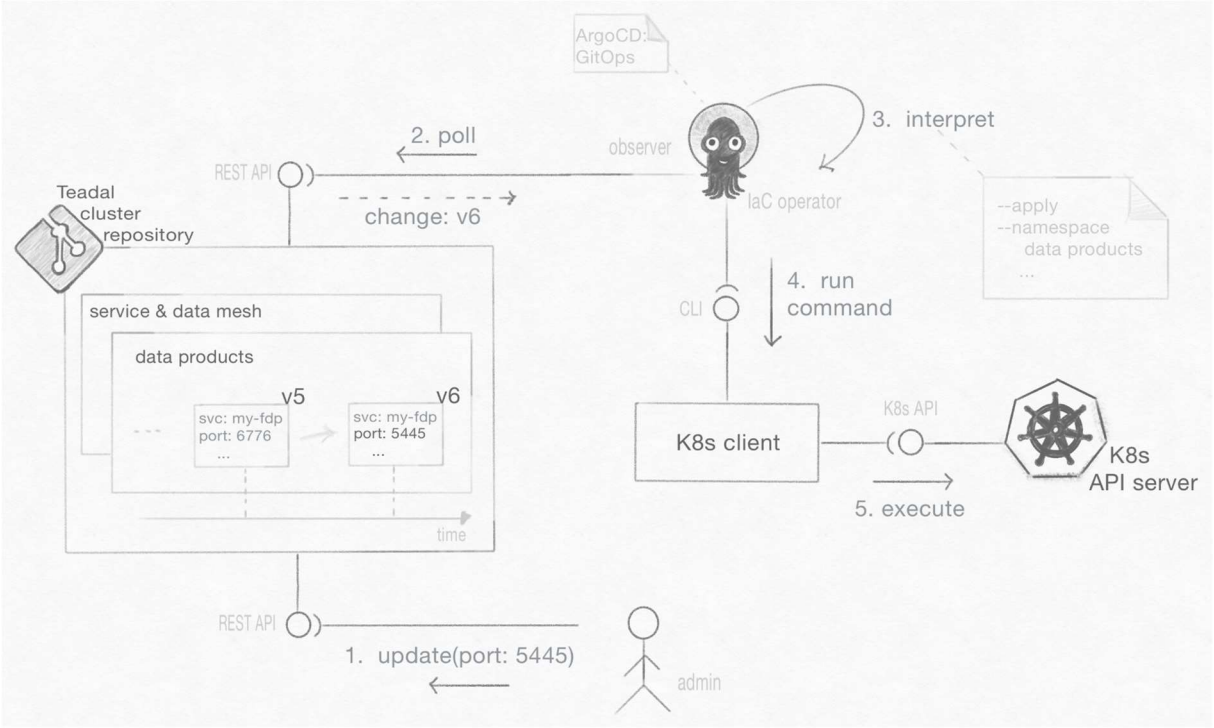

FIGURE 2: TEADAL GITOPS WORKFLOW EXAMPLE

Indeed, the diagram depicts a typical scenario where the cluster administrator carries out a change to a data service. As can be seen, the Git repository contains descriptors for an FDP named *my-fdp* as well as other descriptors, not explicitly shown, for the service and data mesh components in the various cluster runtime layers mentioned earlier. The latest Git revision is *v5* where the FDP service port is 6776. The administrator changes the port to 5445, making a new Git revision *v6*. ArgoCD periodically polls the Git repository to detect any new revisions. Thus, shortly after the administrator pushed revision `v6` to the Git repository, ArgoCD realises that the current cluster runtime state refers to a stale revision, *v6*, whereas *v6* is the latest. Hence, ArgoCD proceeds to interpret the stanzas in the YAML file as a command line that the Kubernetes client can understand. After assembling the required command, ArgoCD invokes the Kubernetes client with it. In turn, the Kubernetes client calls the Kubernetes API which finally triggers the desired deployment actions on the live cluster, resulting in the deployment state to reflect the YAML configuration at revision *v6*—i.e., *my-fdp*'s port is now 5445.

2.4 TEADAL BASELINE REPOSITORY

Each TEADAL Cluster has a corresponding Git repository and ArgoCD service to manage deployments. The baseline software stack (service mesh, TEADAL tools) is developed in a master repository (TEADAL.Node). Each cluster repository is a fork of this master, allowing inheritance of baseline components and customization for specific deployments. This setup ensures consistency and simplifies maintenance across different pilot rollouts. Below follows a more structured and detailed example.

Thus, for each TEADAL cluster deployed as part of a TEADAL pilot rollout there is a corresponding Git repository and ArgoCD service running in that cluster that reconciles the Git repository with the cluster. For example, the cluster for the viticulture pilot is deployed from the

Funded by the European Union

Smart Viticulture TEADAL.Node repository[6] and runs its own Argo CD service which monitors the *Smart Viticulture TEADAL.Node* repository and, upon detecting a change in the repository, reconfigures the viticulture cluster to match the updated deployment declarations in the repository.

Now, any two pilot rollout repositories need to include the same baseline software stack — service and data mesh, TEADAL tools, etc. In other words, the baseline software stack comprises the components in the *mesh infrastructure* and *core services* layers. As detailed earlier, components of the baseline software stack include service mesh elements like Istio, monitoring tools like Prometheus and Grafana as well as TEADAL-specific tools such as the Catalogue and Advocate. Almost always, the stack will be the same for any two given pilot rollouts, hence the code in their respective repositories will also be the same.

For this reason, the baseline software stack is developed and managed in a master repository (*TEADAL.Node*) and each pilot rollout repository is a fork of the master which contains data products and services specific to the pilot rollout. Forking allows each pilot rollout repository to inherit the baseline components from TEADAL.Node, while also enabling customization for specific deployments. This arrangement avoids duplicating the baseline software stack in each pilot rollout repository and provides the means to easily propagate any change from the master repository to the pilot rollout forks as illustrated in the diagram in Figure 3.

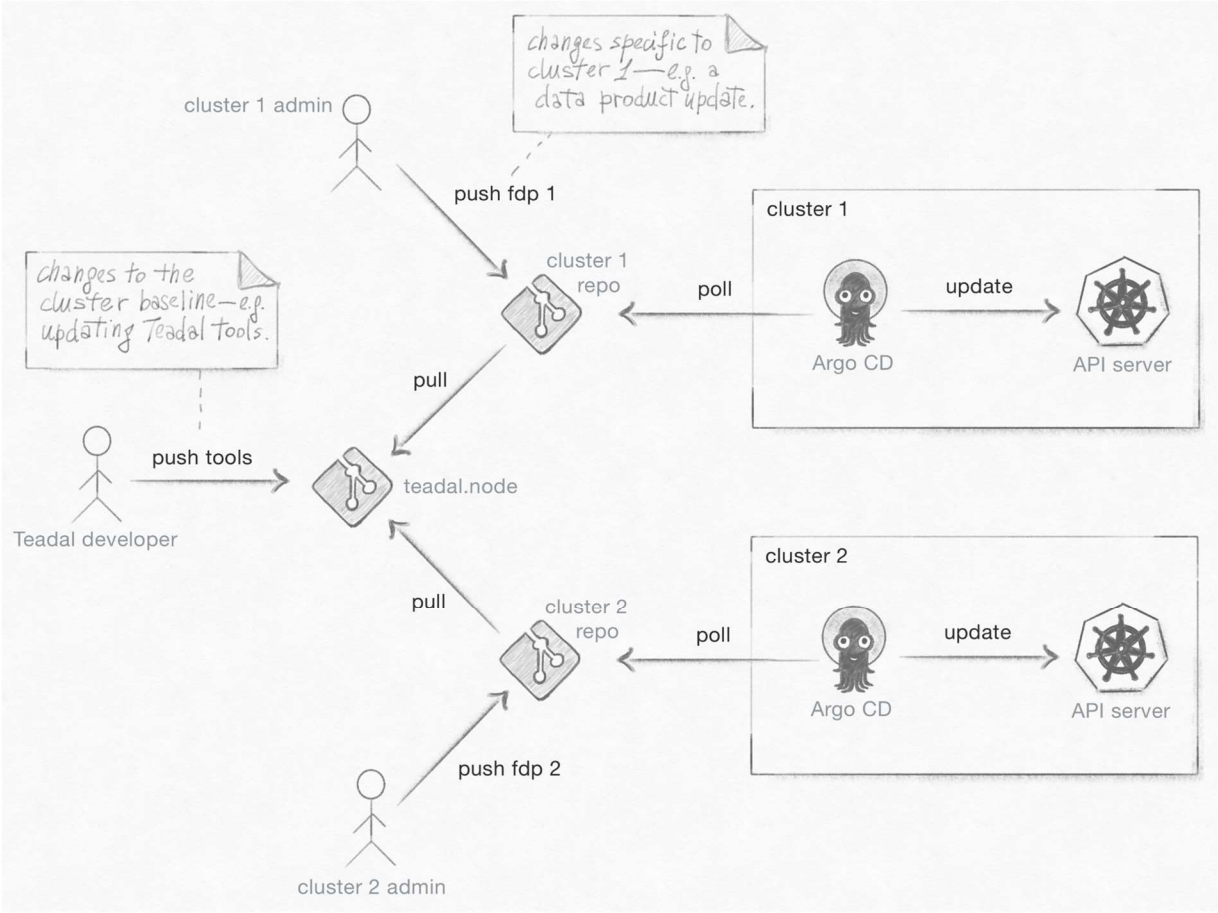

FIGURE 3: TEADAL.NODE FORKING EXAMPLE

[6] https://gitlab.teadal.ubiwhere.com/teadal-pilots/viticulture-pilot/smart-viticulture-teadal-node

Funded by the European Union

Thus, when a new feature or update is added to the master repository, it can be seamlessly integrated into all forks, ensuring consistency and reducing maintenance overhead.

2.5 INTEGRATION PROCESS

Developers follow a typical cloud-native workflow to integrate their software into a TEADAL cluster. As most TEADAL functionality is delivered through cloud (micro)services, developers typically carry out several steps to make their software available in a TEADAL cluster as a cloud service. First, the software is packaged in container images and these images published to an online registry. Then Kubernetes resources are developed with Kustomize and added to the Git repository from which the cluster is deployed. These Kubernetes resources specify how to download and run the published images and typically configure service storage, traffic routing, connections to other services, identity and access management, GitOps deployment, and possibly other cluster resources.

Although the actual details of the integration procedure may vary considerably from service to service, the conceptual workflow is mostly the same. Hence, to make the integration procedure somewhat more concrete, the TEADAL core developers have implemented several working examples of how to integrate services, in particular FDPs and SFDPs. Detailed documentation accompanies these examples, and it can be found in the TEADAL node repositories:

- Deployment Guide: https://gitlab.teadal.ubiwhere.com/teadal-tech/teadal.node/-/blob/main/docs/QuickStart.md
- Integration Process Guide: https://gitlab.teadal.ubiwhere.com/teadal-tech/integration.guide/-/blob/main/integration.md

Following the Deployment Guide, the deployments process of each node articulates the following steps:

- Linux environment setup
- Manual deployment of Kubernetes
- Manual deployment of Istio
- Manual deployment of Argo CD
- Automated deployment of TEADAL tools from the pilot repository, via ArgoCD

The above-mentioned Integration Process Guide describes the typical cloud-native workflow to integrate pilot software into a TEADAL cluster.

Funded by the European Union

3 TEADAL CI/CD TOOLS

On TEADAL, CI/CD process helps TEADAL developers to avoid bugs and code failures while maintaining a continuous cycle of software development and updates. As TEADAL grows, features of CI/CD can help decrease complexity, increase efficiency, and streamline workflows.

Because CI/CD automates the manual human intervention traditionally needed to get new code from a commit into production, downtime is minimised, and code releases happen faster. With the ability to quickly integrate updates and changes to code, developers' feedback can be incorporated more frequently and effectively, which means positive outcomes for the TEADAL code base.

The CI in CI/CD refers to continuous integration, an automation process for developers that facilitates the more frequent merging of code changes back to a shared branch, or “trunk”. As these updates are made, automated testing steps are triggered to ensure the reliability of merged code changes.

CI (Continuous Integration) states that the entire code for an application should be kept in a common repository so that whenever the developer checks the code into the repository, a script is triggered that picks the latest code from the repository, integrates it with the existing code and runs the test cases designed according to the application. Multiple CI tools are available in the market like Jenkins, Bamboo, GitLab, Subversion, etc. For CI to work, the tool must be integrated with a source code management system. Git is one such common tool that developers widely use for maintaining the versioning of code, GitLab is a Git hosting service used by TEADAL. Each developer works on its branch, which is cut from a master branch. Whenever a developer implements the functionality or fixes a bug, it raises a merge request from the current branch to the master branch. On the TEADAL Git repository, the merge request can be accepted by the Developer, Maintainer or Owner of the repository. This acceptance can be done via the GitLab UI. A script to execute test cases is triggered which runs all the unit tests written. If the code passes all the test cases, the merge is completed, otherwise the merge is rejected. Every developer is responsible for merging his code into the main branch. This adds a sense of accountability on the developer such that one needs to make sure that his changes should not impact the build but should not hamper fellow developers' work.

The CD in CI/CD refers to continuous delivery or continuous deployment, which are related concepts that sometimes get used interchangeably. Both are about automating further stages of the pipeline, but they’re sometimes used separately to illustrate just how much automation is happening.

CD (Continuous Delivery) is defined as an ability to deploy new features and bug fixes into the live server as and when required. The CD part is executed after a successful CI where all the updated code is integrated to the master branch. A script runs that picks the code from the master branch, prepares the build, and deploys it to a test environment or a production environment. This way the developers can test the code beyond unit tests so that updates across multiple parameters such as UI testing, integration testing, etc. can be tested in order to identify the issues pre-emptively. With continuous delivery, there also comes a term called continuous deployment and the major difference between these is that there is no manual intervention or confirmation required when following continuous deployment which means that with every code check-in, a new build will be deployed onto the specified server.

Continuous deployment in TEADAL is assured by using GitOps, which means that a Git repository is used as the single source of truth for the status of the infrastructure. This

Funded by the European Union

repository, which is on GitLab, contains configuration files that describe the TEADAL architecture in a format appropriate to be parsed by the CI/CD tools, IaC (Infrastructure as Code). Like this, it's possible to guarantee that each deployment is in an easily auditable known state. The GitOps deployment tool used in TEADAL is ArgoCD, this has been adopted to simplify the deployment, the customisation, and the maintenance of the TEADAL Nodes. Details on this can be found in a subsequent paragraph/section dedicated to ArgoCD.

3.1 GITLAB

GitLab is a web-based Git hosting service that provides open and private repositories, issue-following capabilities, and wikis, as presented in Figure 4. It is a complete DevOps platform that enables professionals to perform all the tasks in a project, from project planning and source code management to monitoring and security. Additionally, it allows teams to collaborate and build better software[7]. It provides features such as CI/CD pipelines. It does not provide any support for additional plugins to be installed and the entire configuration is done on the pipeline itself.

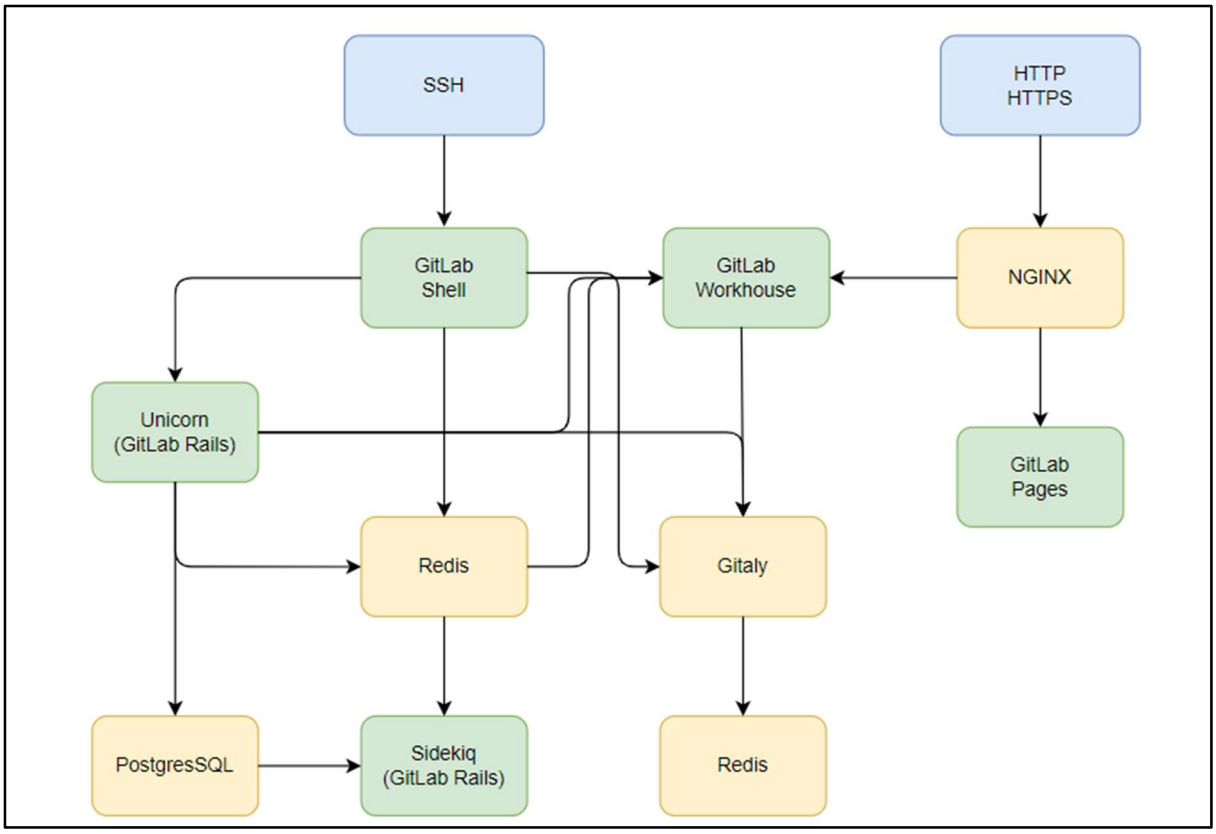

FIGURE 4: GITLAB ARCHITECTURE

Teams can build software faster, automate software delivery, shorten cycle times, and increase developer productivity. It can also scan for vulnerabilities with every push code automatically.

GitLab service helps the engineering teams remove toolchain complexity and accelerate DevOps adoption. It enables the team to develop an application that allows developers to

[7] https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab

Funded by the European Union

manage all efforts with one UI and simplifies the administration by employing one central entry point for all repositories.

GitLab advantages include:

- Version control and repository management based on Git
- Issue management, bug tracking and boards
- Code Review functionality and Review Apps tool
- CI/CD tool
- Code Quality (Code Climate)
- ChatOp tool (Mattermost)
- Service Desk (Ticketing System)

For CI/CD, TEADAL relies on GitLab services such as GitLab CI Pipelines and GitLab Container Registry. This improves collaboration and simplifies management, leading to increased efficiency and better project outcomes.

As outlined in the previous section, UBIWHERE-provided GitLab service hosts a master TEADAL.Node repository used by TEADAL developers to commit code[8]. The master repository is cloned for each TEADAL cluster. Within TEADAL project operations, the cloned repositories are grouped by pilot case and these groups are linked in a pilot project area[9]. Each pilot leverages the TEADAL tools from the central repository and enriches its own repository with custom application, according to the pilot case requirements.

3.2 KUBERNETES

Kubernetes, originally developed by Google, is an open-source container orchestration tool, its architecture is in Figure 5. Essentially, it manages containers, which can be Docker containers or other containerization technologies. Kubernetes helps to manage applications that consist of hundreds or even thousands of containers. These applications can run on physical machines, virtual machines, in the cloud, or even in hybrid environments.

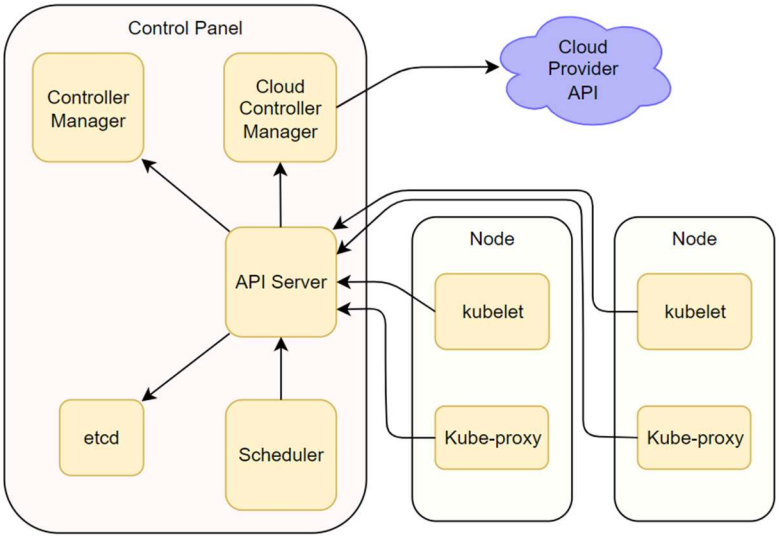

FIGURE 5: KUBERNETES ARCHITECTURE

[8] https://gitlab.teadal.ubiwhere.com/teadal-tech/teadal.node
[9] https://gitlab.teadal.ubiwhere.com/teadal-pilots

Funded by the European Union

Kubernetes makes it possible to manage the cluster and orchestration. For TEADAL, MicroK8s is used. MicroK8s is a way to get Kubernetes up and running with self-healing high availability, transactional OTA updates, and secure sandboxed *kubelet* environments, quickly spin nodes up on CI/CD processes and reduce production maintenance costs. MicroK8s is small, simple, and secure. It also includes a curated collection of manifests for common Kubernetes capabilities and services, like, Service Mesh (Istio, Linkerd), serverless (Knative), monitoring (Fluents, Prometheus, Grafana, Metrics), and automatic updates to the latest Kubernetes version[10]. This lightweight version of Kubernetes makes TEADAL as lightweight as it can be by providing a minimalistic and efficient K8s distribution that requires fewer system resources[11].

3.3 ARGOCD

ArgoCD is a declarative continuous delivery tool for Kubernetes. It follows the GitOps pattern of using Git repositories as the source of truth for defining the desired application state.

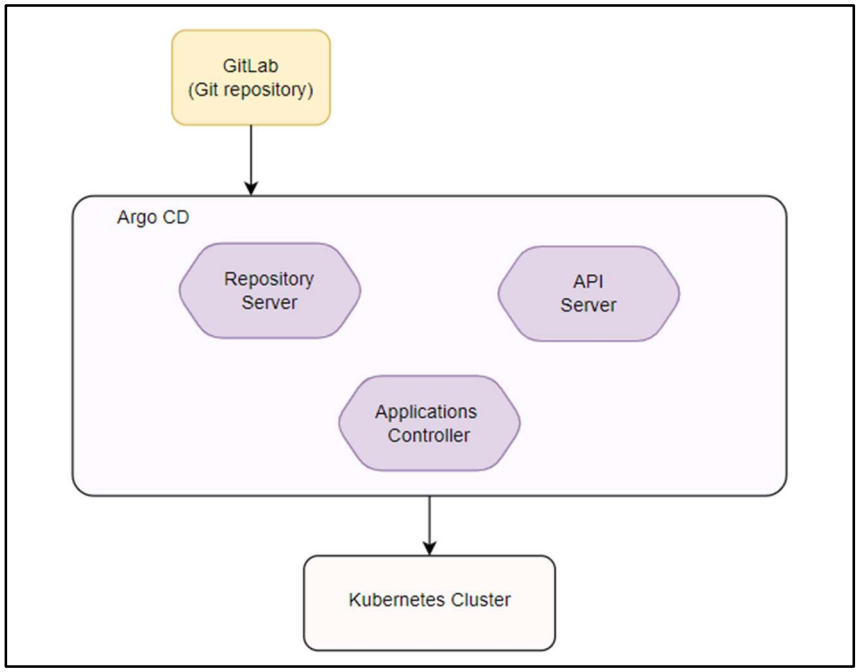

FIGURE 6: ARGOCD ARCHITECTURE

ArgoCD automates the deployment of the desired application states in the specified target environments. Application deployments can track updates to branches, tags, or pinned to a specific version of manifests at a Git commit. It's possible to see the ArgoCD architecture in Figure 6.

ArgoCD is implemented as a Kubernetes controller that continuously monitors running applications and compares the current, live state against the desired target state (as specified in the Git repo). This helps to ensure the desired state is maintained, reducing configuration drift and enhancing reliability. A deployed application whose live state deviates from the target state is considered "*OutOfSync*". ArgoCD reports & visualises the differences while providing

[10] https://github.com/canonical/microk8s
[11] https://canonical.com/blog/introduction-to-microk8s-part-1-2

Funded by the European Union

facilities to automatically or manually sync the live state back to the desired target state. This capability allows for faster detection and correction of issues, improving system stability and developer productivity. Any modifications made to the desired target state in the Git repo can be automatically applied and reflected in the specified target environments[12].

Inside TEADAL, ArgoCD works as a GitOps IaC tool for managing the MicroK8s cluster. ArgoCD was chosen for its robust automation and integration capabilities, which align well with the needs of managing a dynamic and scalable environment. This software is not centrally deployed. Instead, each TEADAL node will host its own instance of ArgoCD, configured according to its needs. This decentralised approach allows for greater flexibility and autonomy of each node, ensuring that configurations can be tailored to specific requirements while maintaining consistency across the entire infrastructure.

[12] https://argo-cd.readthedocs.io/en/stable/

Funded by the European Union

4 TEADAL CI/CD PROCESSES

TEADAL CI/CD workflow involves three tools: GitLab, ArgoCD, and Kubernetes. These tools provide services that are able to work together and complete the CI/CD workflow of the TEADAL node. It is possible to summarise the CI/CD processes in five sections (see Figure 7):

- Developer Git commit to GitLab
- GitLab request to API Handler
- API Handler trigger to ArgoCD
- ArgoCD instructs changes
- HELM packages the application with the updates and makes it available to deploy

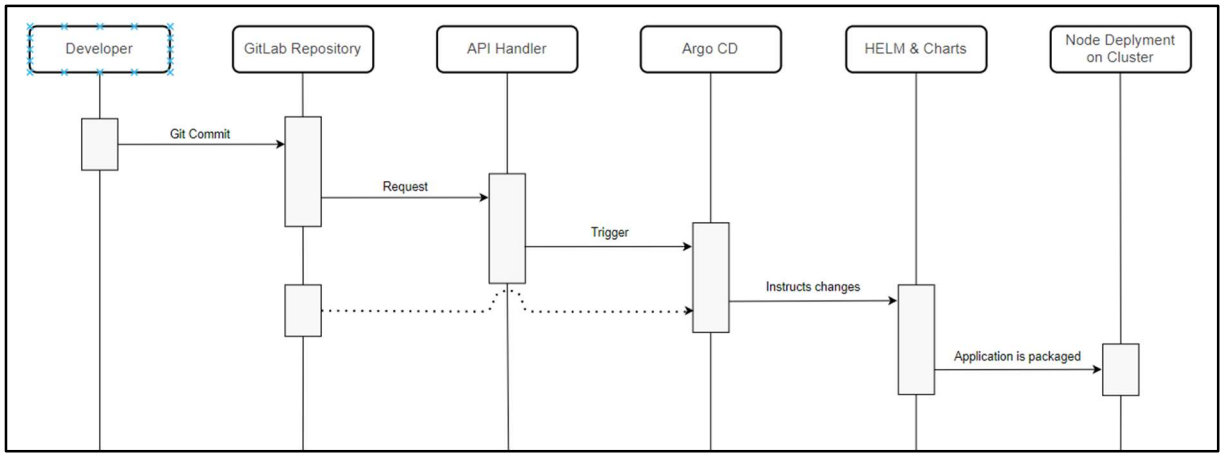

FIGURE 7: TEADAL CI/CD SEQUENCE

The CI/CD Process on TEADAL starts with the Developer. The Developer interacts with the TEADAL Git repository that leverages Gitlab services provided by UBIWHERE. GitLab services provide the main repository of the TEADAL node and are responsible for the CI process.

The Developer can commit code changes to the Git repository, which is hosted on this link: https://gitlab.teadal.ubiwhere.com. Inside the Git repository exists a lot of different repositories, each with its function that can be updated by their group members.

TEADAL has an API Handler that acts as an intermediary component, responsible for receiving Webhooks from GitLab communicating that code has been committed. This communication is an HTTP callback that is triggered by specific events. ArgoCD periodically polls the Git repository to detect any new revisions

ArgoCD, responsible for the CD processes, synchronises the application state in the MicroK8s cluster with the new state defined in the GitLab repository. In short, ArgoCD automates the deployment (CD process) and ensures that the application matches the Git repository. Each TEADAL node hosts its own instance of ArgoCD, unlike GitLab which is common to every TEADAL node. In the end, before the node deployment, HELM packages the application using Charts. Once the application is packaged it gets deployed on the node inside the cluster.

Funded by the European Union

4.1 TEADAL.NODE ON GITLAB

The whole TEADAL Git resources include repositories for development in TEADAL Tech, pilot-related repositories in TEADAL Pilots, and repositories for the Dockerfiles created to integrate the tools needed by the Pilots, in TEADAL Public Images and TEADAL Images. The TEADAL repository is then divided into four Groups (Figure 8).

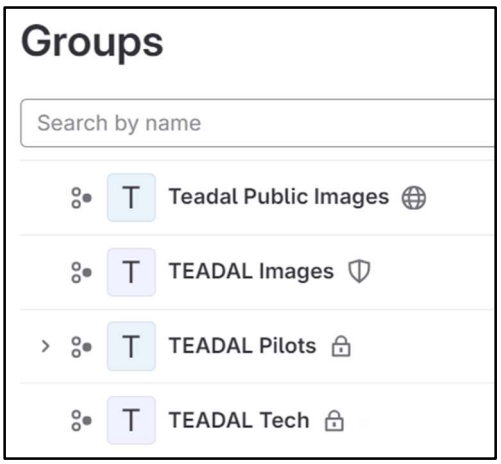

FIGURE 8: TEADAL GIT REPOSITORY

Into the TEADAL Tech group, the Teadal.Node area contains the baseline software (Figure 9) articulated in four subfolders of the *deployment* folder. With respect to the TEADAL Cluster layers described in Section 2.3, Figure 1

- the *mesh-infra* folder hosts the "mesh-infra" layer software (Istio, Keycloack, etc.) depicted in Figure 1;
- the *pilot-services* and *plat-app-services* folders host the "products" layer software (pilot FDPs/sFDPs, pipelines, etc.) depicted in Figure 1;
- the *plat-infra-services* hosts the "core layer" baseline software (PostgreSQL, Kubeflow, etc.) depicted in Figure 1.

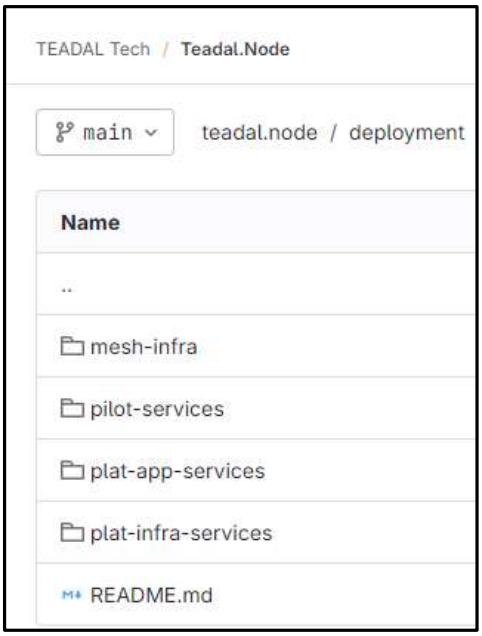

FIGURE 9: TEADAL.NODE GIT REPOSITORY

Funded by the European Union

The TEADAL Pilot groups pilot repositories. Pilot developers can clone and customise the TEADAL Node, and develop their FDPs and sFDPs, having everything separate. This configuration helps with maintaining a better environment for everyone, where the different deployments of the pilots don't influence each other (see Figure 10).

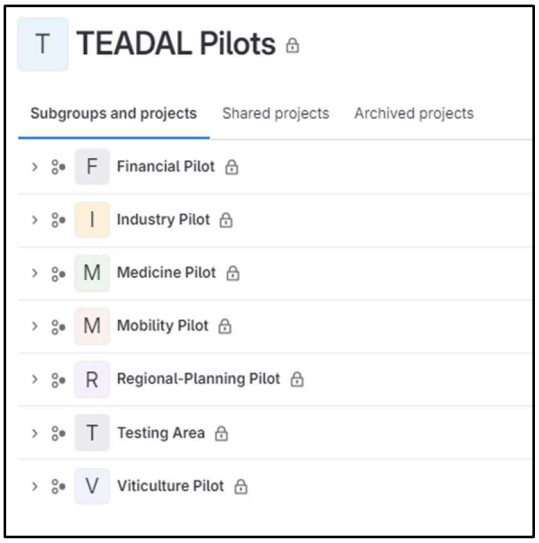

FIGURE 10: TEADAL PILOTS GIT REPOSITORY

UBIWHERE is responsible for maintaining the Git repository and managing who has access to it. Every time a new developer or maintainer joins the TEADAL project, they should ask UBIWHERE to have access to their Git repository, so they are able to customise and update the TEADAL software.

4.2 CLONING AND CUSTOMISING THE TEADAL.NODE

Each pilot developer clones the Teadal.Node repository in a Pilot Node, customises it and deploy it to the established hardware resources. On the node repository there's a readme file that contains the Deployment Guide: this “quick start” document explains every step needed to clone and deploy the TEADAL node.

To clone the TEADAL node all it’s needed is to execute a “git clone” command on the repository:

“git clone https://gitlab.teadal.ubiwhere.com/teadal-pilots/<name of pilot>/<name of pilot>.git”

So, if the developer changes the “name of pilot” to their repository name and executes the command line it will clone the updated TEADAL repository. The TEADAL software and dummy FDPs will be available in a directory named “name of pilot”.

After cloning the TEADAL node, it is necessary to proceed with the installation as described in the Deployment Guide. Any issue with the installation process should be reported to UBIWHERE.

The Deployment Guide is divided into the following parts:

- Setup the environment
- Install MicroK8s

Funded by the European Union

- Setup the Network
- Setup the Mesh
- K8s Storage
- Istio
- ArgoCD connection
- ArgoCD deployment
- K8s Secrets
- Checking the installation (ArgoCD, Istio, Storage, Security, DBs, Dummy FDP/SFDP)

A first deployment of the TEADAL cluster should be done before customising and updating the software; tools like ArgoCD will be useful in those processes.

Now that the TEADAL cluster is ready, the developer can start customising it. The customization can be done with the help of any code tools, like Visual Studio Code, or even through the deployed tools GUI, such as ArgoCD GUI, MinIO GUI, or Keycloak GUI.

4.3 UPDATING THE CUSTOMISED NODE

To update the customised node, developers need to update the Git repository. For that, it's necessary to commit the changes to the origin repository via git commit (CI process). After committing the updates and waiting some minutes for ArgoCD to fetch the changes, the Micro K8s cluster will start receiving the updates and deploying what is new in the repository (CD process).

In short, TEADAL CI/CD was developed so updating the node is the most straightforward and automated process it can be, leaving the hard work of updating everything manually to the CI/CD tools implemented (GitLab, ArgoCD, and Kubernetes).

4.4 DEPLOYING THE CUSTOMISED NODE

If the TEADAL cluster is already running in the machine as explained in subsection 3.3, the process of deploying the customised node is automated. After deploying the first time with the help of the quick start guide, all the updates made to the node will be automatically fetched with ArgoCD. This ArgoCD implementation makes the job of the developer easier, after deploying it once it's not necessary to deploy it anymore.

Funded by the European Union

5 TEADAL CI/CD PIPELINE

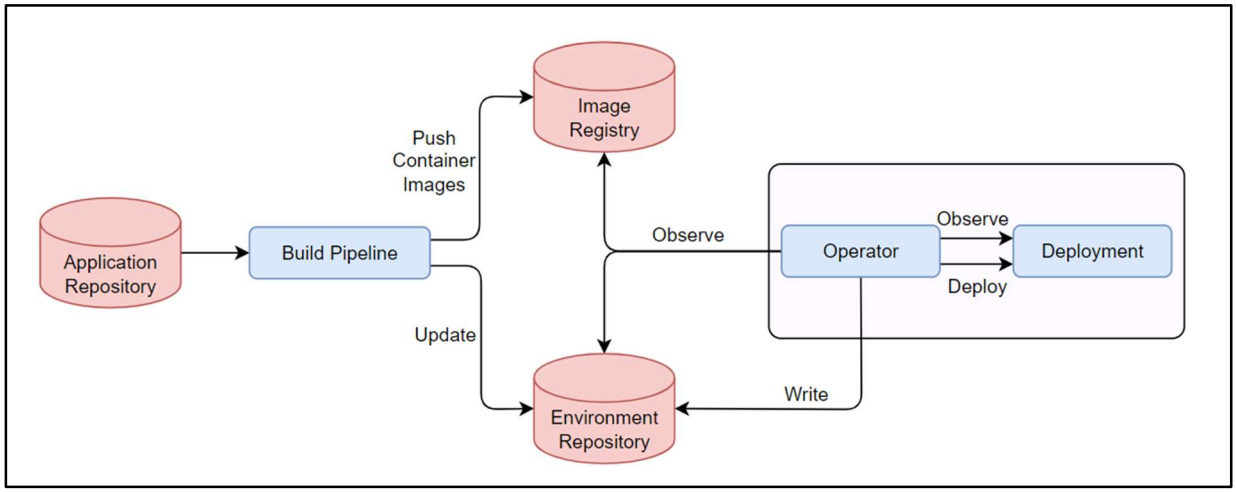

FIGURE 11: PULL-BASED DEPLOYMENT ARCHITECTURE

TEADAL CI/CD pipeline works as a pull-based pipeline (Figure 11), which means that it has an operator working from the inside of the environment taking the role of the deployment pipeline while observing the environment repository. In TEADAL the operator working from the inside of the environment is ArgoCD, and the application repository is based on GitLab.

5.1 CI/CD TOOLS INTEGRATION

CI/CD tools are designed to automate software Development and Testing processes. They are triggered by commits to the codebase and integrate with code repositories, version control systems, and DevOps tools[13].

To build a CI/CD pipeline for Kubernetes it's necessary a CI tool and a CD tool. In TEADAL the technology responsible for the CI pipeline is GitLab and the responsible for the CD pipeline is ArgoCD and Kubernetes (MicroK8s). Kubernetes, ArgoCD, and GitLab work together to achieve the CI/CD pipeline. The diagram of the CI/CD pipeline between these tools is shown in Figure 12.

[13] https://bluelight.co/blog/best-ci-cd-tools

Funded by the European Union

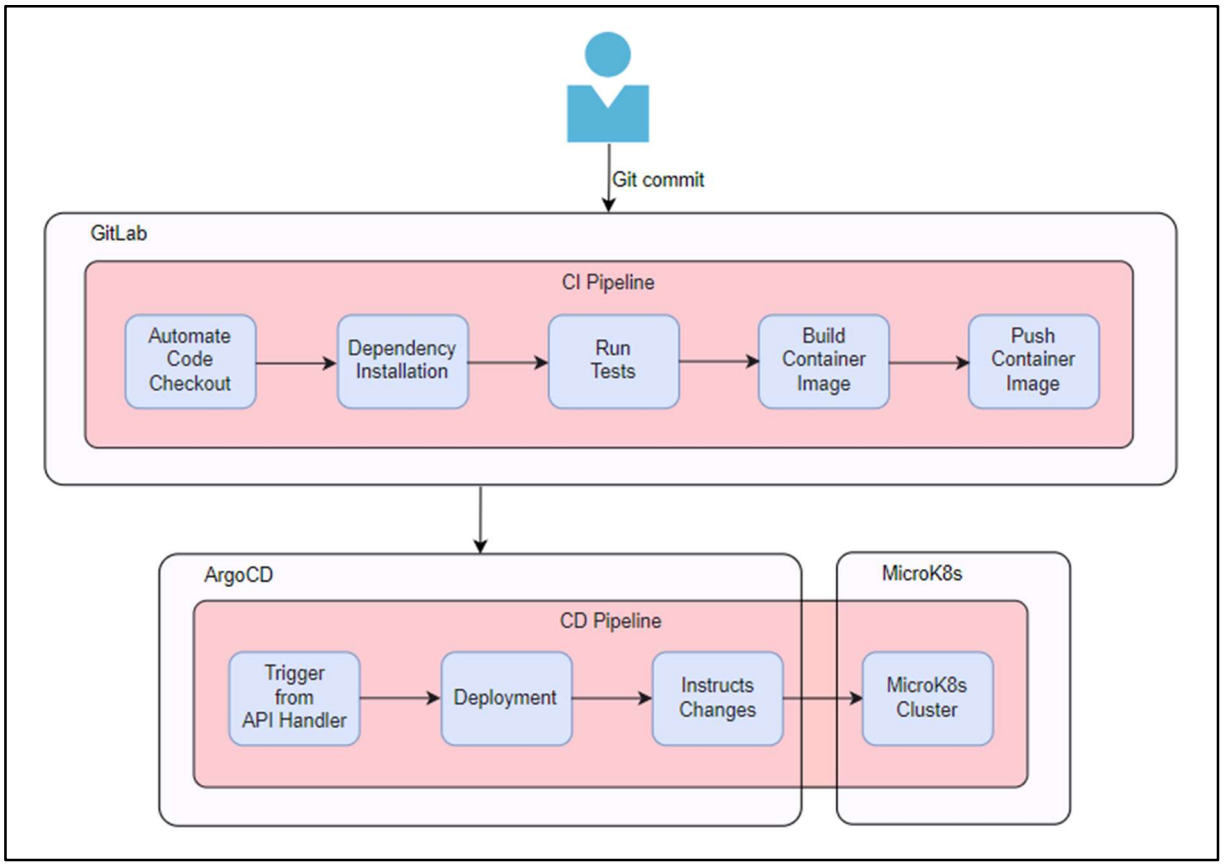

FIGURE 12: CI/CD PIPELINE DIAGRAM

GitLab realises the CI pipeline, which is the first half of the CI/CD pipeline (Figure 13). It mainly involves the integration and testing of the committed code. The steps of the pipeline are:

- **Automate Code Checkout**:

  The CI pipeline begins with code checkout, this is where the latest code stored in GitLab is pulled, ensuring the CI pipeline has the most recent version.

- **Dependency Installation:**

  This step is responsible for the installation of dependencies, like libraries or packages that the software needs to run properly.

- **Run Tests:**

  This step is crucial, it involves executing tests that will check the software for any errors or bugs.

- **Build Container Image:**

  After the tests pass, it's possible to build the container image.

- **Push Image to Registry:**

  The last step of the CI pipeline is pushing the newly created container image to the HELM registry.

Funded by the European Union

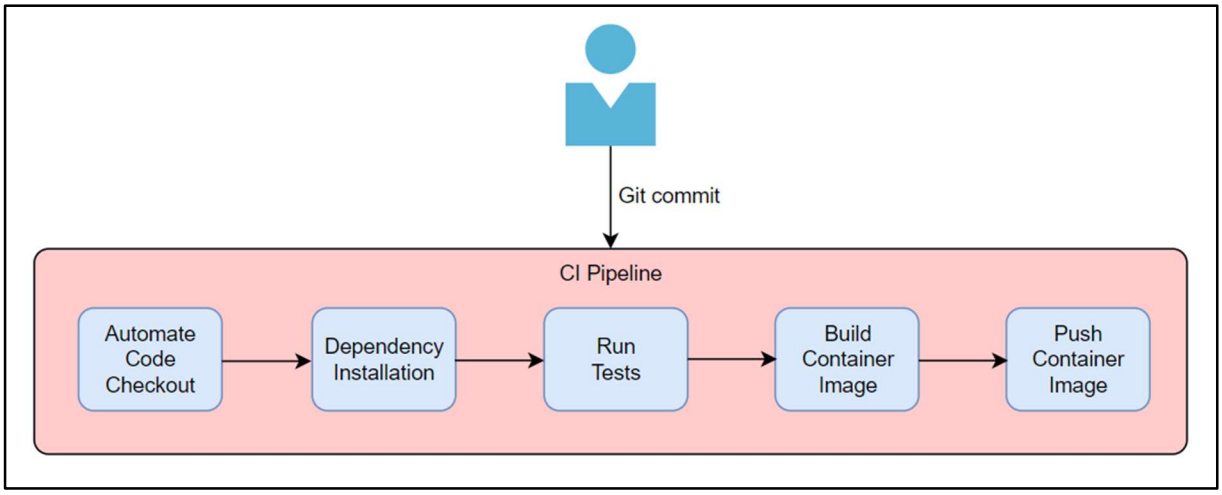

FIGURE 13: CI PIPELINE

After all these steps, made with GitLab, the first half of the CI/CD pipeline is completed. Next, is the CD pipeline that is made with ArgoCD and Kubernetes.

The CD pipeline involves the deployment of the newly created container images to the production environment (Figure 14). It is composed of only two steps which are:

- **Deployment Triggers:**

  These triggers are events that initiate the deployment process, these can be manual or automatic CI pipeline triggers. In TEADAL these triggers are automatically made by the API handler that receives requests from the GitLab CI pipeline.

- **Deployment:**

  The final stage of the CI/CD pipeline, it is the deployment of the initial commit to the production environment. It involves deploying the new version to the established VMs.

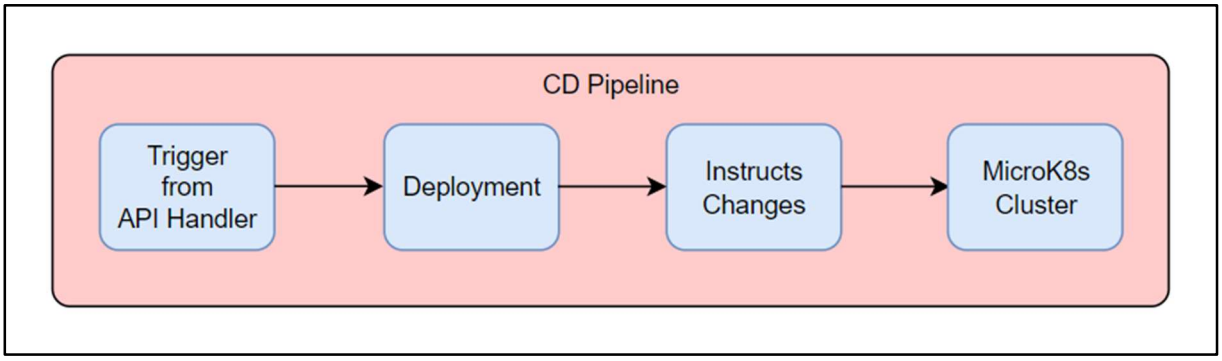

FIGURE 14: CD PIPELINE

After implementing all of the steps above a CI/CD pipeline is achievable. In TEADAL all these steps gave the project a significant enhancement in the development process. Making it more efficient, reliable, and robust.

Funded by the European Union

5.2 CI/CD PIPELINES DESIGN

As explained in Section 3, the CI/CD pipeline leverages three main components: GitLab, ArgoCD, and Kubernetes. The pipeline steps of each Pilot are similar however the different architectures of the Pilots force some changes in the pipelines.

In the following subsections, Pilot pipelines are designed and described[14].

Pilot #1 - Evidence-Based Medicine

Pilot #1, Evidence-Based Medicine, features an architecture with three Virtual Machines (Figure 15), each hosting its own instance of the TEADAL node, a customised node specifically developed for this pilot. Continuous development and monitoring of each TEADAL node are facilitated by ArgoCD, which monitors the GitLab repository linked to each deployed node. A separate repository is used for each VM.

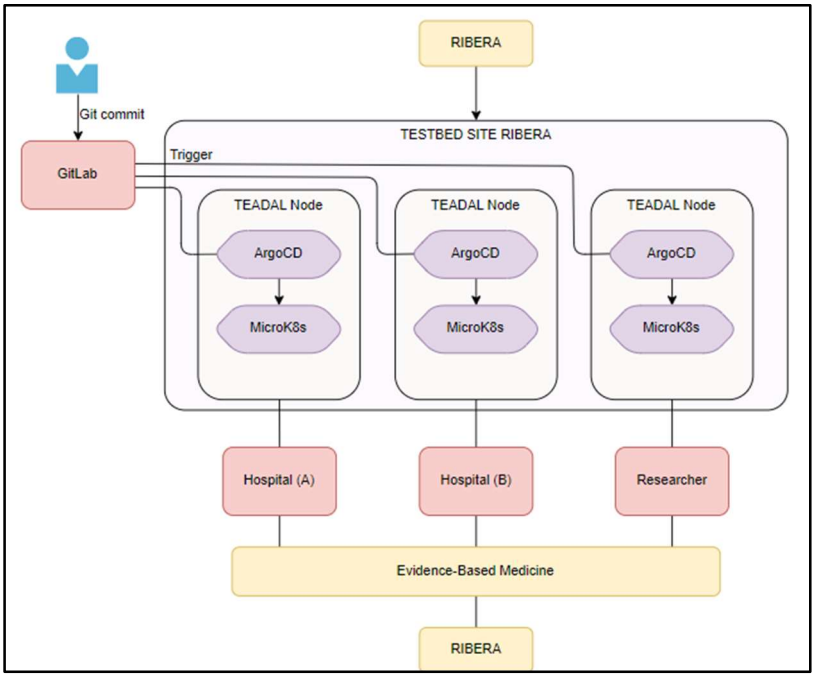

FIGURE 15: PILOT #1 CI/CD PIPELINE DIAGRAM

The CI/CD pipeline steps are consistent across both VMs, ensuring a streamlined process for managing the TEADAL nodes. The general pipeline steps for each VM with the deployed TEADAL node are as follows:

- **Developer Commits Code Updates to GitLab**:
  - Developers make code updates and commit them to the associated GitLab repository.
- **GitLab CI Pipeline Gets Triggered**:
  - The GitLab CI Pipeline is triggered upon detecting new commits (as detailed in Section 3.1).
- **ArgoCD Detects Changes**:
  - ArgoCD monitors the updated GitLab repository and detects any changes.

[14] Please, refer to Deliverable 2.3 “Pilot cases’ final description and intermediate architecture of the platform” for the final pilot use cases’ description

Funded by the European Union

- **ArgoCD Synchronises the Kubernetes Cluster on the VM**:
  - ArgoCD synchronises the Kubernetes cluster on the corresponding VM to apply the updates.
- **Kubernetes Manages Deployment, Scaling, and Operation**:
  - Kubernetes takes charge of deploying, scaling, and operating the containers within the cluster.
- **Kubernetes Monitors the TEADAL Node**:
  - Kubernetes continuously monitors the TEADAL node to ensure it is running in the desired state by overseeing the deployed resources.

Pilot #2 - Mobility Federated Access Point

Pilot #2, Mobility Federated Access Point, features an architecture with two Virtual Machines (Figure 16), each hosting its own instance of the TEADAL node, a customised node specifically developed for this pilot. Continuous development and monitoring of each TEADAL node are facilitated by ArgoCD, which monitors the GitLab repository linked to each deployed node. A separate repository is used for each VM.

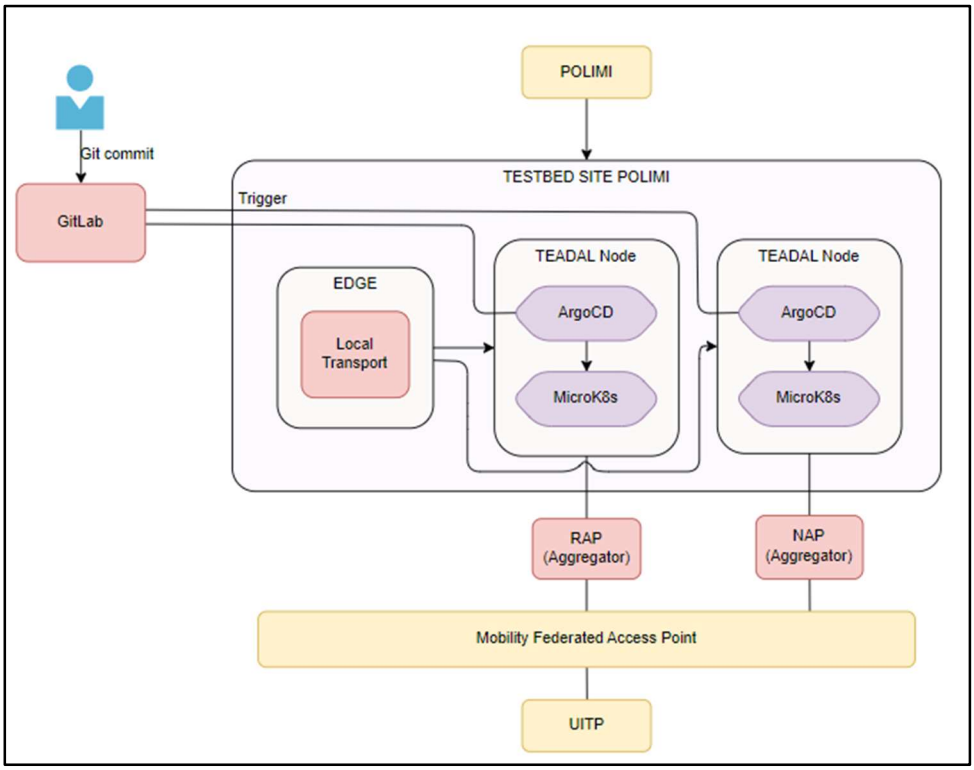

FIGURE 16: PILOT #2 CI/CD PIPELINE DIAGRAM

The CI/CD pipeline steps are consistent across VMs, ensuring a streamlined process for managing the TEADAL nodes. The general pipeline steps for each VM with the deployed TEADAL node are as follows:

- **Developer Commits Code Updates to GitLab**:
  - Developers make code updates and commit them to the associated GitLab repository.
- **GitLab CI Pipeline Gets Triggered**:
  - The GitLab CI Pipeline is triggered upon detecting new commits (as detailed in Section 3.1).
- **ArgoCD Detects Changes**:

Funded by the European Union

- ArgoCD monitors the updated GitLab repository and detects any changes.
- **ArgoCD Synchronises the Kubernetes Cluster on the VM**:
  - ArgoCD synchronises the Kubernetes cluster on the corresponding VM to apply the updates.
- **Kubernetes Manages Deployment, Scaling, and Operation**:
  - Kubernetes takes charge of deploying, scaling, and operating the containers within the cluster.
- **Kubernetes Monitors the TEADAL Node**:
  - Kubernetes continuously monitors the TEADAL node to ensure it is running in the desired state by overseeing the deployed resources.

Pilot #3 - Smart Viticulture Data Sharing

Pilot #3, Smart Viticulture Data Sharing, features an architecture with one Virtual Machine (Figure 17), hosting its own instance of the TEADAL node, a customised node specifically developed for this pilot, and three Edge nodes. Different from the other pilots it also implements three Edge nodes, these Edge nodes have their own instance of the TEADAL node but with fewer technologies and capabilities, adapted to the Edge needs. However, the CI/CD pipeline steps are the same, the only change is the GitLab repository associated with the node. Continuous development and monitoring of each TEADAL node are facilitated by ArgoCD, which monitors the GitLab repository linked to each deployed node. A separate repository is used for each VM and Edge node.

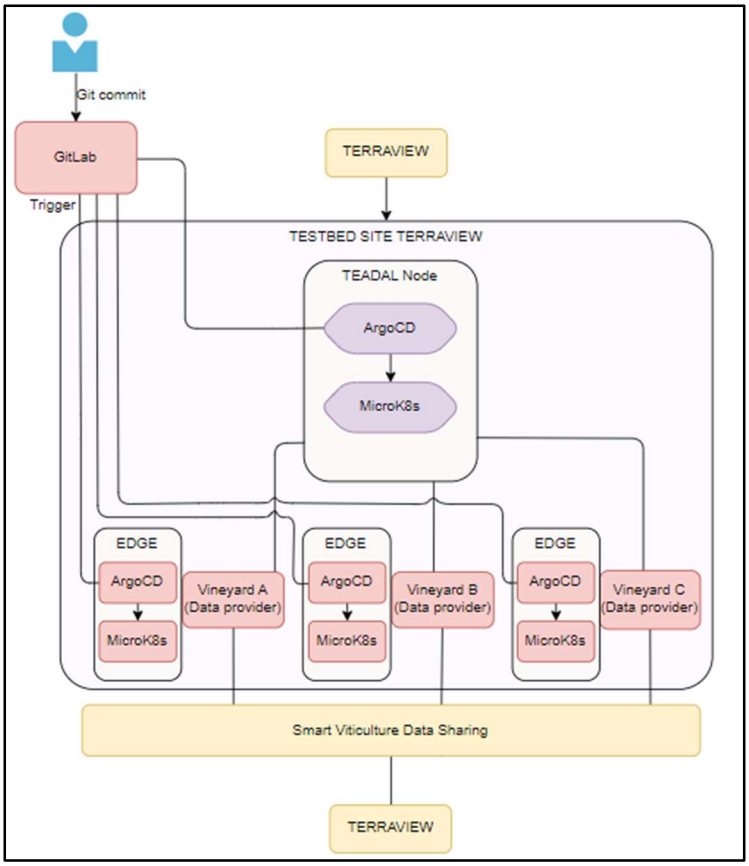

FIGURE 17: PILOT #3 CI/CD PIPELINE DIAGRAM

The CI/CD pipeline steps are consistent across VMs, ensuring a streamlined process for managing the TEADAL nodes. The general pipeline steps for each VM with the deployed TEADAL node are as follows:

Funded by the European Union

- **Developer Commits Code Updates to GitLab**:
  - Developers make code updates and commit them to the associated GitLab repository.
- **GitLab CI Pipeline Gets Triggered**:
  - The GitLab CI Pipeline is triggered upon detecting new commits (as detailed in Section 3.1).
- **ArgoCD Detects Changes**:Can you please check if it's only this?
- https://gitlab.teadal.ubiwhere.com/teadal-tech/integration.guide/-/blob/main/integration.md
  - ArgoCD monitors the updated GitLab repository and detects any changes.
- **ArgoCD Synchronises the Kubernetes Cluster on the VM**:
  - ArgoCD synchronises the Kubernetes cluster on the corresponding VM to apply the updates.
- **Kubernetes Manages Deployment, Scaling, and Operation**:
  - Kubernetes takes charge of deploying, scaling, and operating the containers within the cluster.
- **Kubernetes Monitors the TEADAL Node**:
  - Kubernetes continuously monitors the TEADAL node to ensure it is running in the desired state by overseeing the deployed resources.

Pilot #4 - Industry 4.0 Fast KPI Calculation

Pilot #4, Industry 4.0 Fast KPI Calculation, features an architecture with two Virtual Machines (Figure 18), each hosting its own instance of the TEADAL node, a customised node specifically developed for this pilot. Continuous development and monitoring of each TEADAL node are facilitated by ArgoCD, which monitors the GitLab repository linked to each deployed node. A separate repository is used for each VM.

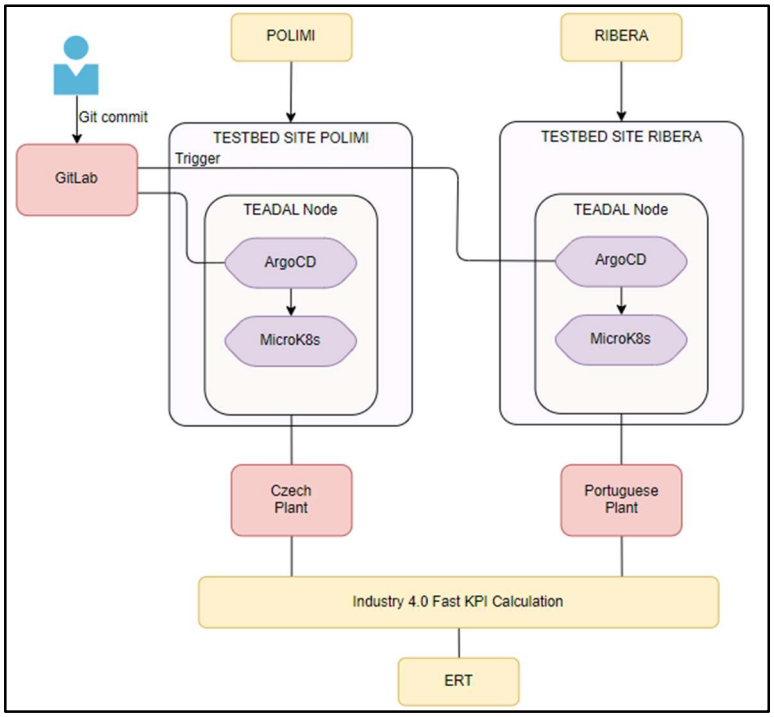

FIGURE 18: PILOT #4 CI/CD PIPELINE DIAGRAM

The CI/CD pipeline steps are consistent across both VMs, ensuring a streamlined process for managing the TEADAL nodes. The general pipeline steps for each VM with the deployed TEADAL node are as follows:

Funded by the European Union

- **Developer Commits Code Updates to GitLab**:
  - Developers make code updates and commit them to the associated GitLab repository.
- **GitLab CI Pipeline Gets Triggered**:
  - The GitLab CI Pipeline is triggered upon detecting new commits (as detailed in Section 3.1).
- **ArgoCD Detects Changes**:
  - ArgoCD monitors the updated GitLab repository and detects any changes.
- **ArgoCD Synchronises the Kubernetes Cluster on the VM**:
  - ArgoCD synchronises the Kubernetes cluster on the corresponding VM to apply the updates.
- **Kubernetes Manages Deployment, Scaling, and Operation**:
  - Kubernetes takes charge of deploying, scaling, and operating the containers within the cluster.
- **Kubernetes Monitors the TEADAL Node**:
  - Kubernetes continuously monitors the TEADAL node to ensure it is running in the desired state by overseeing the deployed resources.

Pilot #6 - Regional Planning for Environmental Sustainability

Pilot #6, Regional Planning for Environmental Sustainability, features an architecture with two Virtual Machines (Figure 19), each hosting its own instance of the TEADAL node, a customised node specifically developed for this pilot. Continuous development and monitoring of each TEADAL node are facilitated by ArgoCD, which monitors the GitLab repository linked to each

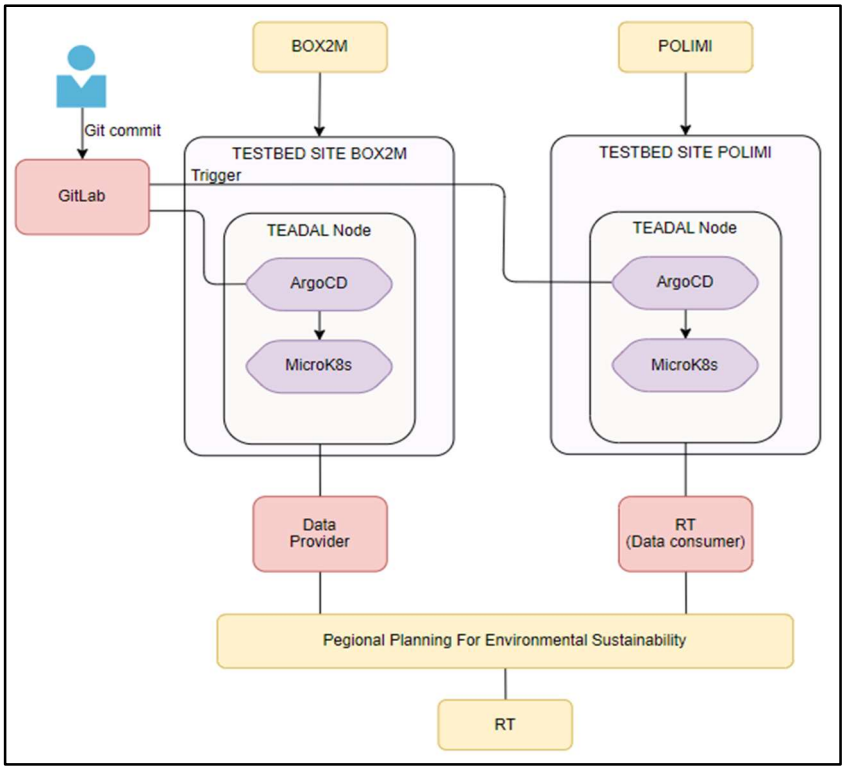

FIGURE 19: PILOT #6 CI/CD PIPELINE DIAGRAM

deployed node. A separate repository is used for each VM.

Funded by the European Union

The CI/CD pipeline steps are consistent across both VMs, ensuring a streamlined process for managing the TEADAL nodes. The general pipeline steps for each VM with the deployed TEADAL node are as follows:

- **Developer Commits Code Updates to GitLab**:
  - Developers make code updates and commit them to the associated GitLab repository.
- **GitLab CI Pipeline Gets Triggered**:
  - The GitLab CI Pipeline is triggered upon detecting new commits (as detailed in Section 3.1).
- **ArgoCD Detects Changes**:
  - ArgoCD monitors the updated GitLab repository and detects any changes.
- **ArgoCD Synchronises the Kubernetes Cluster on the VM**:
  - ArgoCD synchronises the Kubernetes cluster on the corresponding VM to apply the updates.
- **Kubernetes Manages Deployment, Scaling, and Operation**:
  - Kubernetes takes charge of deploying, scaling, and operating the containers within the cluster.
- **Kubernetes Monitors the TEADAL Node**:
  - Kubernetes continuously monitors the TEADAL node to ensure it is running in the desired state by overseeing the deployed resources.

Funded by the European Union

6 PILOT NODES DEPLOYMENT

To assess the integration of TEADAL platform components with pilot testbeds, an initial trial configuration has been realised, deploying a single baseline node per pilot case.

As outlined in the Section 4, an ArgoCD implementation automates most of the job required to pilot partners to deploy their TEADAL nodes.

Each pilot deployed a single node with baseline functionality. Thus, each pilot loaded sample datasets and developed a Federated Data Product in order to validate through the pilot cases the basic functionalities of the platform and ensure all components were working as intended with the underlying supporting testbed.

As outlined in the previous sections, each pilot leverages the TEADAL mesh-infra and core components from the central master repository and enriches its own repository with custom applications, according to the pilot case requirements.

Following an iterative process, pilot testbeds will be continuously enriched with further nodes and components, according to pipelines described in Section 5.

1.1. TRIAL OUTPUTS

The baseline data lake distribution consists in a list of tools (already introduced in Section 2) that are included in the TEADAL node and enable Federated Data Products [15]:

- Airflow: Management platform to define pipelines and workflows on data
- ArgoCD: GitOps IaC tool for Kubernetes clusters
- Keycloak: Access/authentication manager
- Kubeflow: Management platform to define pipelines and workflows on data related to machine learning
- Istio: Service mesh network
- Monitoring tools:
    - Grafana: Platform for data visualisation
    - Prometheus: Tool for collection and storage of computing metrics
    - Kiali: Istio console to monitor and control the service mesh
    - Jaeger: Tracing tool to map data flows and requests
- MinIO: Object storage
- OPA (Open Policy Agent): Security Policy enforcement tool
- PostgreSQL: SQL database

These tools have been packed in a containerized distribution, according to the diagram in Figure 20.

[15] Please, refer also to section 9.5.3 in Deliverable 2.2 “Pilot cases’ intermediate description & initial architecture of the platform V 1.0”

Funded by the European Union

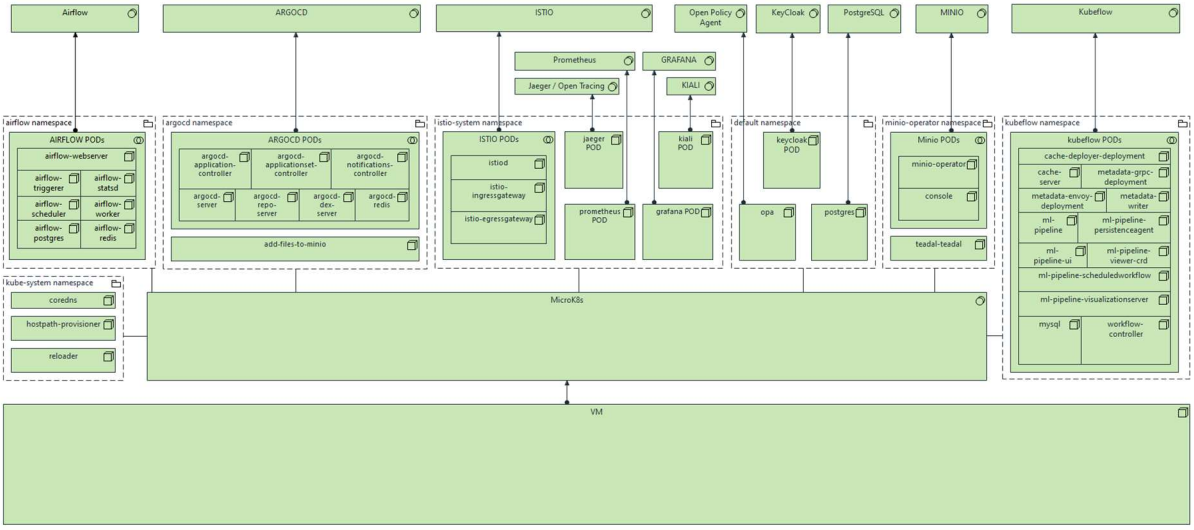

FIGURE 20: BASELINE DATA LAKE DISTRIBUTION

Once completed the node distribution, sample datasets were loaded on the selected storage (MinIO or PostgreSQL) by means of TEADAL tools. Then FDPs were deployed and run, to test the whole platform integration.

Details on the trial pilot testbed configurations are reported in the following subsections.

6.1 PILOT #1 - EVIDENCE-BASED MEDICINE

The Evidence-Based Medicine pilot experimented with the first deployment process with a pipeline starting from the Git repository at https://gitlab.teadal.ubiwhere.com/teadal-pilots/medicine-pilot/evidence-based-medicine-teadal-node-01/. The TEADAL node was deployed on a VM provided by RIBERA on Azure cloud provider, featuring a Hospital data lake.

A first dataset with a few patients data was loaded on MinIO by means of MinIO GUI. The Patients Personal Information dataset description was made available on a Central Data Catalog that has been deployed on a dedicated VM provided “as a service” by CEFRIEL and can be viewed at https://kcong.cefriel.com:8087/assets/Dataset/Ribera%20Person%20Data . Additional datasets descriptions were made available on the Catalog as well.

An FDP was designed to asyncly retrieve data, its OpenAPI specification is available at https://gitlab.teadal.ubiwhere.com/teadal-pilots/medicine-pilot/fdp-medicine/-/blob/main/api/fdp-medicine.yaml

The FDP deployment is still in progress; the FDP description was made available on the Catalog at https://kcong.cefriel.com:8087/assets/Federated%20data%20product/Ribera%20Salud%20fdp-medicine

Table 1 reports the trial configuration as described above.

Funded by the European Union

Pilot #1	Evidence-based medicine
Node location	RIBERA VM on Microsoft Azure
TEADAL.Node clone	https://gitlab.teadal.ubiwhere.com/teadal-pilots/medicine-pilot/evidence-based-medicine-teadal-node/
Organisation	Hospital
Dataset(s) loaded	Patients Personal Information and clinical data.
Datasets description on Data Catalog	Drug Exposure Data https://kcong.cefriel.com:8087/assets/Dataset/Ribera%20Drug%20Exposure%20Data Procedure Occurrence Data https://kcong.cefriel.com:8087/assets/Dataset/Ribera%20Procedure%20Occurrence%20Data Person Data https://kcong.cefriel.com:8087/assets/Dataset/Ribera%20Person%20Data Observation Data https://kcong.cefriel.com:8087/assets/Dataset/Ribera%20Observation%20Data Measurement Data https://kcong.cefriel.com:8087/assets/Dataset/Ribera%20Patient%20Measurement%20Data
FDP REST API definition	https://gitlab.teadal.ubiwhere.com/teadal-pilots/medicine-pilot/fdp-medicine/-/blob/main/api/fdp-medicine.yaml
FDP source code on Gitlab	https://gitlab.teadal.ubiwhere.com/teadal-pilots/medicine-pilot/fdp-medicine/-/tree/main (In Progress)
FDP URL	In Progress
FDP Description on Data Catalog	https://kcong.cefriel.com:8087/assets/Federated%20data%20product/Ribera%20Salud%20fdp-medicine

TABLE 1: EVIDENCE-BASED MEDICINE TRIAL CONFIGURATION

6.2 PILOT #2 - MOBILITY FEDERATED ACCESS POINT

The Mobility Federated Access Point pilot experimented with the first deployment process with a pipeline starting from the Git repository at https://gitlab.teadal.ubiwhere.com/teadal-pilots/mobility-pilot. The TEADAL node was deployed on a VM provided by POLIMI, featuring the AMT Transport Operator data lake.

A first dataset with a few static data was loaded on MinIO by means of MinIO GUI. The GTFS Open Data From AMTS Catania dataset description was made available on the Central Data Catalog and can be viewed at https://kcong.cefriel.com:8087/assets/Dataset/GTFS%20Open%20data%20from%20AMTS%20Catania. Additional datasets descriptions were made available on the Catalog as well.

An FDP was designed to retrieve bus stops list; its OpenAPI specification is available at https://gitlab.teadal.ubiwhere.com/teadal-pilots/mobility-pilot/fdp-amts-gtfs-static/-/blob/main/api/fdp-amts-gtfs-static.yaml.

The FDP was deployed and tested using the endpoint at http://mobility.teadal.ubiwhere.com/fdp-amts-gtfs-static/stops, retrieving stops data and demonstrating that the AMT Transport Operator TEADAL node was working as expected.

Finally, the FDP description was made available on the Catalog at https://kcong.cefriel.com:8087/dashboard/assets/Federated%20data%20product/AMTS%20Integrated%20Mobility%20FDP.

Table 2 reports the trial configuration as described above.

Funded by the European Union

Pilot #2	Mobility Federated Access Point
Node location	POLIMI VM
TEADAL.Node clone	https://gitlab.teadal.ubiwhere.com/teadal-pilots/mobility-pilot
Organisation	Transport Operator: AMTS
Dataset(s) loaded	Transport static data in GTFS format
Datasets description on Data Catalog	GTFS Open Data From AMTS Catania https://kcong.cefriel.com:8087/assets/Dataset/GTFS%20Open%20data%20from%20AMTS%20Catania AMTS Real Time Trips Data https://kcong.cefriel.com:8087/assets/Dataset/AMTS%20real%20time%20trips%20data AMTS Real Time Bus Position Data https://kcong.cefriel.com:8087/assets/Dataset/AMTS%20real%20time%20bus%20position%20data
FDP REST API definition	https://gitlab.teadal.ubiwhere.com/teadal-pilots/mobility-pilot/fdp-amts-gtfs-static/-/blob/main/api/fdp-amts-gtfs-static.yaml
FDP source code on Gitlab	NodeJS implementation https://gitlab.teadal.ubiwhere.com/teadal-pilots/mobility-pilot/fdp-amts-gtfs-static
FDP URL	NodeJS implementation https://gitlab.teadal.ubiwhere.com/teadal-pilots/mobility-pilot/fdp-amts-gtfs-static
FDP Description on Data Catalog	http://mobility.teadal.ubiwhere.com/fdp-amts-gtfs-static/stops

TABLE 2: MOBILITY FEDERATED ACCESS POINT TRIAL CONFIGURATION

6.3 PILOT #3 - SMART VITICULTURE DATA SHARING

The Smart Viticulture Data Sharing pilot experimented with the first deployment process with a pipeline starting from the Git repository at https://gitlab.teadal.ubiwhere.com/teadal-pilots/viticulture-pilot/smart-viticulture-teadal-node. The TEADAL node was deployed on a VM provided by TERRAVIEW, featuring the Terraview Core data lake.

A first dataset was loaded on MinIO by means of MinIO Operator UI and MiniIO API. The Aquaview dataset description was made available on the Central Data Catalog and can be viewed at https://kcong.cefriel.com:8087/assets/Dataset/Aquaview%20Dataset.

An FDP was designed to retrieve soil moisture data; its OpenAPI specification is available at https://gitlab.teadal.ubiwhere.com/teadal-pilots/viticulture-pilot/fdp-viticulture-moistures/-/blob/main/smartviticulture_fdp.yaml.

The FDP, implemented in Python, was deployed and tested using the endpoint at http://20.4.3.245/fdp-viticulture-moistures//measurement/soilmoisture/aois, retrieving soil moisture data and demonstrating that the Terraview Core TEADAL node was working as expected.

Finally, the FDP description was made available on the Catalog at https://kcong.cefriel.com:8087/dashboard/assets/Federated%20data%20product/Aquaview%20FDP.

Table 3 reports the trial configuration as described above.

Funded by the European Union

Pilot #3	Smart Viticulture Data Sharing
Node location	TERRAVIEW VM
TEADAL.Node clone	https://gitlab.teadal.ubiwhere.com/teadal-pilots/viticulture-pilot/smart-viticulture-teadal-node
Organisation	Terraview Core
Dataset(s) loaded	Soil moisture data
Datasets description on Data Catalog	https://kcong.cefriel.com:8087/assets/Dataset/Aquaview%20Dataset
FDP REST API definition	https://gitlab.teadal.ubiwhere.com/teadal-pilots/viticulture-pilot/fdp-viticulture-moistures/-/blob/main/smartviticulture_fdp.yaml
FDP source code on Gitlab	https://gitlab.teadal.ubiwhere.com/teadal-pilots/viticulture-pilot/fdp-viticulture-moistures/-/blob/main/teadal.py
FDP URL	http://20.4.3.245/fdp-viticulture-moistures//measurement/soilmoisture/aois
FDP Description on Data Catalog	https://kcong.cefriel.com:8087/dashboard/assets/Federated%20data%20product/Aquaview%20FDP

TABLE 3: SMART VITICULTURE DATA SHARING TRIAL CONFIGURATION

6.4 PILOT #4 - INDUSTRY 4.0 FAST KPI CALCULATION

The Industry 4.0 fast KPI calculation pilot experimented with the first deployment process with a pipeline starting from the Git repository at https://gitlab.teadal.ubiwhere.com/teadal-pilots/industry-pilot/industry-teadal-node. The TEADAL node was deployed on a VM provided by POLIMI, featuring the ERT plant data lake.

A first dataset was loaded on MinIO by means of MinIO GUI. The ERT Plant dataset description was made available on the Central Data Catalog and can be viewed at https://kcong.cefriel.com:8087/assets/Dataset/ERT%20Czech%20Plant%20data.

An FDP was designed to retrieve plant data; its OpenAPI specification is available at https://gitlab.teadal.ubiwhere.com/teadal-pilots/industry-pilot/fdp-czech-plant/-/blob/main/api/fdp-czech-plant.yaml.

The FDP, implemented with NodeJS, was deployed and tested using the endpoint at http://industry.teadal.ubiwhere.com/fdp-czech-plant/shipments.

http://industry.teadal.ubiwhere.com/shipments/customer/501220267, retrieving plant data and demonstrating that the ERT Plant TEADAL node was working as expected.

Finally, the FDP description was made available on the Catalog at https://kcong.cefriel.com:8087/dashboard/assets/Federated%20data%20product/Czech%20plant%20data%20access%20FDP.

Table 4 reports the trial configuration as described above.

Pilot #4	Industry 4.0 fast KPI calculation
Node location	POLIMI VM
TEADAL.Node clone	https://gitlab.teadal.ubiwhere.com/teadal-pilots/industry-pilot/industry-teadal-node
Organisation	ERT Plant
Dataset(s) loaded	Plant related data in CSV format
Datasets description on Data Catalog	https://kcong.cefriel.com:8087/assets/Dataset/ERT%20Czech%20Plant%20data
FDP REST API definition	https://gitlab.teadal.ubiwhere.com/teadal-pilots/industry-pilot/fdp-czech-plant/-/blob/main/api/fdp-czech-plant.yaml

Funded by the European Union

FDP source code on Gitlab	https://gitlab.teadal.ubiwhere.com/teadal-pilots/industry-pilot/fdp-czech-plant/-/tree/main/src/services
FDP URL	http://industry.teadal.ubiwhere.com/fdp-czech-plant/shipments http://industry.teadal.ubiwhere.com/shipments/customer/501220267
FDP Description on Data Catalog	https://kcong.cefriel.com:8087/dashboard/assets/Federated%20data%20product/Czech%20plant%20data%20access%20FDP

TABLE 4: INDUSTRY 4.0 FAST KPI CALCULATION TRIAL CONFIGURATION

6.5 PILOT #6 - REGIONAL PLANNING FOR ENVIRONMENTAL SUSTAINABILITY

The Regional planning for environmental sustainability pilot experimented with the first deployment process with a pipeline starting from the Git repository at https://gitlab.teadal.ubiwhere.com/teadal-pilots/regional-planning-pilot. The TEADAL node was deployed on a VM provided by BOX2M, featuring the BOX2M data lake.

No datasets have been loaded into the BOX2M data lake, as Energy Consumption and Building Occupancy Data are retrieved from a BOX2M service in real-time. The BOX2M dataset description was made available on the Central Data Catalog and can be viewed at https://kcong.cefriel.com:8087/assets/Dataset/BOX2M%20energy%20consumption%20and%20building%20occupancy%20data. The data catalog registers datasets belonging to Regione Toscana, Servizio Idrologico Regione Toscana and ARPAT Regione Toscana as well.

An FDP was designed to retrieve BOX2M data; its OpenAPI specification is available at https://gitlab.teadal.ubiwhere.com/teadal-pilots/regional-planning-pilot/box2m-fdp/-/blob/main/draft-fdp-box2m.yaml.

The FDP was deployed and tested using the endpoint at http://teadal.box2m.io, retrieving plant data and demonstrating that the BOX2M TEADAL node was working as expected.

Finally, the FDP description was made available on the Catalog at https://kcong.cefriel.com:8087/dashboard/assets/Federated%20data%20product/BOX2M-FDP.

Table 5 reports the trial configuration as described above.

Pilot #6	Regional planning for environmental sustainability
Node location	BOX2M VM
TEADAL.Node clone	https://gitlab.teadal.ubiwhere.com/teadal-pilots/regional-planning-pilot
Organisation	BOX2M
Dataset(s) loaded	BOX2M Energy Consumption and Building Occupancy Data
Datasets description on Data Catalog	https://kcong.cefriel.com:8087/assets/Dataset/BOX2M%20energy%20consumption%20and%20building%20occupancy%20data
FDP REST API definition	https://gitlab.teadal.ubiwhere.com/teadal-pilots/regional-planning-pilot/box2m-fdp/-/blob/main/draft-fdp-box2m.yaml
FDP source code on Gitlab	https://gitlab.teadal.ubiwhere.com/teadal-pilots/regional-planning-pilot/box2m-fdp
FDP URL	http://teadal.box2m.io
FDP Description on Data Catalog	https://kcong.cefriel.com:8087/assets/Federated%20data%20product/BOX2M-FDP

TABLE 5: REGIONAL PLANNING FOR ENVIRONMENTAL SUSTAINABILITY TRIAL CONFIGURATION

Funded by the European Union

7 CONCLUSIONS

This deliverable has presented the implemented integration process to allow the different pilots to integrate in their corresponding testbeds. The integration and deployment tools have been described together with the CI/CD process.

The big effort spent in the implementation of the integration process has resulted in an automatization of the deployment process that can be performed in a matter of minutes, producing a great improvement over the actual manual deployment operated by our pilot partners at the moment.

Furthermore, the first pilot results following the TEADAL Node baseline deployment have been presented.

Funded by the European Union