



EMERALDS

Project Title	Extreme-scale Urban Mobility Data Analytics as a Service
Project Acronym	EMERALDS
Grant Agreement No.	101093051
Start Date of Project	2023-01-01
Duration of Project	36 months
Project Website	https://emeralds-horizon.eu/

D2.2 – Containerized EMERALDS Toolset v1

Work Package	WP 2, Reference Architecture and Toolset Integration
Lead Author (Org)	George Tsakiris (KONNECTA, KNT)
Contributing Author(s) (Org)	Antonis Mygiakis (KONNECTA, KNT), Foivos Galatoulas (INLECOM), Argyrios Kyrgiazos (CARTO), Ignacio EliceGUI (ATOS)
Due Date	30.06.2024
Date	28.06.2024
Version	V1.0

Dissemination Level

<input checked="" type="checkbox"/>	PU: Public
<input type="checkbox"/>	SEN: Sensitive, only for members of the consortium (including the Commission)

Versioning and contribution history

Version	Date	Author	Notes &/or Reason
0.1	06/02/2024	George Tsakiris & Antonis Mygiakis (KONNECTA, KNT)	TOC and V0.1
0.2	20/03/2024	George Tsakiris (KONNECTA, KNT)	Chapter 3 – CI/CD
0.3	19/04/2024	George Tsakiris (KONNECTA, KNT)	Chapter 2 - Containerization
0.4	21/05/2024	George Tsakiris & Antonis Mygiakis (KONNECTA, KNT), Foivos Galatoulas (INLECOM), Argyrios Kyrgiazos (CARTO), Ignacio EliceGUI (ATOS)	Chapter 4, 5 & 6
0.5	18/06/2024	George Tsakiris (KONNECTA, KNT)	Address Review Comments





Quality Control (includes peer & quality reviewing)

Version	Date	Name (Organisation)	Role & Scope
0.4	02/06/2024	Argyris Kyriazos (CARTO), Georgios Theodoropoulos (UPRC), Foivos Galatoulas (INLE)	First Round Internal Review Comments
0.5	12/06/2024	Argyris Kyriazos (CARTO), Georgios Theodoropoulos (UPRC), Foivos Galatoulas (INLE)	Second Round Internal Review Comments
0.6	20/06/2024	Yannis Theodoridis (UPRC)	Scientific and Technical Manager Review
1.0	29/06/2024	Foivos Galatoulas (INLE)	Final review by Coordinator



**Funded by
the European Union**

This project has received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101093051

Disclaimer

EMERALDS - This project has received funding from the Horizon Europe R&I programme under the GA No. 101093051. The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided "as is" without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/ her sole risk and liability.

Copyright message

©EMERALDS Consortium. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation, or both. Reproduction is authorized provided the source is acknowledged.



Table of Contents

- 1 Introduction 9
 - 1.1 Purpose and scope of the document 9
 - 1.2 Relation to Work Packages, Deliverables and Activities 9
 - 1.3 Contribution to WP2 and Project Objectives 10
 - 1.4 Structure of the document 11
- 2 EMERALDS Toolset 12
 - 2.1 Integration Plan 14
 - 2.1.1 Service Architecture and Communication 15
- 3 Containerization of EMERALDS Services 17
 - 3.1 Container engines 18
 - 3.2 Development Process 19
 - 3.2.1 Dockerfile walkthrough 19
- 4 Continuous Integration/Continuous Deployment 21
 - 4.1.1 State of the Art in CI/CD 21
 - 4.1.2 CI/CD Platforms 22
 - 4.1.3 GitHub Actions 23
 - 4.2 Continuous Integration 23
 - 4.2.1 GitHub Actions Management 26
 - 4.2.2 Container Registry 29
 - 4.3 Continuous Deployment 31
 - 4.3.1 CD Platform 31
 - 4.3.2 Platform Deployment and Configuration 36
 - 4.3.3 Deployment of Emerald Services on the CD Platform 39
- 5 Integration with Analytics as a Service Platforms 41
 - 5.1 ATOS Integration 41
 - 5.2 CARTO Integration 42
- 6 Conclusions and next steps. 44

List of Figures

- FIGURE 1-1 THE INTERTWINED DATAOPS, MLOPS AND DEVOPS EMERALDS METHODOLOGY 10
- FIGURE 2-1 THE EMERALDS REFERENCE ARCHITECTURE 12
- FIGURE 2-2-EMERALD SERVICE INTEGRATION ARCHITECTURE 15
- FIGURE 3-1 AN ECOSYSTEM FOR MANAGING DATA PIPELINES ON THE COMPUTING CONTINUUM ¹ 17
- FIGURE 3-2 - DOCKERFILE WALKTHROUGH EXAMPLE 20
- FIGURE 4-1 - CI/CD PROCESS 21
- FIGURE 4-2 - CI PATH 24
- FIGURE 4-3 - GITHUB ACTIONS MANAGEMENT PORTAL 26
- FIGURE 4-4 - GITHUB SBOM PRESENTED 27
- FIGURE 4-5 - DOCKER SCOUT VULNERABILITIES ANALYSIS. 28
- FIGURE 4-6 - GITHUB ACTIONS LOGGING 29
- FIGURE 4-7 - GITHUB PACKAGES 30
- FIGURE 4-8 - MANAGE ACCESS LEVEL 31
- FIGURE 4-9 - KUBEEDGE INSTALLATION SCHEMA 35
- FIGURE 4-10 - CD PLATFORM RESOURCE DIAGRAM 37
- FIGURE 4-11 - EMERALDS HOSTING SERVICE INSTALLATION FLOW 38
- FIGURE 4-12 - ARGO CD DEPLOYMENT FLOW 40



List of Tables

TABLE 1-1 - TERMINOLOGY	5
TABLE 1-2 - MATRIX OF ALIGNMENT	7
TABLE 2-1 - EMERALD TOOLSET REPOSITORIES FOR CONTAINERIZATION AND CI/CD	12
TABLE 2-2 - EMERALDS-HOSTING-SERVICES	14
TABLE 4-1 - GITHUB PACKAGE MANAGER PERMISSION SCHEMA	30
TABLE 4-2 - KUBERNETES VARIANTS	36

Terminology

Terminology/Acronym	Description
AI	Artificial Intelligence
AlaaS	AI as a Service
API	Application Programming Interface
CC	Compute Continuum
CI/CD	Continuous Integration/Continuous Delivery
CNCF	Cloud Native Computing Foundation
CPU	Central Processing Unit
DoA	Description of Action
EC	European Commission
GPU	Graphics Processing Unit
IaC	Infrastructure As Code
IoT	Internet of Things
LXC	Linux Containers
MAaaS	Mobility Analytics as a Service
ML	Machine Learning
NAT	Network Address Translation
OCI	Open Container Initiative
QA	Quality Assurance
REST	Representational State Transfer
RIA	Research and Innovation action
SBOM	Software Billing of Materials
SotA	State of the Art
TRL	Technology Readiness Level
UX	User Experience
VPN	Virtual Private Network
YAML	YAML - YAML Ain't Markup Language

Table 1-1 - Terminology

GA Matrix of alignment

GA Components Title (and type)	GA Component Outline	Document Chapter(s)	Justification
Deliverable			
Deliverable D2.2 – Containerized EMERALDS Toolset v1	Early integration of the developments from WP3 and WP4 into a re-usable and containerized toolset to be demonstrated in applications foreseen in WP5 and WP6 adhering to continuous integration and deployment software development principles. Introducing an efficient, interoperable and easy-to-deploy urban MAaaS toolset, containing methods for executing extreme data workflows.	Chapters 2,3,4,5	<p>Chapter 2 introduces the EMERALDS Toolset concept, a re-usable library of software components and the main points of integration on existing or new data pipelines.</p> <p>Chapter 3 provides the rationale and the implementation details regarding the containerization of the Emeralds Services to facilitate the creation of the Emeralds Toolset.</p> <p>Chapter 4 presents a platform solution based on the demo environment that can be used in production as well and acts as the infrastructure for the MAaaS toolset.</p> <p>Chapter 5 provides a description of the integration of the EMERALDS toolset with two commercial MAaaS platforms.</p>
Tasks			
Task 2.1 Reference Architecture and Containerization of Services (M1-M33)	KNT will setup a Continuous Integration/Continuous Deployment (CI/CD) stack to support the entire software lifecycle processes, from testing of workflows up to the release of the fully tested and deployed solutions in the use cases and proof-of-concept environments.	Chapter 4	Chapter 4 presents the CI/CD pipeline that has been developed to accommodate the creation of the Emeralds Toolset based on the individual Emeralds services. It provides details about the static code analysis, creation of the containerized images, the storage of these image and the deployment of them into the testing and production environments. All steps have been designed to be executed with minimum human intervention.
	A solid continuous integration plan will be developed, that will analyse all software resources (e.g., mechanisms, modules, components, services) available, and identify, specify, and document the integration points amongst these resources. Both inter-module and inter-component integration will be included.	Chapter 2, 4, 5	<p>Chapter 2 outlines the available integration plan for the Emerald Toolset. It details the strategy for formulating effective data pipelines using the Emeralds Services provided in D3.1 and D4.1 and explains how external services or platforms can leverage these integration points to enhance existing or new use cases.</p> <p>Chapter 4 introduces the Emeralds Hosting Service as a derivative of the continuous</p>

			<p>development platform, capable of hosting and exposing the Emeralds Service functionalities to interested parties.</p> <p>Chapter 5 provides an example of the integration methods with two external commercial platforms, ATOS and CARTO, with the Emeralds Toolset.</p>
	An early containerization of tools from D3.1, D4.1 will be performed on M18 (D2.2)	Chapter 3, 4	<p>Chapters 3 and 4 describe the methods for containerizing the Emerald Services developed in D3.1 and D4.1. These chapters also explain how to automate the entire process to deliver the Emeralds Toolset to end users more quickly and with higher quality.</p>
	The containerised Toolset will be interoperable with two analytics as a service platforms (one operated by ATOS and the other established commercial cloud platform run by CARTO), whilst users will be able to call-out selected methods matching the needs of their extreme data workflow task in the form of micro-services (EMERALDS).	Chapter 5	<p>Chapter 5 outlines the primary interaction points between the Emeralds Services and the ATOS and CARTO platforms. The ATOS platform is presented as a Mobility AlaaS (Artificial Intelligence as a Service) platform that offers the development infrastructure for the training and inference of the Emeralds. Integration with CARTO involves various methods for ingesting the output from the Emeralds Service into CARTO’s analytics toolbox.</p>

Table 1-2 - Matrix of Alignment

Executive Summary

EMERALDS's vision is to design, develop and create an **urban data-oriented Mobility Analytics as a Service (MAaaS) toolset**, consisting of the proclaimed '*emeralds*' services, compiled in a proof-of-concept prototype, capable of exploiting the untapped potential of extreme urban mobility data. The toolset enables stakeholders of the urban mobility ecosystem to collect and manage ubiquitous spatio-temporal data of high-volume, high-velocity and of high-variety, analyse them both in online and offline settings, import them to real-time responsive AI/ML algorithms and visualise results in interactive dashboards, whilst implementing privacy preservation techniques at all data modalities and at all levels of a mobility data analytics workflow architecture. The toolset offers advanced capabilities in data mining (searching and processing) of large amounts and varieties of urban mobility data.

In the process of developing the EMERALDS toolset, a thorough analysis was conducted by reviewing the needs captured in D5.1 – Use Cases Scoping Document and the outputs of the core research streams of WP3 and WP4, as reported in D3.1 Mobility Data Processing Services v1 and D4.1 Mobility Data Analytics and Learning Services v1 as monitored and recorded through T1.2 Scientific & Technical Management. The combined study of these project research streams, the technical specification as monitored and recorded through T1.2 Scientific & Technical Management and Toolset architecture defined in D2.1 EMERALDS Reference Architecture, state D2.2 as a cornerstone in the trajectory towards the achievement of Milestone 2, M18: **Primary Implementation Cycle and Initial Integration**. The software released aims to support the diverse range of functional and non-functional end-user requirements that have been collected in D2.1 as well as within the frame of WP5. In addition, this analysis served as a foundation for the design of each 'emerald' service adhering to the overall reference architecture and design specifications required for the release of the first version of the EMERALDS toolset.

This deliverable (of type **OTHER**) presents a comprehensive version of 'emeralds' services, organized into different groups based on a taxonomy serving the project's research goals on the fields of extreme scale data mining, filtering, aggregation and analytics, and releases an integrated version of developments from WP3 and WP4 into a reusable and containerized toolset. The following emeralds typologies have been identified: 1) Privacy-aware in situ Data Harvesting, 2) Data Fusion and Management, 3) Extreme-Scale Cloud and Fog Data Processing, 4) Extreme Scale Mobility Data Analytics at Computer Continuum, 5) Extreme Scale Mobility Data Analytics at Compute Continuum and 6) Security and Data Governance. The containerized services constituting the EMERALDS toolset will be demonstrated in applications foreseen within WP5 and exploited through the planned activities of WP6. Continuous integration and deployment software development principles facilitate the utilization of an efficient, interoperable and easy-to-deploy MAaaS toolset, including the necessary tooling for substantiating extreme data workflows. The implementation of these principles is demonstrated and explained through detailed code walkthroughs, and a full list of links to all Github repositories is provided.

1 Introduction

1.1 Purpose and scope of the document

D2.2 presents an early integration of the developments from WP3 and WP4 into a re-usable and containerized toolset to be demonstrated in applications foreseen in WP5 use cases and exploited via the commercialization strategies devised under WP6. The offering presented in D2.2 has been designed to adhere to continuous integration and deployment software development principles.

Catering for the deployment of services across the computing continuum, from edge to cloud, the toolset ensures scalability and real-time responsiveness, reducing bandwidth requirements and optimizing resource usage. Interoperability is a key focus, with the toolset designed to seamlessly integrate with existing urban mobility systems and data sources. This is also achieved through adherence to open standards and the development of robust APIs that facilitate easy integration and data exchange. The EMERALDS toolset also prioritizes ease of deployment, providing containerized versions of its services that can be quickly and efficiently deployed across various platforms.

Combining the above elements positions the EMERALDS toolset as a transformative solution for urban mobility management, capable of executing extreme data workflows and delivering advanced analytics and insights to support smarter, more sustainable urban transportation systems.

Moreover, D2.2 EMERALDS containerized toolset v1 is of type OTHER, therefore it is accompanied by a report detailing the concept, design, development, testing and release of the integrated EMERALDS toolset along with the relative links to the source code repositories on GitHub.

1.2 Relation to Work Packages, Deliverables and Activities

D2.2 serves as a direct continuation of the activities described in D2.1 and implements methods to facilitate the deployment, versioning and service delivery of emeralds services, and reports the progress of the work carried out under T2.1 directly feeding into the implementation process required for the attainment of a combined emeralds services solution under one unique offering, namely the EMERALDS toolset.

In this regard, D2.2 is positioned within WP2 which operates as the technical backbone of EMERALDS, linking the scientific outputs and developments with cutting-edge technologies and resource management tools functional across the computing continuum. This entails, introducing established best practices from serverless computing, network function virtualization, microservices architectures and edge-to-cloud communication protocols. Ultimately, the EMERALDS toolset possesses all the foundations supporting the interstitching of the DataOps, MLOPs and DevOps paradigms (Figure 1-1) and is organized in the layers described in D2.1 EMERALDS Reference architecture. Interactions with other tasks in WP2 have informed aspects of the development process and the overall integration points with execution environments envisioned as endpoints of the emeralds services' functionalities. As per the EMERALDS project toolset implementation plan (can be found in D2.1), D2.2 comprises a significant milestone, which concludes the first software design, development, implementation, and integration cycles emphasizing the iterative process of learning loops and software development.



Figure 1-1 The intertwined DataOps, MLOps and DevOps EMERALDS methodology

D2.2 releases a selection of methods catered to supporting the utilisation of the tools, algorithms and solutions reported in D3.1 and D4.1. In this sense, D2.2 is driven by the combination of know-how from the technical backgrounds of technical partners (software development SMEs, Industry software houses, technology integrators), core research partners from WP3, WP4 and WP5 and the prospective first users (use case partners and early adopters) of the EMERALDS toolset as envisioned in the Use Cases Scoping Document D5.1. D2.2. is also pivotal for the EMERALDS implementation since the integration of components to accommodate their wider use in multiple environments as in the case of D5.2-D5.8 is achieved via the software integration and deployment tools presented herein. Finally, D2.2 combines emeralds in an interoperable software stack which scales with different resource options, infrastructures, and ingested data sources, enabling data fusion from heterogenous sources as well as code execution in determined system nodes thereby inducing a holistic outlook over the treatment of extreme scale urban mobility data. This output is key for the activities of WP6, and the respective market valorisation and commercialization tasks aimed at assessing the exploitation potential and effectiveness of the toolset.

D2.2 serves as means for verification for MS2 Primary Implementation Cycle and Initial Integration (M18) releasing a containerized early version of the EMERALDS toolset enabling the establishment of data workflows and corresponding pipelines for use case demonstrators (in the frame of WP5) to conclude the first agile development sprint and gather feedback for the overall toolset performance during the 1st assessment cycle concluding in M24 (MS3 Use Cases 1st Assessment Cycle).

1.3 Contribution to WP2 and Project Objectives

The first version of the EMERALDS toolset is a central result of the EMERALDS project contributing to the project wide objectives as well as the accomplishment of the WP2 objectives.

D2.2 is directly connected with Objective 1 (O1) Design a service-oriented reference architecture of a palette of services ('emeralds') for extreme scale urban mobility data analytics, underpinned by a distributed computing environment that includes edge/fog nodes and cloud nodes, that ensure that both edge and cloud processing contribute towards establishing a robust processing pipeline. Specifically, D2.2 fleshes out the common vision of unifying the novel tools and technologies delivered within the frame of WP3 and WP4, along with provisions for the wrapping of D2.6 security tools, in an interoperable software stack - the EMERALDS toolset. Furthermore, D2.2 delivers guidelines for the iterative and continuous improvement of the emeralds' performance and efficiency. It performs a cutting-edge integration of tools into a plug-and-play easy to use proof-of-concept TRL 5 toolset. The developed codebases and configurations ensure a streamlined, robust, efficient, and scalable

deployment process for a variety of urban mobility analytics services from edge to cloud also instilling adoption and integration across urban mobility ecosystems.

Additionally, D2.2 contributes towards the fulfilment of O2 **Develop extreme-scale acquisition and processing methods and tools for urban mobility data**, which will be scalable with the data at hand, and at the same time facilitate accurate and low-latency data collection, pre-processing (including cleansing, filtering, smoothing, etc.), mining, fusion, and management and O3 **Develop mobility data analytics and AI/ML tools and services – MAaaS**, appropriately designed to perform along the edge/fog/cloud continuum to achieve substantial speedups for analytics jobs. These include location-aware analytics aimed at providing actionable information close to the data source, as well as AI/ML algorithms providing fast and accurate predictions, respectively by housing the tools released under WP3 and WP4 under a commonly accessible repository and container registry that includes containerized versions suitable for deployment across a multitude of computational environments (spanning the entirety of the computing continuum).

With respect to (O4) **Demonstrate and measure the efficiencies of the novel Extreme Scale Analytics services through three pilot use cases and validate the concepts and tools usefulness as well as overall improvements in extreme data workflows through two early adoption applications**, this deliverable substantiates the integration of emeralds into use case environments, permitting the validation and assessment of performances and extreme scale capabilities in various urban mobility data analytics workflow scenarios across Europe.

Significant contribution is also evident in the attainment of the WP2 objectives as the software accompanying this report concentrates novel methodologies that advance state-of-the-art architectures mostly dealing with Big Data Processing and Analytics in cloud computing environments introducing capabilities to handle heterogeneous structured and unstructured data streams from multiple sources offering integration points and interoperability with the core WP3 and WP4 services. To another extent, the solution addresses common challenges met in MAaaS data pipelines such as seamless integration of data sources and processing components and the implementation of ready-to-go integration connectors that allow the interoperability of internal and external data sources.

1.4 Structure of the document

The structure of the document is as follows:

- Chapter 2 provides information regarding the containerization of the Emerald services that has been developed as part of WP3 and WP4 activities and delivered as part of the D3.1 and D4.1. It also presents the benefits, the prerequisites, and the methodology to achieve the containerization.
- Chapter 3 provides an overview of the Continuous Integration/Continuous Delivery methodology, the tools, and best practices to facilitate the process within the context of EMERALDS Project. This chapter is a continuation of Chapter 2 and present the entire Development and Operational cycle.
- Chapter 4 refers to the Emeralds Services Integration plan. The chapter presents the service architecture and communication patterns. The integration plan plays a significant role in the reusability of the Emeralds services.
- Chapter 5 is focused on the integration of the Emerald Services with the Analytics as a Services platforms of the project, CARTO's Mobility Analytics Cloud platform and Mobility AI as a Service platform of ATOS.
- Chapter 6 provides the conclusions and next steps of the EMERALDS toolset development process describing the foreseen improvements towards the release of the second and more mature version of the toolset.

2 EMERALDS Toolset

The EMERALDS toolset provides an advanced solution for large-scale urban mobility analytics as a service, offering exceptional value to city planners, transportation authorities, and mobility service providers. It is specifically designed to manage high-volume, high-velocity, and diverse data, enabling comprehensive and real-time analysis of urban mobility patterns. The analytics methods, referred to as ‘emeralds,’ are implemented within specific tool categories (emeralds typology) and can be easily configured and integrated into task sequences (service containerization in T2.1). The reference architecture upon which the EMERALDS toolset design is based can be seen in Figure 2-1 as reported in D2.1.

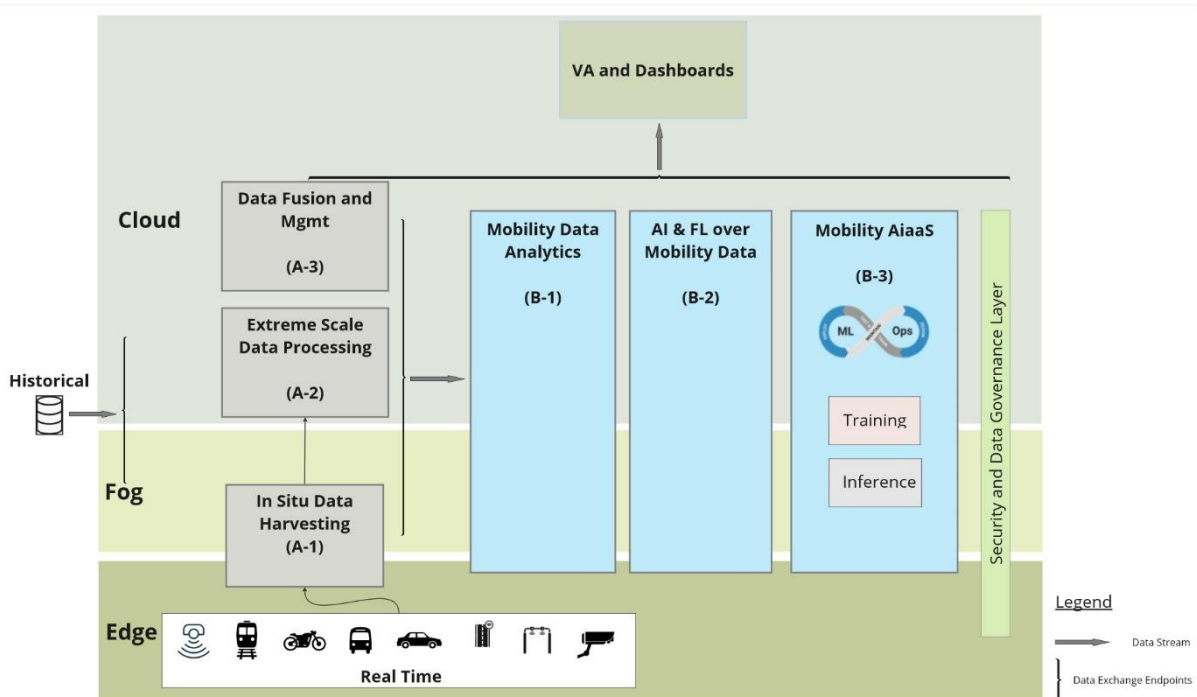


Figure 2-1 The EMERALDS Reference Architecture

The added value of the EMERALDS toolset lies in its ability to deliver precise, scalable, and privacy-aware analytics. The toolset maximizes the use of edge and fog computing devices, ensuring efficient data processing, and minimizing bandwidth requirements. The EMERALDS toolset is a dynamic and evolving software repository (v1 released M18) housing several key software components from WP3 and WP4 as in Table 2-1 along with their Dockerfile links and Continuous integration and Continuous Delivery (CI/CD) configuration files.

Table 2-1 - Emerald Toolset Repositories for Containerization and CI/CD

Task/Emerald	Dockerfile	CI configuration file	CD configuration file
Privacy aware data ingestion (T3.1)	privacy evaluation Dockerfile link preprocessing Dockerfile	Work In Progress/ To be reported in D2.3	Work In Progress/ To be reported in D2.3

Extreme-scale stream processing orchestrator (T3.1)	Dockerfile github link	Github workflow link	Github Kubernetes links
Extreme-scale map-matching (T3.2)	Dockerfile github link	Github workflow link	Github Kubernetes links
Weather enrichment (T3.2)	Dockerfile Github link	Github workflow link	Github Kubernetes links
Spatio-temporal querying (T3.2)	Dockerfile Github link	Github workflow link	Github Kubernetes links
Hot-spot analysis (T3.2)	Emerald to be reported in D3.2	Work In Progress/ To be reported in D2.3	Work In Progress/ To be reported in D2.3
Mobility/trajectory data compression (T3.3)	Dockerfile Github link	Github workflow link	Github Kubernetes links
Sensor (GPS, GTFS, radar, etc.) data fusion (T3.3)	Dockerfile Github link	Github workflow link	Github Kubernetes links
Probabilistic Trip Chaining¹ (T4.1)	Work In Progress/ To be reported in D4.2	Not Applicable	Work In Progress/ To be reported in D2.3
Dropoff/Destination Prediction² (T4.1)	Dockerfile link	Not Applicable	Work In Progress/ To be reported in D2.3
Monitoring and Forecasting Shared Mobility Demand³ (T4.1)	Emerald to be reported in D4.2	Not Applicable	Work In Progress/ To be reported in D2.3
Trajectory Data Analysis⁴ (T4.1)	Not Applicable/ QGIS Processing Trajectorytools plugin	Not Applicable/ QGIS Processing Trajectorytools plugin	Not Applicable/ QGIS Processing Trajectorytools plugin
Real-Time Extreme Scale Map Matching (T4.1)	Not Applicable/ Close-Source Emerald from SISTEMA	Not Applicable/ Close-Source Emerald from SISTEMA	Not Applicable/ Closed-Source Emerald from PTV
Traffic State / Flow Forecasting (T4.2)	GMAN Dockerfile link MSTL-ARIMA Dockerfile	Not Applicable	Work In Progress/ To be reported in D2.3
Parking Garage Occupancy Prediction⁵ (T4.2)	Work In Progress/ To be reported in D4.2	Not Applicable	Work In Progress/ To be reported in D2.3

¹ Previously referred to as “Probabilistic Approach for Trip Chaining” in D2.1

² Previously referred to as “Trajectory/Route Forecasting and Origin/Destination Estimation” in D2.1

³ New emerald, not previously listed in D2.1

⁴ Previously referred to as “Trajectory Data / Travel Time Analysis” in D2.1

⁵ Previously referred to as “Parking garage occupancy forecasting” in D2.1

Crowd Density Prediction⁶ (T4.2)	Work In Progress/ To be reported in D4.2	Not Applicable	Work In Progress/ To be reported in D2.3
Active Learning & XAI for Crowd Prediction⁷ (T4.2)	Work In Progress/ To be reported in D4.2	Not Applicable	Work In Progress/ To be reported in D2.3
Active Learning for Risk Category Classification (T4.2)	Emerald to be reported in D4.2	Not Applicable	Work In Progress/ To be reported in D2.3
Federated Learning Models for Mobility Data (T4.2)	Emerald to be reported in D4.2	Not Applicable	Work In Progress/ To be reported in D2.3

On top of the Emerald Services developed as part of the T3.1 and T4.1, a Continuous Delivery environment has been developed to facilitate the Emeralds Development Process. This execution environment is automatically configured and can be utilized as a hosting service for external users of the EMERALDS toolset. End-users are having the option either to use their own platform and integrate the emeralds Services or they may offer the hardware infrastructure – such physical or Virtual Servers with Linux Operating System installed – and use the scripts to install and configure the required software. Table 1-2 lists direct links on sub-folders of the emeralds-hosting-services repository. Each sub-folder is responsible for specific provisioning task. Further details on the subject are provided on Section 4.

Table 2-2 - Emeralds-hosting-services

Github link	Description	Status
Configuration Scripts	Scripts that should be used to configure the servers with the required software	Completed
cloud-infrastructure	Configuration files to create cloud resources, such as Virtual Machines, Networks, etc.	Work in progress. To be reported in D2.3
Kubernetes Configuration files	Manifest files used by the CI/CD Platform to configure system resources.	Work in progress. To be reported in D2.3

2.1 Integration Plan

This subsection outlines the integration strategy for the diverse services and applications under development, focusing on how these components will interact, integrate, and eventually deliver an interconnected urban mobility toolset.

⁶ Previously referred to as “Crowd density forecasting” in D2.1

⁷ Previously referred to as “Active Learning & XAI for crowd/flow forecasting” in D2.1

2.1.1 Service Architecture and Communication

The service architecture and communication are based on some basic rules that will be analysed in this sub-section. Due to the diversity of the Emerald services, the rules shall not apply uniformly to all of them, and the exceptions shall be provided as needed. Furthermore, regarding the WP4 Emeralds, the rules mainly apply for the inference of the generated models and not for the training. This is due to fact that the model training requires large datasets that cannot easily be fed through API interfaces. In such cases, the data are stored on databases and the related Emeralds are interacting with them directly.

API Development and RESTful Services

Each Emerald Service is already offering or will offer its own set of **APIs**, ensuring robust and flexible communication capabilities not only between other Emeralds, but also with external services. These APIs will support **RESTful** interactions, allowing for efficient and standardised communication patterns. This approach will enable the Emeralds services to easily being adapted from both existing and new Urban MaaS solutions, facilitating the creation of new and enhanced data pipelines. Where applicable Emerald Service will not only function as REST Servers but also as REST Clients, making remote calls to external services.

To facilitate the usage of different data sources and output destinations a modular approach for the Emeralds services have been followed. Each service is broken into three components, as it is depicted on Figure 2-2. The most important component is the “Emerald-Core”, the software component that performs the implements the algorithm. The core component will have a strictly defined interface, both for data input as well as data output. The two other components are responsible with the data ingestion and a data output to external services. These components are interchangeable and different implementation may offer access to various data sources, such as APIs, databases, and shared folders. An instantiation of this architecture will be provided by Atos as part of their Mobility AlaaS platform, which is offering an API Wrapper for the ML/AI bases Emeralds. More information for this initiative will be available on D4.2 Mobility Data Analytics and Learning Services v2.

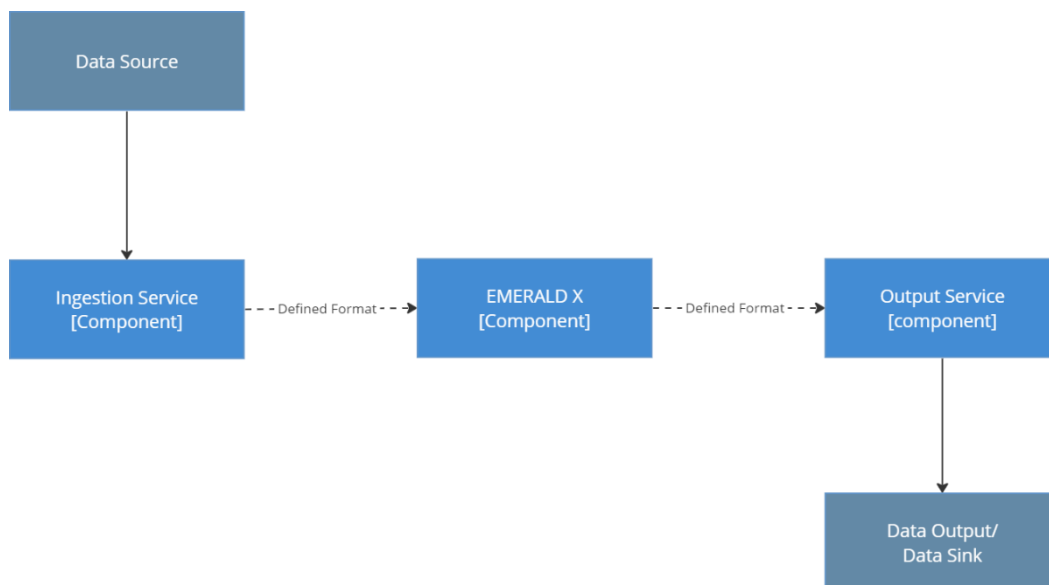


Figure 2-2-Emerald Service Integration Architecture

Based on the deliverables D3.1 and D4.1 two exceptions have been identified.

- “Trajectory Data Analysis”, which is a QGIS⁸ Processing Trajtools plugin, and therefore it can only be used under the strictly defined rules of QGIS Processing toolbox.
- “Mobility/trajectory data compression” Emerald, which is an extension of the MobilityDB⁹. As such the specific Emerald adheres to the programming interface offered by the MobilityDB.

Event-Driven Architecture and Message Brokering

To facilitate real-time data processing and adhere to modern data processing architecture practices, an **event-driven architecture** is also supported. The Emerald service “Extreme-scale stream processing orchestrator” of Task 3.1, has been designed as a lightweight message broker, among other functionalities that offers, and it shall be the core component of such approach. Emerald services shall utilize it to produce or consume data in the form of events. This approach will allow to streamline the creation of effective data pipelines by decoupling the services, while providing the means to achieve the required scalability for extreme scale data processing. The “Extreme-scale stream processing orchestrator” can be used either by other emeralds services, or from external services as well. The only prerequisite is that each message broker user should adhere to the interface defined by the ““Extreme-scale stream processing orchestrator””.

Containerization

The main goal of the EMERALDS Project is the creation of **Urban mobility toolset of services**. To this end, all Emerald Services shall be **containerized**, as described in Section 3, and will be available to all interested parties through the [EMERALDS-Horizon Github Container Registry](#). The landing page of the repository shall provide an extended list of all available services, along with links to detailed instructions and examples regarding the usage of the services. Moreover, the latest versions of the services shall be available as containers.

The use of containerization packages the execution requirements of the EMERALDS services, decoupling them from the operating system. This ensures error-free deployment and facilitates the scaling of services based on demand.

⁸ <https://qgis.org/en/site/>

⁹ <https://mobilitydb.com/>

3 Containerization of EMERALDS Services

In this section, we provide an overview of the containerization and its relevance to the deployment of EMERALDS services in urban mobility data workflows and applications. The computing continuum introduces new opportunities for big data pipeline management addressing heterogeneity and untrustworthiness of data resources. Despite the acclaimed disruptive effect of cloud computing in the past years, stating resources accessible and configurable as a service for a variety of internet applications in tandem offering elastic capacity and customizable connectivity over large scale networks, big data processing and analytic tasks' resilience, sustainability and collaborative requirements ensue an interoperable end-to-end ecosystem that shifts data centre operations, infrastructure services and functions closer to the data sources, meaning the manipulation of remote nodes and adaptation to execute processing, privacy preservation or lite analytics tasksⁱ. Figure 3-1 depicts an ecosystem for managing big data pipeline lifecycle management on the computing continuum comprising six phases involving the relevant stakeholders at each stage.

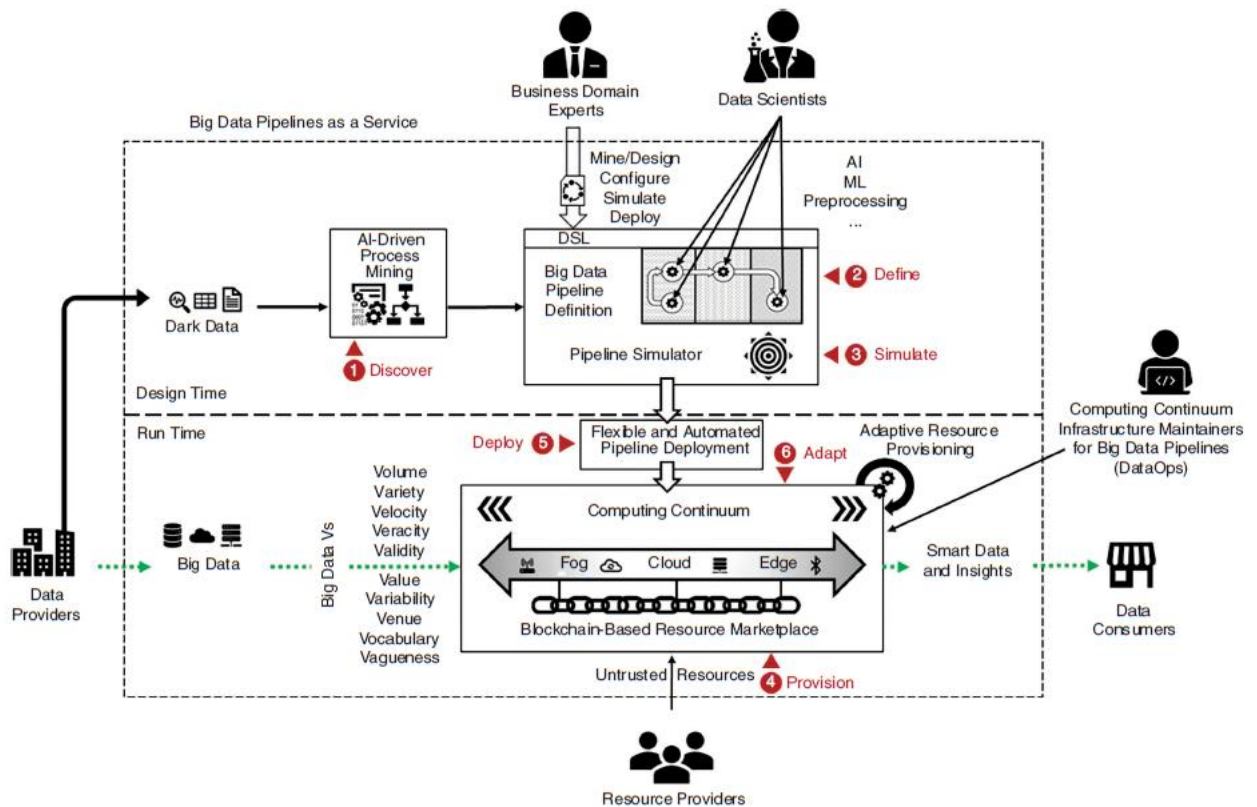


Figure 3-1 An ecosystem for managing data pipelines on the computing continuum ⁱ.

Serverless computing and containerization are two pivotal technologies that synergize to enable advanced edge-to-cloud extreme data analytics. Serverless computing abstracts away the complexities of managing underlying infrastructure, allowing developers to focus solely on writing and deploying code and is ideal for handling sporadic, event-driven workloads, typical in urban mobility analytics, where data influxes can be unpredictable.

Containerization, on the other hand, encapsulates applications and their dependencies into portable, self-sufficient units called containers. These containers can run consistently across different computing environments, from edge devices to centralized cloud servers.

A combination of serverless computing and containerization provides a robust foundation for edge-to-cloud data analytics by ensuring seamless scalability, portability, and orchestration of data processing tasks. Containers ensure that the serverless functions have all necessary dependencies packaged within, facilitating rapid deployment and execution across heterogeneous environments. Together, these technologies lay the groundwork for extreme data analytics by enabling a distributed, flexible architecture. Serverless functions can be deployed at the edge to preprocess and filter data close to the source, reducing latency and bandwidth usage. Processed data can then be aggregated and further analysed in the cloud using containerized applications, which benefit from the scalable compute resources available there.

Containerization is a powerful strategy for streamlining the development, deployment, and management of the diverse computer services. Containers package application and their dependencies into a single unit, making them highly **portable** across different infrastructure environments, such as development, testing, and production. These single units can be executed under any Operating System equipped with a container engine. But the most crucial factor for the EMERALDS project is the **multi-language support**, which allows each partner to develop their own service on the programming framework that best meets their business and functional requirements. On the same ground, this approach ensures **consistency** between different environments, by minimizing the risk of configuration drift and deployment errors. On the aspect of application **security**, each container provides a separated runtime environment, thus enforcing isolation for the deployed services. All the above characteristics are key elements for the EMERALDS project on the design of a toolset that can be used by other Mobility Analytics as a Service platforms and solutions.

The benefits of Containerization are extremely important during the development phase, as the containers are the base for creating effective CI/CD pipelines. As we will further analyze on Section 2, a CI/CD pipeline allows for the **streamline of the development** process by allowing faster testing and deployment of the service.

Containerization builds upon the concepts of **virtualization** by providing a lighter and more efficient alternative to certain use cases. Containerization can be seen as a complementary to virtualization, offering additional deployment options and flexibility within virtualized environments. In many scenarios on private and public cloud, organizations deploy containers within virtualized environments to leverage the benefits of both technologies. This is a very common approach with the utilization of Kubernetes orchestration tool, which is running on top of Virtual Machines (VM) and is managing the lifecycle of containerized applications.

3.1 Container engines

The idea of containerization was first introduced in 2008, with the Linux Containers (LXC). The LXC was based on Linux Kernel features such as namespaces and groups to provide lightweight process isolation. The concept was popularized with the **Docker**, which was launched in 2013 and revolutionized the containerization by introducing a user-friendly interface and standardized container image format, thus making it the de-facto standard.

Docker follows a client-server architecture model, where the Docker client interacts with the Docker daemon (also known as the Docker engine), a background service responsible for managing Docker objects such as containers, images, networks, and volumes. It listens for Docker API requests from the Docker Client and executes them accordingly. Other core components of the Docker framework are the **“images”**, which are read-only templates that contains the application and its

dependencies. The image is the output of the docker building process and it is stored in **Registries or Container Hubs**. Images can be downloaded and deployed as docker **containers**, which are isolated processes on the host systems, with their own filesystem, network, and process space.

Over the years alternative container engines have been developed. Podman¹⁰ has been considered a direct replacement for the Docker ecosystem. Its main characteristic is its daemonless engine, which does not require root access on the hosted system. It is considered more secure compared to Docker, while it maintains full compatibility with it. As a newer technology, it is less mature, with smaller community and third-party integrations.

Docker is based on **containerd**¹¹ engine. This is an industry-standard container runtime which has been donated to Cloud Native Computing Foundation (CNCF) and adheres to the Open Container initiative (OCI) standards. It provides only the core functionalities for container management and therefore is much lighter than the Docker Engine. It is one of the available container runtimes for Kubernetes. Another alternative is **cri-o**¹², which is a runtime primarily tailored for Kubernetes, with focus on performance and security. Its tightly integration with Kubernetes, make it a less suitable solution for standalone deployments or scenarios requiring advanced container features. It is worth noting that the choice of container engine does not impact the container image building process, since all engines comply with the same OCI Standards, ensuring interoperability. Consequently, the decision to use a particular engine in testing or production environments should be guided by the specific needs of the underlying hardware or its compatibility with the container management platform.

3.2 Development Process

To achieve the containerization of the Emerald services, **each emerald repository must contain at least one Dockerfile**¹³. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build command users can create an automated build that executes several command-line instructions in succession. This file is essentially a blueprint for building a Docker image, which is the final output of the CI part of the CI/CD pipeline.

3.2.1 Dockerfile walkthrough

Figure 3-2 depicts a simplified Dockerfile for emeralds “Extreme-scale stream processing Orchestrator”. The following key steps are explained.

FROM python:3.9-slim: The FROM command defines the base image, which is used to create the final image for the application. In this specific example A Linux based image with Python¹⁴ version 3.9 installed is being used. There are a great number of base images available that support different programming languages. Developers may also use a base Linux image and install the desired language ecosystem through the use of the **RUN** command.

WORKDIR /usr/src/app: This sets the working directory inside the container.

COPY . . _: The COPY Command copies the current directory's contents into the working directory in the container. The “.” Is an alias for Linux current directory.

¹⁰ <https://podman.io/>

¹¹ <https://containerd.io/>

¹² <https://cri-o.io/>

¹³ <https://docs.docker.com/reference/dockerfile/>

¹⁴ <https://www.python.org/>

RUN pip install --no-cache-dir -r requirements.txt: The RUN Command is used to pass and execute shell commands into the container. In this case, the “pip install --no-cache-dir -r requirements.txt” is being executed within the docker container and it is responsible for installing all required python modules for the application. As mentioned above, the command can be used to further customize the final docker image.

EXPOSE 8080: This makes port 8080 available for use in the container.

ENV EMERALD_APP=data_broker: The ENV command can be used to pass environment variables into the container. In this case sets an environment variable FLASK_APP to app.py.

CMD ["python", "./main.py"]: The CMD Command is providing the running command for the docker container. Docker containers are designed to support at least a process, which is usually defined through the CMD command.

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy the current directory contents into the container at /usr/src/app
COPY . .

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 8080 available to the world outside this container
EXPOSE 8080

# Define environment variable
ENV EMERALD_APP=data_broker

# Run app.py when the container launches
CMD ["python", "./main.py" ]
```

Figure 3-2 - Dockerfile Walkthrough Example

4 Continuous Integration/Continuous Deployment

Continuous Integration and Continuous Delivery pattern, usually referred to as CI/CD, is a set of practices that automates and streamline the development lifecycle of software applications. Its main goal is to increase the productivity of the software development teams by seamlessly integrating code changes, automated testing and rapidly deploy them into production environments.

CI/CD is closely related to Agile methodologies, which emphasizes into iterative development, through customer collaboration. CI/CD aligns with these principles by enabling teams to deliver small increments of functionality frequently, gather feedback quickly, and adapt their approach accordingly. These principles allow the development teams to maintain a higher productivity levels by reducing the overhead associated with manual testing, integration, and deployment tasks. Figure 4-1 depicts this iterative process.

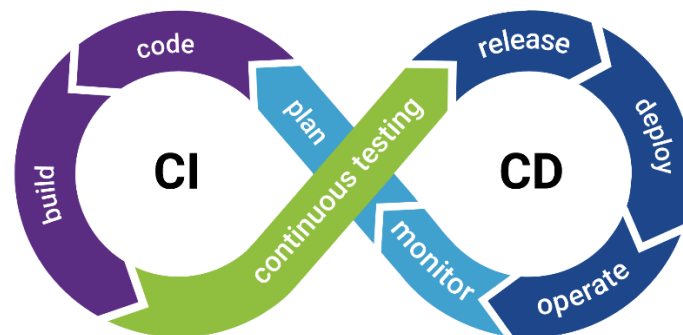


Figure 4-1 - CI/CD Process

By automating these repetitive processes on every code commit, the teams identify issues early in the development process and improve the quality of the software. Furthermore, it improves the cost efficiency as it reduces the manual effort by the engineers, avoid error-prone tasks while increasing the delivery speed. It is also worth mentioning that these practices allow the individual members of a development team to grow as engineers by allowing them working on more meaningful tasks rather than on non-brain activities.

The CI/CD pipelines that we will be described later in this section, are by design scalable and flexible. They can adapt on varying workloads and project requirements, as they can be adapted with minimum effort to accommodate different development workflows, environments, and technology stacks, while providing the necessary scalability as development teams may increase in size and number.

4.1.1 State of the Art in CI/CD

The CI/CD practices are constantly evolving rapidly, driven by the advancements in technology and methodologies. One such method is the **infrastructure-as-Code (IaC)**, which considers the infrastructure as code, allowing for automated provisioning and configuration management. This enables deployments to be consistent and repeatable across different environments.

Supplementary to IaC, the **GitOps** approach uses Git as the source of truth for application code and declarative infrastructure. With GitOps any changes to infrastructure are following a similar approach as the changes in the code by following the same Git flows, such as pull request and review, versioning, logging, auditing, transparency, and better collaboration between the various actors involved.

As cybersecurity is one of the most critical aspects in the design and implementation of software module, a trend that has been observed is a **shift-left Security**. This refers to the integration of security considerations earlier in the pipeline and closer to the development itself. Practices such as static code analysis, policy enforcement and vulnerability scanning are happening on every commit automatically and may deny the acceptance of a code commit.

A CI/CD Pipeline may consist of multiple steps with a lot of moving parts or variations based on the deployment targets. **Observability and monitoring** tools are essential for gaining insights into the performance, availability, and health of applications and infrastructure deployed through CI/CD pipelines. Advanced monitoring solutions incorporate telemetry data, distributed tracing, to facilitate proactive monitoring and easier troubleshooting for the DevOps team.

The most prominent advancement thought in this area is the **adoption of ML/AI** models to improve various aspects of a CI/CD pipeline. Automated unit-tests can be generated by scanning the source code and improve the code coverage or identify potential bugs. Recommendations on existing test cases may improve the time needed to execute the test suites. In the area of the static code analysis, AI/ML may provide better insight and identify more complex defects, refactoring opportunities, and security risks. AI/ML-powered self-healing mechanisms can automatically detect infrastructure failures or performance degradation in CI/CD environments. By analysing telemetry data, distributed logs, and system behaviour in real-time, these solutions can dynamically adjust configurations, scale resources, or restart services to maintain system stability and availability.

4.1.2 CI/CD Platforms

A diverse range of CI/CD platforms is available, spanning both open-source and commercial offerings. Each platform incorporates a suite of state-of-the-art tools designed to streamline development workflows and enhance deployment efficiency.

Jenkins¹⁵ is one of the most popular and widely used open-source frameworks for CI/CD. It may be considered as a pioneer on the domain as it was one of the original solutions for setting up CI/CD pipeline on local development environments. There is vast support from the community, both in the form of plugins availability as well as through various forums. One of the most impressive features is the support of pipeline as code, which can be integrated with version control.

In the area of Cloud hosted CI/CD tools, all major Cloud Providers offer their own solutions such as Azure DevOps¹⁶, AWS CodePipeline¹⁷, Google Cloud DevOps¹⁸ suites. Similarly, the most common cloud git providers are also offering their own platforms, the GitHub Actions¹⁹ and GitLab pipelines²⁰. In all the above cases, each provider offers the required infrastructure and a Domain Specific Language, which allows the developers and DevOps engineers to define their pipeline in a declarative way.

For EMERALDS, the decision was to implement the CI/CD Pipelines using the GitHub Actions. The rationale behind this was due to consortium's prior selection of GitHub as the preferred code repository, owing to its widespread adoption within the developer community²¹. Furthermore, all

¹⁵ <https://www.jenkins.io/>

¹⁶ <https://azure.microsoft.com/en-us/products/devops/>

¹⁷ <https://aws.amazon.com/codepipeline/>

¹⁸ <https://cloud.google.com/devops>

¹⁹ <https://github.com/features/actions>

²⁰ <https://docs.gitlab.com/ee/ci/pipelines/>

²¹ <https://survey.stackoverflow.co/2022/#section-version-control-version-control-platforms>

essential features, including code repository, CI/CD pipeline resources, and Container Registry, are available either free of charge or at a minimal cost.

4.1.3 GitHub Actions

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform, part of the GitHub Development Ecosystem, designed to automate workflows for software development and deployment directly within the GitHub repository.

Next, we will provide a brief description for the core components of the GitHub Actions ecosystem²².

Workflow is a configurable automated process, defined by a yaml²³ file checked into the same repository that hosts the application source code, under a specific subfolder. Multiple workflows might exist on the same repository to perform different tasks, such as building and storing the application as image on Container registry, while other workflows are responsible for deployment on different environments, such as testing, staging and production.

An **event** is a method for triggering workflow runs. A push commit, a creation of a pull request can be considered such events. Workflows can also be triggered through scheduler or even manually from the Github actions dashboard given the proper configuration.

A **job** is a set of steps executed on the same runner. Jobs can run sequentially or in parallel, depending on the workflow configuration. Each job runs on a separate instance of the execution environment, isolated from other jobs in the workflow. Each **step** is a single task, which can include shell commands, scripts or predefined actions provided by GitHub, third-party providers or created by the owner of the repository. Steps are executed sequentially and on the same runner, thus they can share data. **Actions** are reusable extensions that simplify the creation of workflows. They can be written in any programming language and uploaded to GitHub Actions as a docker container.

Workflows are being executed on **runners**, which are virtual machines or docker containers. GitHub provides hosted runners with pre-installed software environments for common platforms such as Linux, Windows, and macOS. Additionally, repository owners may set up self-hosted runners to execute workflows in their own infrastructure, enabling greater customization and control over execution environments.

4.2 Continuous Integration

In this subsection, we will provide further details regarding the CI part of the pipeline. This part covers the building of the emerald Service, including the static analysis of the code, as well as the packaging and subsequent storage of the emerald Services in the Container Registry. Additionally, it will outline the management console features of GitHub Actions and provide strategies for debugging a failed pipeline.

GitHub Actions is enabled by default on all repositories and GitHub Organizations. Policies may be enforced to regulate how organization members utilize GitHub Actions, although the granularity of those depends on the subscription plan that have been chosen by the Organization. Common policies include access to local runners, restrictions on using reusable workflows and actions, automated

²² <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>

²³ <https://yaml.org/> . YAML is a human-readable data serialization language that is used for writing configuration files.

repository modification triggered by running workflows on main line and forks and configuration of GITHUB_TOKEN permissions. Typically, anyone with write access to the repository have the authority to manage most of these policies and, naturally, create or modify workflows, since they are stored within the repository as YAML files.

For the creation of a workflow, properly formatted YAML files is being added under the each Emerald `#{REPOSITORY_ROOT}/.github/workflows` folder. The files have been creating either manually or through utilizing the GitHub Actions wizard, a feature provided by GitHub. This wizard scans the repository and suggests a series of configurable workflows based on the technology stack used. Additionally, GitHub Marketplace offers a wide range of available workflows.

The workflow YAML file consist of three distinct sections. The first one provides basic information about the creator, the usage, and any other the potential configuration of the workflow. The second part defines the events that may trigger it. Multiple events can be defined by the development team in a “or” configuration. The last section holds utmost significance, as it defines the jobs requiring execution, the corresponding runner assigned to accommodate them, and the steps necessary to accomplish the task.

Figure 4-2 presents all the steps that need to be executed in the CI Pipeline, for each non-ML Emerald Service

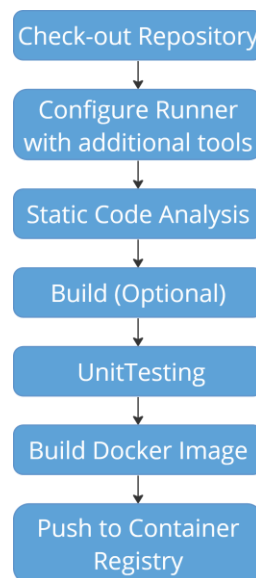


Figure 4-2 - CI Path

Check-our Repository: Since a CI pipeline is being executed on a runner, the actual repository needs to be cloned and checked out locally on that runner. This is a mandatory step for such pipelines and GitHub offers a reusable workflow²⁴ for performing this task.

Configure Runner with additional tools: Depending on the tech stack used by the Emerald Service, and mainly the selected programming language, different steps are required, and different tools should be used. As an example, services based on scripting languages such as Python do not require a build step compared to services written in Java²⁵. Consequently, either a preconfigured

²⁴ <https://docs.github.com/en/actions/using-workflows/reusing-workflows>

²⁵ <https://www.java.com/en/>

runner needs to be used for the execution of a workflow or extra configuration steps should be considered. GitHub offers a few generic preconfigured runners, which can be used as a base for further customization. As an alternative, an organization may create their own preconfigured runners based on their needs, thus reducing the total execution time of a pipeline by avoiding those extra repetitive provisioning steps.

Static Code Analysis²⁶ is the analysis of software programs performed without the need to be executed. The primary use of them is to improve code quality, security, and maintainability during the software development process and it can be considered as part of a Code Review process²⁷. The sophistication of the analysis performed by tools varies from those that only consider the behaviour of individual statements and declarations, to those that include the complete source code of a program in their analysis. The uses of the information obtained from the analysis vary from highlighting possible coding errors (e.g., the lint tool) to formal methods that mathematically prove properties about a given program.

The most notable strengths are the scalability and the identification of defects and bugs on areas such as SQL Injection, buffer overflow, memory leaks and syntax errors with the latter especially applies for scripting languages. Also, they are great to enforce coding standards, an approach that ensures consistency, readability, and maintainability across the codebase, especially in large development teams or projects with multiple contributors.

On the other hand, as a main weakness can be considering the high number of false positives and false negatives identification that led to additional maintenance effort. This is happening due to limited context awareness as the tools operate without any knowledge of environment or dependencies, or difficulty in analysing complex codes. To that end, the state-of-the-art on Static Code Analysis is the use of sophisticated ML models for analysing code and even writing code fixes automatically. This brings up another limitation of the static code analysis pattern, the resource intensive nature of the tools, which may lead to longer pipelines execution times, and the dependency on expensive hardware resources. Furthermore, there is significant cost for the use of such tools and organization should make their own cost-benefit analysis, before proceeding with the adoption of them.

In the context of EMERALDS project, we have mainly focused on how such tools can be integrated in the CI/CD pipeline. It is also important to state that different set of tools are required for different tech stack that have been used from the EMERALDS development teams. At the time of drafting this report, “templated” workflows have been defined for four different programming languages, Python, Java, Scala²⁸ and Rust²⁹.

The **build** process for a software program refers to the series of steps and operations required to compile, package, and prepare the source code into executable software artifacts or deliverables. This step may not apply for script-based languages, such Python, especially since it has been decided that the main mean of packaging for Emerald Services shall be a container.

Unit Testing is a software testing method where individual units of source code are being tested to determine whether they are fit for use. The unit refers to the smallest component that can be isolated with the complex structure of an app, and it could be a method or a class for Object Orient Programming languages³⁰. The execution of unit tests on EMERALDS Project requires the

²⁶ https://owasp.org/www-community/controls/Static_Code_Analysis#

²⁷ https://en.wikipedia.org/wiki/Static_program_analysis

²⁸ <https://www.scala-lang.org/>

²⁹ <https://www.rust-lang.org/>

³⁰ https://en.wikipedia.org/wiki/Unit_testing

implementation of such test cases by Emeralds development teams and therefore it has been marked as optional in the CI Pipelines.

Build Docker Image: EMERALDS project strictly defines the containerization of the services as a mean for delivering the EMERALDS Toolset to the community. To that end, each emerald will be packaged as docker image and **pushed on GitHub's Container Registry**. More on that subject can be found on section 2. What is related to CI Pipeline is the access to the Container Registry and more specifically the authorization for a pipeline to push build docker images to the Registry. GitHub Actions provides a mechanism that generates a secret key, named `GITHUB_TOKEN`, which can be used by the pipeline to be complete this action. The Repository owners may select to create their own secret instead and define their access policy that will be bind to the key. In the context of EMERALDS Project, we have used the internal generated process to authenticate pipelines. Furthermore, as part of the Docker Build process, we have taken advantage of the Docker Scout³¹ feature which supports the creation of Software Billing of Materials (SBOM) for the Docker Image. The SBOM provides to the emerald Service development team and the final user of it enhanced visibility and license compliance by listing all the third-party software in use and the relative license schemas that associated with them. It also offers support for Security and Vulnerability Management, by allowing the stakeholders to compare the components listed in the SBOM against known vulnerability databases.

4.2.1 GitHub Actions Management.

The GitHub Actions offers a management console that allows the developers to manage the Actions on the repository.

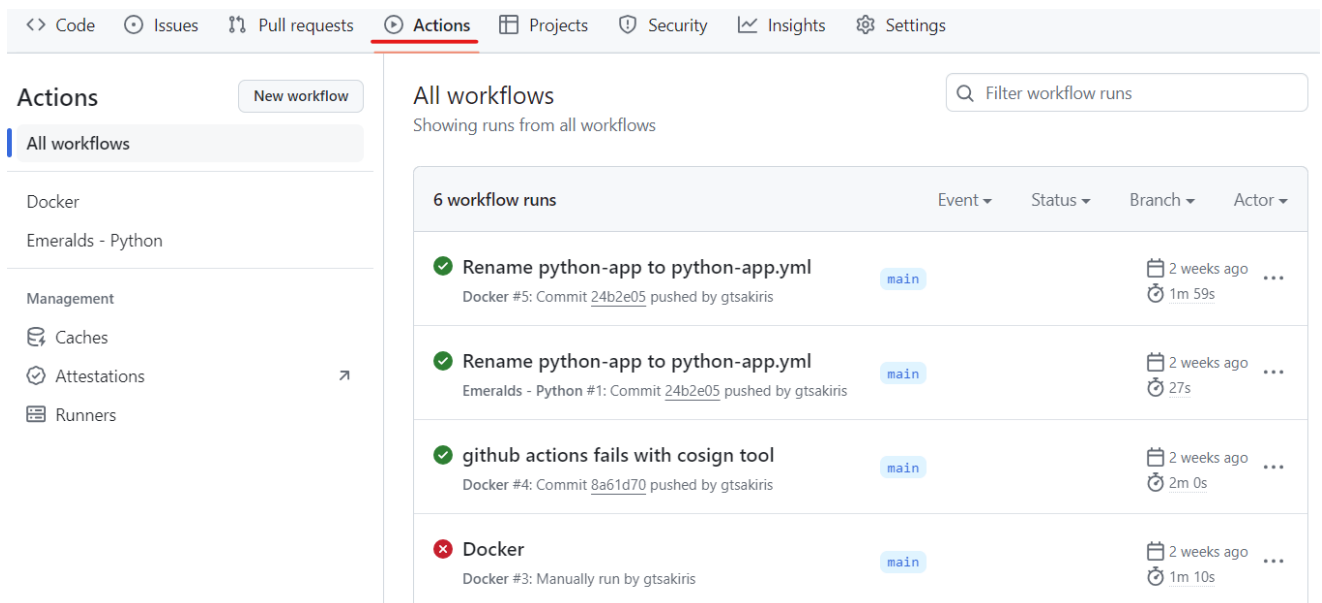


Figure 4-3 - Github Actions Management Portal

In Figure 4-3 there is a snapshot of the console. Each execution of a pipeline is being logged and presented in chronological order. By selecting one of the four workflows as presented, the Console User may inspect the output of the workflow. In Figure 4-4, there is an such an example.

³¹ <https://docs.docker.com/scout/>

Build and Push a docker image of an Emerald summary ...

Recommended fixes for image `ghcr.io/emeralds-horizon/data_broker:main`

Base image is `python:3`

Name	3.12.3
Digest	sha256:76831bb094456f98613115be37cc8d4f9ab0d128467726ed3160044c80546437
Vulnerabilities	critical 1 high 9 medium 7 low 116
Pushed	1 month ago
Size	381 MB
Packages	573
Runtime	3.12.3

The base image is also available under the supported tag(s): `3-bookworm`, `3.12`, `3.12-bookworm`, `3.12.3`, `3.12.3-bookworm`, `bookworm`, `latest`

Refresh base image

Rebuild the image using a newer base image version. Updating this may result in breaking changes.

✓ This image version is up to date.

Change base image


Tag	Details	Pushed	Vulnerabilities
<p><code>3-slim</code></p> <p>Minor runtime version update</p> <p>Also known as:</p> <ul style="list-style-type: none"> 3.12.3-slim 3.12-slim slim slim-bookworm 3-slim-bookworm 3.12-slim-bookworm 3.12.3-slim-bookworm 	<p>Benefits:</p> <ul style="list-style-type: none"> Minor runtime version update Image is smaller by 318 MB Image contains 417 fewer packages Image introduces no new vulnerability but removes 103 Tag is using slim variant 3-slim was pulled 26K times last month <p>Image details:</p> <ul style="list-style-type: none"> Size: 48 MB Runtime: 3.12.3 	1 month ago	<p>critical 1→0</p> <p>high 9→1</p> <p>medium 7→0</p> <p>low 116→29</p>
<p><code>alpine</code></p> <p>Tag is preferred tag</p> <p>Also known as:</p> <ul style="list-style-type: none"> 3.12.3-alpine 3.12.3-alpine3.19 3.12-alpine 3.12-alpine3.19 3-alpine 3-alpine3.19 alpine3.19 	<p>Benefits:</p> <ul style="list-style-type: none"> Image is smaller by 343 MB Image contains 517 fewer packages Tag is preferred tag Image introduces no new vulnerability but removes 128 alpine was pulled 41K times last month <p>Image details:</p> <ul style="list-style-type: none"> Size: 21 MB Flavor: alpine OS: 3.19 	1 month ago	<p>critical 1→0</p> <p>high 9→1</p> <p>medium 7→4</p> <p>low 116→0</p> <p>unspecified 0→1</p>

🔍 Vulnerabilities of `ghcr.io/emeralds-horizon/data_broker:main`


Figure 4-4 - Github SBOM Presented

Figure 4-5 is the generated output of the Docker Scout analysis that has been initiated as part of the CI pipeline and presents all vulnerabilities along with each severity. The analysis is utilized to minimize the attack surface of the generated image by applying the required security fixes. The process is automated and takes place during the image build phase, with the appropriate commands to be added in the Dockerfile.

Vulnerabilities of `ghcr.io/emeralds-horizon/data_broker:main`

▼  Image Reference `ghcr.io/emeralds-horizon/data_broker:main`

digest	<code>sha256:e73c34bd8d7efa3a936f3b5210c71c319ebb64c411ab92c8fd657ba0f2f438bb</code>
vulnerabilities	critical 1 high 9 medium 7 low 116
platform	linux/amd64
size	381 MB
packages	573

▶  Base Image `python:3`

▶ C 1 H 2 M 0 L 4	<code>git 1:2.39.2-1.1 (deb)</code>
▶ C 0 H 5 M 3 L 11	<code>bluez 5.66-1+deb12u1 (deb)</code>
▶ C 0 H 1 M 0 L 0	<code>pip 24.0 (pypi)</code>
▶ C 0 H 1 M 0 L 0	<code>libyaml 0.2.5-1 (deb)</code>
▶ C 0 H 0 M 2 L 2	<code>libwmf 0.2.12-5.1 (deb)</code>
▶ C 0 H 0 M 1 L 0	<code>nghttp2 1.52.0-1+deb12u1 (deb)</code>
▶ C 0 H 0 M 1 L 0	<code>mariadb 1:10.11.6-0+deb12u1 (deb)</code>
▶ C 0 H 0 M 0 L 13	<code>openjpeg2 2.5.0-2 (deb)</code>
▶ C 0 H 0 M 0 L 9	<code>tiff 4.5.0-6+deb12u1 (deb)</code>
▶ C 0 H 0 M 0 L 9	<code>openssh 1:9.2p1-2+deb12u2 (deb)</code>
▶ C 0 H 0 M 0 L 9	<code>imagemagick 8:6.9.11.60+dfsg-1.6+deb12u1 (deb)</code>
▶ C 0 H 0 M 0 L 7	<code>glibc 2.36-9+deb12u7 (deb)</code>
▶ C 0 H 0 M 0 L 7	<code>binutils 2.40-2 (deb)</code>
▶ C 0 H 0 M 0 L 4	<code>openldap 2.5.13+dfsg-5 (deb)</code>
▶ C 0 H 0 M 0 L 4	<code>patch 2.7.6-7 (deb)</code>
▶ C 0 H 0 M 0 L 4	<code>systemd 252.22-1~deb12u1 (deb)</code>

Figure 4-5 - Docker scout vulnerabilities analysis.

By selecting the name of the Workflow, the End-User is presented with a complete log of the pipeline that was executed along with basic performance metrics for each step. In Figure 4-6 we are presenting such a log. An interesting observation is the time consumed for the “Set up job” test, which

was approximately 30% of the entire workflow run time. A preconfigured runner would avoid this step and would bring down the required time, thus increasing productivity for large repositories with many contributors. The creation of such runners will be one of the enhancements that will be part of the “Deliverable D2.3 – Containerized EMERALDS Toolset v2”.

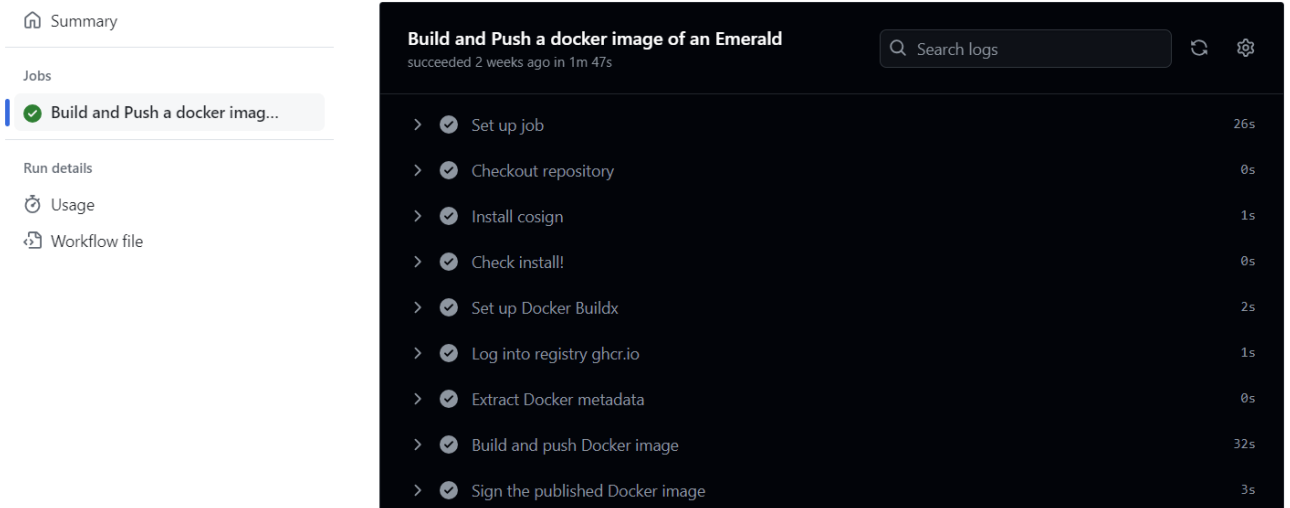


Figure 4-6 - Github Actions logging

4.2.2 Container Registry

EMERALDS main goal is the creation of a containerized toolset of Mobility Services that can be reused in multiple Use Cases, as it will be demonstrated at WP5 and WP6. Toward that end, the last steps of the designed CI pipeline are the packaging of each emerald service into a docker container and its storage into the **GitHub Container Registry**.

GitHub Packages along with the Github Container Registry are software packages hosting services that allow developers to host their own software packages privately or publicly and use packages as dependencies in their projects. GitHub Packages offers different package registries for commonly used package managers, such as npm³², RubyGems³³, Apache Maven³⁴, and Gradle³⁵. GitHub's Container registry on the other end, is optimized for containers and supports Docker and OCI images. As the goal of EMERALDS Project is the containerization of the Services, the use of Container Registry is preferred.

The permissions for a docker image are either inherited from the repository where the package is hosted or can be defined for specific users or organizations. Packages with granular permissions are scoped to a personal account or organization. Package admins may change the access control and visibility of the package separately from a repository that is connected (or linked) to a package. These rules allow the Project Partners to select how each emerald service can be published, by separating the source code from the executable deliverable.

³² <https://www.npmjs.com/>

³³ <https://rubygems.org/>

³⁴ <https://maven.apache.org/>

³⁵ <https://gradle.org/>

The Table 4-1 offers an Access Overview of the Container registry.

Table 4-1 - Github Package Manager Permission schema

Permission	Access D
Read	Can download package. Can read package metadata
Write	Can upload and download this package Can read and write package metadata
Admin	Can upload, download, delete, and manage this package. Can read and write package metadata. Can grant package permissions.

The main repository page, presented in Figure 4-7, displays a list of all available containers within the repository. For organizations, an additional tab is provided, showing containers across the organization based on the user's access level. Furthermore, administrators can configure granular access and visibility settings for containers through the Repository's or Organization's Settings Tab, as shown in Figure 4-8. GitHub employs an inheritance model, with the Organization level serving as the apex of this hierarchy.

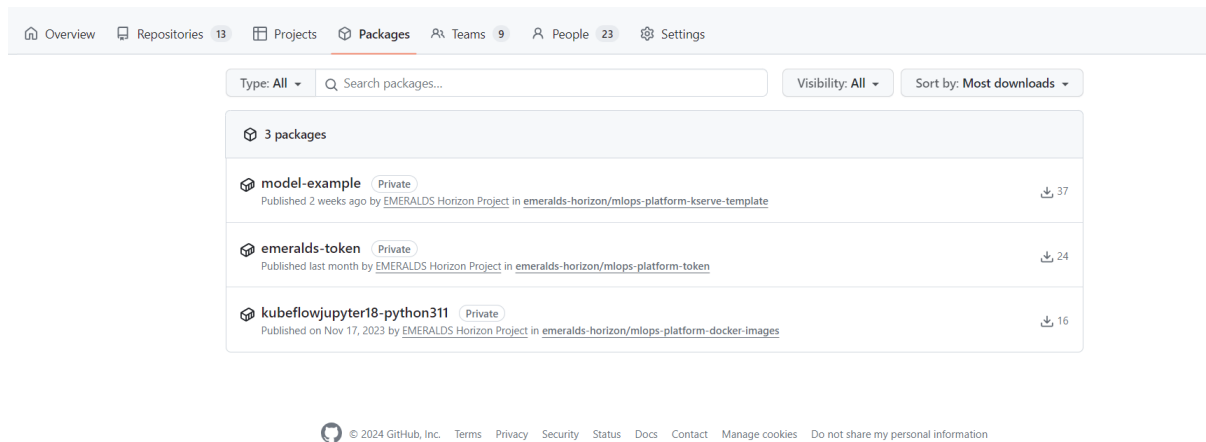


Figure 4-7 - Github Packages

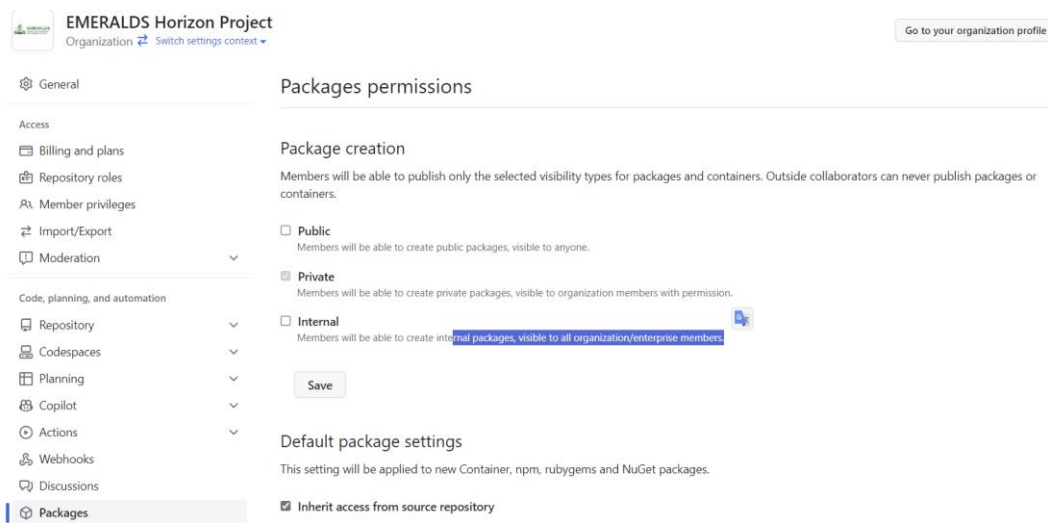


Figure 4-8 - Manage Access Level

For containers marked as private, User may download them through the available API. In this case User should be authenticated using personal access token.

4.3 Continuous Deployment

Continuous Deployment (CD) is a software development practice where code changes are automatically deployed to execution environments without manual intervention. A similar term named **Continuous Delivery** is also used widely in the same context. There is a lot of controversy regarding the difference between the “deployment” and the “delivery” term, with the most common states that the “deployment” is a fully automated process of deploying the changes to the production system, while the “delivery” includes a manual step, where the final deployment to the production is triggered by the operation team.

Within Continuous Deployment practices, a range of deployment environments are commonly employed to ensure the reliability and quality of software releases before they reach production. These environments include testing, staging and production and in each environment, a unique array of testing suites is utilized to validate the integrity and quality of the code. The level of automation of those testing suites defines the actual maturity of the entire pipeline.

Regarding the EMERALDS project, the CD process as a practice is valid and followed by both ML and non-ML emerald Services. Generally, both types of emerald services are stored in the same Container Registry in the form of Docker containers and can be deployed on hosts supporting containerization. Slight differences might exist related to ML Services, as they could benefit from the use of GPUs during execution.

4.3.1 CD Platform

As discussed, part of the CD Pipeline is the execution environment that shall be the final deployment destination for the emerald Services. The following list presents the primary requirements derived from grouping of emerald Service according to their architecture and technology stack utilized in the development. The requirements are focused on production systems rather than staging or testing.

1. Edge Devices support

This is one of the primary objectives of the EMERALDS Project. The management of edge devices and the corresponding network is a complex endeavor, as it needs to consider the quality of services offered, the security of the network and the operational costs. It is essential that the emerald Services are readily deployable and capable of receiving over-the-air updates in an automated and timely fashion. Additionally, the service orchestration platform should provide lightweight agents for the Edge node in terms of hardware requirements.

2. Batch and Stream data processing

The management platforms should be able to consume data both in the form of batch and as streams. This requirement is more bound to the emerald services, rather than the operational platforms. The latter should offer auxiliary services such as message brokers or Pub/Subs, storage services and data warehouses.

3. Containerization

As it has already been analyzed in section 2 of this report, it will be pursued to offer all emerald Services as containers. This approach allows for packaging each service in a single object for distribution and easy integration to the state of art orchestration platforms, such as Kubernetes and its extensions.

4. Apache SPARK ³⁶

To effectively support the processing of extreme scale data, it is imperative to utilize distributed systems capable of horizontal scaling. Apache Spark is such a framework, upon which the emerald Services of Task 3.2 are built.

5. Dynamic Scaling and Resource Management

To successfully cope with the process of extreme scale data, the orchestrator should be able to scale dynamically based on the load and the availability of edge devices and their computational resources.

Load distribution or migration to available nodes, edge computation, small hardware footprint of the management framework are just a few of the characteristics of an efficient orchestrator.

6. Task Scheduling

As task scheduling, we define the algorithm responsible for determining the deployment strategy of each emerald Service to suitable nodes. This decision is made considering factors such as the hardware capabilities of the target nodes, the hardware prerequisites of the service, and Quality of Service (QoS) parameters like response latency, request throughput, and privacy considerations. Ideally, the scheduling algorithm should dynamically adapt to changes in these factors and adjust its behavior accordingly.

³⁶ <https://spark.apache.org/>

7. Monitoring and logging

Monitoring and logging hold crucial roles in ensuring the reliability, performance, and security of edge devices and associated services. It facilitates troubleshooting for emerald Services, while gathering performance metrics provides the necessary means for the KPI calculation of the toolset.

8. Upgrade and rollback mechanism.

The ability to support automated upgrades for the new version of the software released is of crucial importance to provide uninterrupted service for the end user. In a similar manner, rollback back to the previous stable state, mitigating disruptions is a failsafe mechanism for the stability of the system.

9. Network Flexibility

An important requirement for service orchestration platforms that are to be used within the Compute Continuum is to allow for network connectivity between nodes that reside on different local area networks. Access to public endpoints or NAT techniques might not always be available.

10. Federated Learning

Adopting a Continuous Deployment process is an essential requirement for establishing a successful Federated Learning network across edge devices and cloud services. The process allows organizations to efficiently manage and deploy machine learning models across distributed edge environments, ensuring seamless updates, performance optimization, and scalability of decentralized learning systems. This feature has been further investigated in “*Deliverable 4.1 - Mobility Data Analytics and Learning Services v1*”.

Several platforms were evaluated for hosting the continuous delivery (CD) of the emerald Services. The selection criteria included the extent to which each platform fulfilled the predefined requirements outlined at the beginning of this section, along with considerations of the tool's maturity, adoption within the DevOps community, and whether it is open source with a permissive license. Kubernetes³⁷, K3s³⁸, MicroK8s³⁹ and were chosen for further investigation, while other tools such as Docker Swarm⁴⁰, Eclipse ioFog⁴¹, OpenYurt⁴², SuperEdge⁴³, and Open Horizon⁴⁴ were excluded from further analysis either because they have become obsolete, as is the case with Docker Swarm, or of lower popularity compared to the selected options. The popularity of each tool was assessed based on metrics such as stars and forks in their respective GitHub repositories.ⁱⁱ

Kubernetes, often abbreviated as K8s, is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications and it is considered the leading technology in the cloud-native ecosystem. Two of its main components are the Control Plane, which acts as the brain of a Kubernetes Cluster, and the workers node, which are responsible for running the actual workloads.

The primary challenge in integrating remote workers located on edge nodes into a Kubernetes cluster lies in ensuring uninterrupted communication between the control plane and worker nodes.

³⁷ <https://kubernetes.io/>

³⁸ <https://k3s.io/>

³⁹ <https://microk8s.io/>

⁴⁰ <https://docs.docker.com/engine/swarm/>

⁴¹ <https://iofog.org/>

⁴² <https://openyurt.io/>

⁴³ <https://superedge.io/>

⁴⁴ <https://open-horizon.github.io/>

One workaround involves establishing a VPN network connecting the different nodes. The VPN module, developed as part of Task 2.5, is a perfect fit to address this issue. Obviously, such a solution brings in an extra configuration overhead that will not scale effectively in solutions with thousands of edge nodes.

Another consideration of using Kubernetes on edge networks is its hardware requirements. While Kubernetes is often perceived as resource-intensive, the actual hardware footprint of a K8s worker node is primarily determined by the workload it needs to execute, rather than the management overhead of the cluster.

In the context of edge computing, service **orchestration** implies service management and deployment across the available distributed execution environment. Kubernetes has introduced a dedicated control plane component that executes the service scheduling in a two-step procedure. As a first step, the algorithm **filters** out all available worker nodes with sufficient resources based on specific service resource requirements, as being described in the Kubernetes manifest files. The second step is named **scoring**, and in this case the scheduler assigns a score to each node that survived filtering. The nodes with the highest score are selected for running the workload. There are multiple plugins that can be installed and configured to further customize the scheduling procedure, and additionally there is a possibility to configure a static approach by using *nodeSelectors*, *node affinity* and *Taint/Tolerance* to label the nodes for selection. However, there are some deficiencies to be addressed for the scheduling mechanism to be utilized within the edge environment, primarily the inability to specify custom parameters, such as *latency*, *throughput*, and *privacy*. A solution to this problem would be the creation of a separated Service that can also be deployed within the Kubernetes cluster and take into account the variability of the QoS parameters defined previously and engage the scheduler to dynamically distribute the workloads to the appropriate edge nodes.

K3s is a lightweight, fully compliant Kubernetes distribution focused on running in constrained devices, bearing a much lower memory footprint than other available K8s distributions. It is also optimized for several CPU architectures, such as ARM32, ARM64, and ARMv7, and with a binary size of less than 100 MB, it has become the preferable for edge environments based on Raspberry Pis⁴⁵ or NVIDIA Jetson⁴⁶ boards. Its binary contains all Kubernetes control plane components encapsulated within a single process, while the number of external dependencies is minimized. Moreover, Rancher⁴⁷, the company behind the initial development of the framework, also delivers an OS for nodes optimized for running K3s, the K3OS⁴⁸.

Conversely, the same features that make it ideal candidate for edge device, limit its suitability for demanding workloads running on more powerful servers and cloud infrastructure. In this scenario, the default K8s cluster is the preferred solution, as it offers better scalability and resource efficiency. It is worth mentioning that K3s may match its performance to K8s, with additional configuration such as the replacement of integrated SQLite⁴⁹ database with etcd⁵⁰, with obvious impact on hardware footprint and installation complexity.

Another benefit of K3s is the ability to include devices without public IP addresses. This is achieved with the initiation of the registration process from the worker node – or agent node according to K3s nomenclature – and the establishment of WebSocket tunnel with the controller node. The data traffic between the agent node and the controller can then be exchanged securely through this newly established bidirectional link.

⁴⁵ <https://www.raspberrypi.org/>

⁴⁶ <https://www.nvidia.com/en-eu/autonomous-machines/embedded-systems/>

⁴⁷ <https://www.rancher.com/>

⁴⁸ <https://k3os.io/>

⁴⁹ <https://www.sqlite.org/>

⁵⁰ <https://etcd.io/>

Microk8s is a similar platform with K3s, focusing on a lightweight k8s cluster for small servers and edge devices. Compared to K3s, it has a bigger memory footprint, but it has the advantage of being one of the most easily customizable distributions via the installation of add-ons through simple one liner commands. The available add-ons include some of Kubernetes' most popular modules, including Helm⁵¹ and Istio⁵². Additionally, it offers a straightforward path to achieving Kubernetes High Availability features.

Last, **KubeEdge**⁵³ is also an open-source platform, build on top of Kubernetes as extension, with main goal to support distributed clusters across the compute-continuum. Like K3s and MicroK8s, KubeEdge is flexible to support the edge heterogeneity and offer the same approach on worker's node communication with the control plane. Its main difference comes from the fact that while K3s is a strip down version of Kubernetes, the KubeEdge is a framework specifically adapted to the edge that follows Kubernetes principles without reusing the same components. In fact, in order to realize the cloud-to-edge continuum based on K3s, the use of multi-cluster K8s-like clusters is required, a condition that does not hold true with KubeEdge. Figure 4-9(a) presents the K3s deployment approach, while Figure 4-9(b) depicts the KubeEdge main architecture.

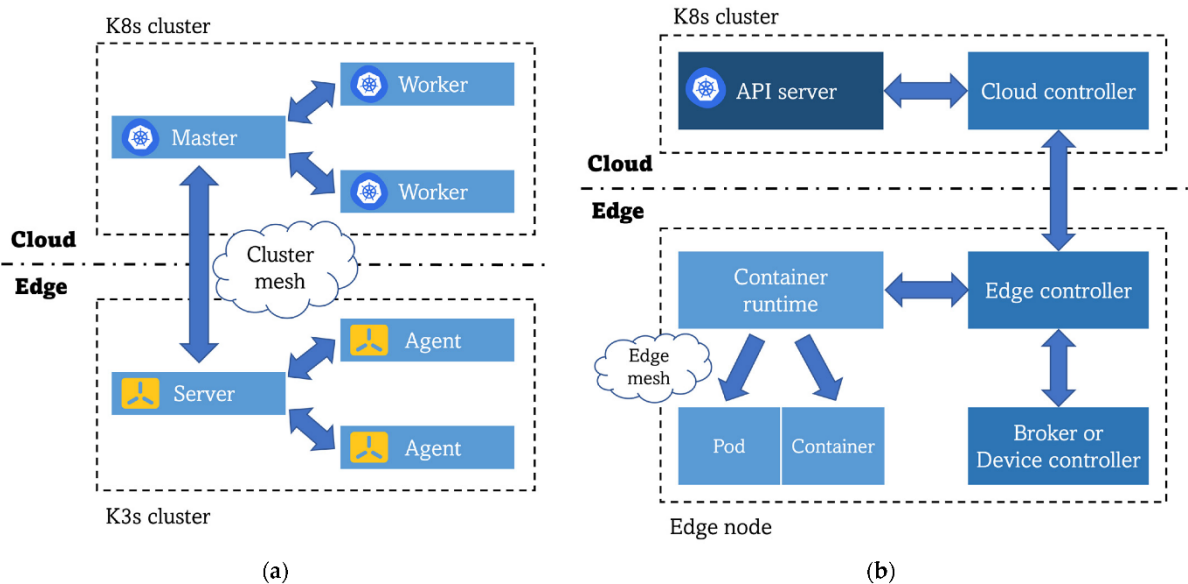


Figure 4-9 - KubeEdge Installation Schema

A useful addition of KubeEdge, is the out-of-the-box support of MQTT⁵⁴ protocol, a lightweight messaging protocol which is the norm for IoT devices and machine-to-machine communication. Leveraging the synergy between edge and cloud facilitated by KubeEdge, **Sedna**⁵⁵, an edge-cloud synergy AI project, can enable collaborative training and inference capabilities across edge-cloud environments. This includes features such as joint inference, incremental learning, federated learning, and lifelong learning. Sedna is another possible candidate for the orchestration of the Federated Learning emerald Services of Task 4.2, which is further investigated in D4.1 and D4.2.

All frameworks that have been listed are following the same principles regarding the scheduling algorithm, which is expected as all of them are derived from Kubernetes.

⁵¹ <https://helm.sh/>

⁵² <https://istio.io/>

⁵³ <https://kubedge.io/>

⁵⁴ <https://mqtt.org/>

⁵⁵ <https://github.com/kubedge/sedna>

Table 4-2 - Kubernetes Variants

	Kubernetes	K3s	MicroK8s	KubeEdge
Edge Devices support	No	Yes	Yes	Yes
Batch and Stream data processing	Install 3 rd party app	Install 3 rd party app	Install 3 rd party app	Natively
Apache SPARK	Install 3 rd party app	Configuration Is needed	Install 3 rd party app	N/A
Upgrade and rollback mechanism	Yes	Yes	Yes	Yes
Network Flexibility	No	Yes	Yes	Yes
Federated Learning	Install 3 rd party app	Install 3 rd party app	Install 3 rd party app	Natively

For purposes of CD platform, the use of MicroK8s shall be used in parallel with KubeEdge. This approach should be sufficient to allow the project to execute demanding workloads on Fog/Cloud servers, such as Spark tasks, taking advantage of the edge features offered by KubeEdge, while keeping the installation complexity relatively low and maximizing the resource utilization.

4.3.2 Platform Deployment and Configuration

Based on DevOps practices, the deployment of the CD platform will adhere to the principle of Infrastructure as Code (IaC). This entails automating all tasks related to software provisioning, configuration management, and application deployment using configuration management tools, such as Ansible⁵⁶.

Ansible is a popular choice for managing configuration across edge devices due to its lightweight agentless architecture and support for automating tasks through simple YAML playbooks. Ansible's ease of use and minimal resource requirements make it well-suited for managing diverse and distributed edge environments.

The ansible playbooks to be used for the deployment and configuration of the CD Platform can be found on Github emeralds-horizon organization, under [Emeralds Hosting Services](#) repository. The scripts will be applicable not only during testing and demonstration phases but also in production scenarios. As such, the scripts are creating a form of a template for creating a hosting environment for the emeralds Toolset, which is called **Emeralds Hosting Services**. **End Users may clone the repository and, by supplying the necessary hardware resources, effortlessly set up a platform to host the emeralds along with any other services they might have.**

The primary objectives of utilizing Ansible within the EMERALDS Project include installation and configuration of container's engine and containers management frameworks – specifically Docker and Kubernetes respectively - on Cloud Nodes and to deploy the KubeEdge framework across both cloud and edge nodes.

The provisioning of the initial operating system in the servers depends on the environment that will host them. The SoTA approach is the use of Virtual Machines either on private or public cloud infrastructure. For the EMERALDS Project, the use of the Azure Public Cloud Provider has been selected and the required resources have been created through Azure Portal. To further automate the process and make it re-usable we are planning to deploy the Virtual Machines and the required

⁵⁶ <https://www.ansible.com/>

network resources with the use of tools like Terraform⁵⁷ and OpenTofu⁵⁸. Both tools are supporting the IaC approach and allow the system administrators to redeploy or scale the CD Platform as will. Furthermore, both tools can be used with minor changes for deploying the platform on different cloud providers such as AWS, GCP or even on selected private cloud hypervisors such as KVM and Proxmox⁵⁹.

For edge nodes, the OS installation depends on the type of these devices. For example, physical machines such as Single Board Computers require manual actions while Virtual Machines depends on the underlying hypervisor. For demonstration purposes, the creation of micro-VMs on Azure Cloud Provider and the use of Single Board Computers, such as Raspberry PI, shall be used.

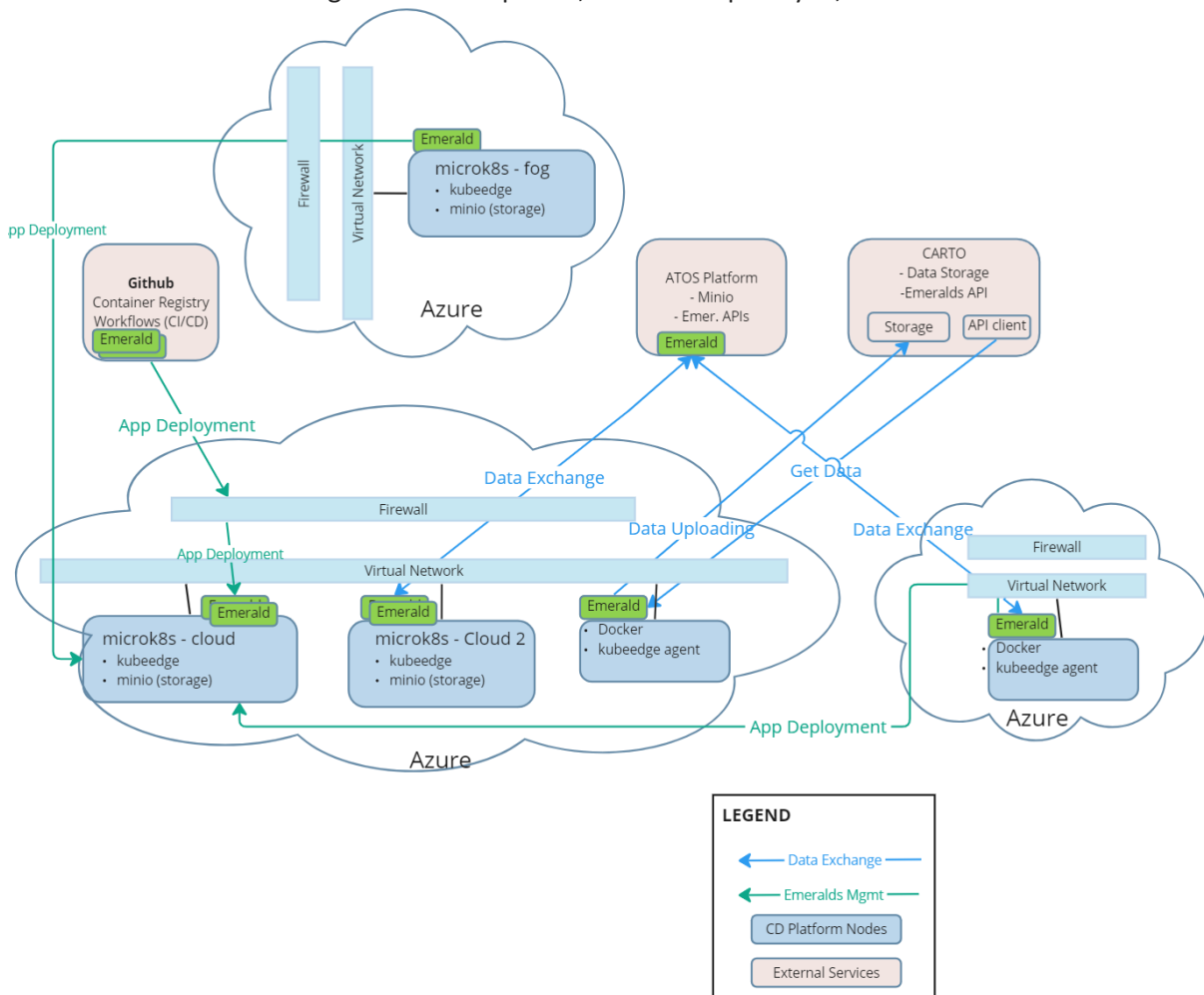


Figure 4-10 - CD Platform Resource Diagram

Figure 4-10 provides a high-level representation of the different resource groups that will be deployed on Azure Cloud Provider.

⁵⁷ <https://www.terraform.io/>

⁵⁸ <https://opentofu.org/>

⁵⁹ <https://www.proxmox.com/en/>

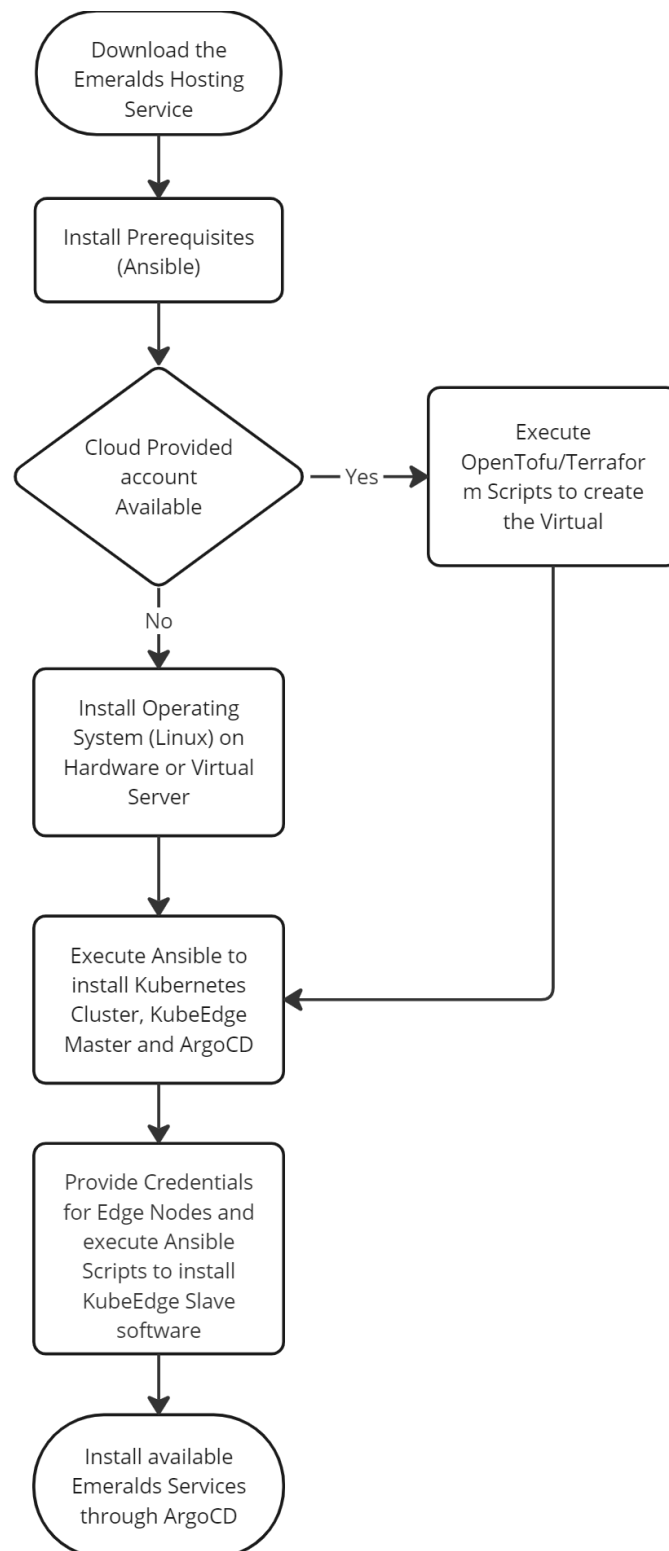


Figure 4-11 - emeralds Hosting Service Installation Flow

Figure 4-11 presents the installation flow of the emeralds Hosting Service. The process begins with cloning the repository from the EMERALDS Toolset portal. Users need to provide a Linux server with network access to the public internet and install the Ansible software.

The next step depends on the availability of cloud servers to host the emeralds services. If the user has access to public cloud provider infrastructure, the repository contains the necessary OpenTofu scripts to automatically create the cloud resources—virtual machines, network, and firewall rules. These scripts are a work in progress and shall be available as part of the “D2.3– Containerized EMERALDS Toolset v2”. If not, the user must install a Linux operating system on the required servers. The operating system installation process is outside the scope of this task as it depends on multiple parameters such as available hardware for servers and networks and cannot be automated for all possible cases.

Once the servers are available, the user must execute simple Ansible commands to install the Kubernetes cluster, the KubeEdge master node, and ArgoCD. Similarly, the user must provide the necessary information (location and credentials) for any edge nodes in the solution. The provided Ansible scripts will install the KubeEdge slave software on these nodes. The final step is the installation of the required emerald services in the cluster through the ArgoCD portal. All emerald services are available in the portal as configured in the previous step. The user needs to select the required emerald services and enable the sync process. After this step, the selected services are automatically deployed and monitored for any future upgrades. Users can also use the web portal to sync additional emerald services or remove those that are no longer needed.

All steps are documented in a text file available from the Emeralds Hosting Services repository. Users only need to provide information regarding the IP addresses of the available servers and the necessary credentials for accessing these servers or the public cloud provider’s account. A basic understanding of Linux shell commands and basic networking is required for the user.

4.3.3 Deployment of Emerald Services on the CD Platform

For the deployment of the new versions of emerald Services on the CD platform, the use of **Argo CD**⁶⁰ tool has been selected. Argo CD is an open source, Kubernetes native CD tool. It is based on **GitOps principles** to manage the deployment of applications. The tool continuously monitors changes in the repositories and automatically reconciling the desired state with the actual one running in the Kubernetes cluster.

Argo CD and generally GitOps considers the code repository as the **source of truth** for the application definitions. It is a declarative tool, meaning that the desired state of the application is declared in Kubernetes manifest files stored in Git. Four basic features of the tool are its rollback capabilities, which can be combined with health monitoring of the application, the extensive integration with multiple Git providers and the existence of user-friendly Web User Interface and Command Line Interface.

Figure 4-12 presents the CD workflow to be implemented on top of Argo CD. Development teams are pushing their latest source code modifications into the source code repository. This action triggers the “Github Action”, which runs the CI Pipeline and eventually build and pushes the final container image into the Github Container Repo. In the CD testing environment, the ArgoCD has been installed and configured to watch for any changes in the Github repository of choice. Whenever such changes are being observed, ArgoCD is downloading the latest changes and applies the Kubernetes manifest files into the testing environment, which eventually updates the installed application.

⁶⁰ <https://argo-cd.readthedocs.io/en/stable/>

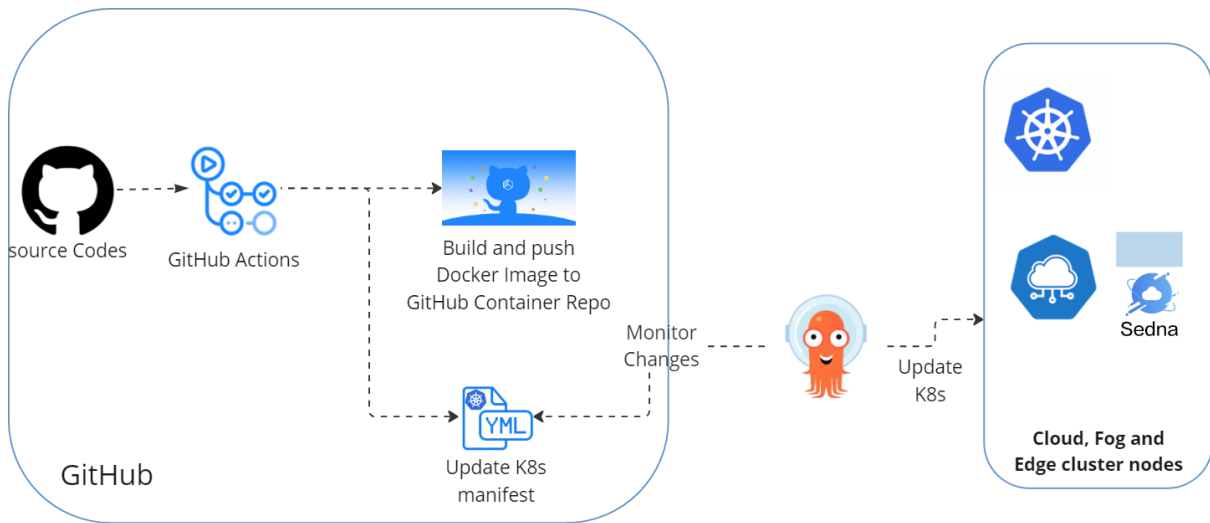


Figure 4-12 - Argo CD Deployment flow

As an example of the process the following list presents the github repositories that contains the required code for the emerald “Extreme-scale stream processing orchestrator” for each steps along with a short description of each task.

1. CI Pipeline

Repo: [Extreme-scale stream processing orchestrator github actions](#)

Description: Execute automation test, static code analysis, build container image and push it to Container Registry.

2. Kubernetes Manifest files

Repo: [Extreme-scale stream processing orchestrator Kubernetes Configuration files](#)

Description: Kubernetes configuration files for installing the container image and create the necessary

3. ArgoCD Installation scripts

Repo: [Emeralds-hosting-services ansible scripts](#)

Description: Ansible Scripts that are used for the installation and configuration of the ArgoCD Tool. All emerald Services shall be available on the CD Platform and synchronised automatically with their repository. The execution of these scripts requires a Linux Server with Ansible application installed.

4. Kubernetes Basic Configuration Scripts

Repo: [Emeralds-hosting-services Kubernetes Configuration files](#)

Description: These are generic configuration files for the Kubernetes cluster that apply for all emeralds services. It contains installation instructions about the Ingress Controller which is a Kubernetes network resource that acts as a load balancer, and Kubernetes Volume resources, which are used as storage for files or folder. As a next step additional resources are added such as specific databases images – MobilityDB, Minio.

5 Integration with Analytics as a Service Platforms

Within the context of EMERALDS project, two established MAaaS platforms are foreseen as maturity enhancing checkpoints for the overall EMERALDS technologies developed in WP3 and WP4 and bundled under T2.1 into the EMERALDS toolset.

ATOS is offering a Mobility AlaaS platform capable of training and inferencing machine learning and artificial intelligence models. CARTO's analytics cloud platform is one of the pioneers in the domain and a key player in the geo-spatial analysis through its cloud based offer.

EMERALDS provides a versatile collection of specialised software modules containing containerised versions of the tools and software stacks as shown in Section 3 and their bundling is the focal point of T2.1. Following successful integration of the tools in the EMERALDS MAaaS toolset, a dual-focus deployment and integration with two cloud-based platforms addressing different niche user bases is performed: in one case, urban data scientists and users with strong technical backgrounds (T4.3, ATOS open research environment fully compatible with the EU AI on Demand ecosystem, fostering the reproducibility of method advancements and open access of innovative technologies reported in D4.1); in the other case, needs and extending functionalities (extreme scale cloud native data management, heavy caching visualizations and corresponding dashboards) of broader user bases (T2.3, in CARTO established Geospatial Analytics as a Service commercial environment reported in D2.4). The ATOS Mobility AI as a Service (Mobility AlaaS) platform offers several key benefits: a) interoperability between computed or ingested data through synergies between tools at all levels, b) seamless data transfer from processing to analytics to visualizations, and c) the ability for stakeholders, mobility operators, and end users to configure models and tools according to their workflow requirements. Consequently, users can explore analytics services across all time horizons (hindsight, insight, foresight) and receive support in time-critical operations and decision-making.

Conversely, the CARTO Geospatial Analytics as a Service platform presents an easy-to-use version of the developed tools, targeting a broader audience. Initially, it unlocks ML analytics processes in the CARTO Analytics Toolbox, offering solutions for new types of users who need to solve challenges intuitively and rapidly with a low entry barrier. Existing analytics users are familiarized with spatial analytics using technologies and interfaces they are accustomed to, rather than creating a specialized system for spatial experts like current state-of-the-art solutions.

As an endpoint of the EMERALDS architecture, the CARTO platform houses data visualizations, dashboards, and VA tools, all controlled by user-formed queries. In this sense, users can interact with and leverage the full potential of the analytics tools seamlessly.

In this section, a description of the fundamental integration and deployment processes of the EMERALDS toolset with the ATOS and CARTO platforms is explained.

5.1 ATOS Integration

The ATOS Mobility AlaaS platform (presented in D4.1) is engineered to support the development, deployment, and scaling of machine learning models specifically designed for urban mobility applications. As outlined in D2.1 - EMERALDS Reference Architecture, the platform serves as a foundation for the MLOps process but does not directly offer Analytics as a Service. Instead, it provides a variety of tools and frameworks that enable Project Partners to focus on delivering real value through their algorithms to the Emerald services they are developing, rather than focusing on the underlying infrastructure requirements.

To aid in the development and inference processes of the WP4 Emerald Services, the platform includes the following features:

1. Data Management Integration

The platform features two storage services, Minio⁶¹ and MobilityDB, which are equipped with robust authentication and authorization services based on Keycloak⁶². These storage solutions support the training of models by storing both raw and processed datasets from the project's use case scenarios. They also facilitate data exchange during both the development phase and production, enabling integration with external analytic services, such as those provided by CARTO's platform.

2. Deployment and Scaling

The platform is based on SotA MLOps open-source projects to ensure hassle-free deployment of the generated models, while provides reliable scalability mechanism using Kubernetes cluster⁶³.

3. Machine Learning Models Development and Federated Learning

The development framework within the Mobility AlaaS platform is used to design, test, and refine machine learning models. For Emeralds of WP4 that benefit from federated learning, such as models that require data privacy preservation or models that operate in decentralized environments, the Mobility AlaaS platform's federated learning module provides the necessary infrastructure.

4. Dashboards and Virtual Analytics

The Mobility AlaaS Platform is offering JupyterHub⁶⁴, a version of the Jupyter server hosted on Cloud Infrastructure, accessible to the used through a web browser. JupyterHub is a multi-user version of the notebook, that spawns, manages, and proxies multiple instances of the single-user Jupyter notebook server.

As Jupyter can support out-of-box various data visualizations libraries in Python, it can be used to create high-quality static, interactive or animated visualizations. This capability makes Jupyter Notebooks an excellent tool for exploratory data analysis, where visualizations play a crucial role in understanding data. The downside of these approach is that the use of the tool requires programming knowledge - typically in Python and its libraries, and performance - as it is more oriented for small to medium size data tasks.

Further details about the ATOS Mobility AlaaS platform are available in Chapter 4 of ***"Deliverable D4.1 — Mobility Data Analytics and Learning Services V1"***.

5.2 CARTO Integration

CARTO is a leading platform that specializes in location intelligence and spatial data analysis, offering robust solutions for visualizing, analysing, and interpreting geographic data. This integration facilitates enhanced decision-making processes and insights for projects requiring detailed spatial analysis. Among its standout features are **interactive data visualization**, which enables the creation of dynamic maps that clearly present complex spatial data. The platform also boasts a **complete data management** framework that supports a standard SQL interface, ensuring scalability and data

⁶¹ <https://min.io/>

⁶² <https://www.keycloak.org/>

⁶³ <https://kubernetes.io/>

⁶⁴ <https://jupyter.org/hub>

integrity. Further, CARTO'S **advanced spatial data analysis** allows users to conduct detailed spatial queries, utilize sophisticated GIS capabilities, and identify geographic patterns. For developers, CARTO offers a **developer toolset**, including a frequently updated suite of APIs and SDKs, enabling the creation of custom applications and the extension of the platform's existing functionalities. Additionally, as a **cloud-native solution**, CARTO provides a scalable, high-performance infrastructure capable of hosting large datasets and handling complex analytical tasks without sacrificing responsiveness.

Within the framework of the EMERALDS Project, CARTO Platform facilitates integration with the newly developed Emerald Services in the following ways:

- REST API Integration: REST Clients can remotely trigger and exchange data with Emerald Services that function as REST API Servers, enabling seamless interactions and data flows.
- Remote Data Access: CARTO platform may direct access to generated datasets stored on external storage facilities, such as the PostgreSQL and PostGIS instance in Atos Mobility AlaaS platform.
- Data Storage and Analysis: Emerald Services can directly utilize CARTO'S storage facilities to dump generated data. Subsequently, CARTO'S Analytics Toolbox can be utilized to analyse this data and extract valuable insights. An example of this approach is detailed in the blog post "Analysing Mobility Hotspots with MovingPandas" available at <https://carto.com/blog/analyzing-mobility-hotspots-with-movingpandas>.

Further details about the CARTO Platform and deployment of emeralds services are provided as part of **Deliverable D2.4 — Demonstration of integrated services (EMERALDS) V1**

6 Conclusions and next steps

In this deliverable, we summarize the critical advancements and concepts developed throughout this project, focusing on the integration and implementation of key technologies within EMERALDS.

Currently in its first version, the EMERALDS toolset sets the foundation for a robust and scalable analytics framework. Future updates will build on this foundation, introducing new features and enhancements to address the evolving needs of urban mobility analysis. As a living software repository, the EMERALDS toolset is continuously updated and refined based on user feedback and technological advancements. This iterative development process ensures that the toolset remains at the forefront of urban mobility analytics, offering users the latest capabilities and improvements.

We detailed the process and the benefits of containerized EMERALD services, emphasizing the versatility and efficiency brought about by this approach. Containerization ensures consistent environments across different stages of development, reducing the risks associated with configuration drift and deployment errors.

Next, we focused on establishing CI/CD pipelines as a crucial step towards automation and streamlining of the development lifecycle. We explored the integration of security practices early in the development phases and the utilization of modern tools and platforms like GitHub Actions, Kubernetes Cluster and remote management tools for edge nodes, such as KubeEdge.

We provided a strategic overview of integrating diverse services and applications. We listed the key architectural considerations necessary to ensure seamless interaction between different components of the EMERALDS project. This integration is crucial for supporting the complex data workflows that will be implemented as part of WP5. Last we provided the main points of integration with external Analytics as a Service platforms. Specifically, we analysed how platforms like ATOS Mobility AlaaS and Carto can enhance the capabilities of EMERALDS services through advanced data management, processing, and visualization tools.

Covering the period M19-M36, the focus will be on concluding the CD infrastructure by providing additional services such as central load balancer and storage layer, while improving the installation automation process for public cloud providers. Furthermore, to fully support the Emerald Toolkit, the Emeralds Hosting Service shall be enhanced to include ML frameworks which will be able to facilitate the inference process of the Emeralds Services developed under WP4. Last the integration of the security emeralds developed under “Task 2.3 Security & Data Governance” with the CD platform will also be part of the “Deliverable D2.3 – Containerized EMERALDS Toolset v2” to support a secure and trusted environment for the Emeralds Ecosystem.

Annex 1: Ethics Checklist and Questionnaire

A. PERSONAL DATA

1. Are **personal data** going to be processed for the completion of this deliverable?
- If “yes”, do they refer only to individuals connected to project partners or to third parties as well?

No

2. Are “**special categories of personal data**” going to be processed for this deliverable? (whereby these include personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, and trade union membership, as well as, genetic data, biometric data, data concerning health or data concerning a natural person's sex life or sexual orientation)

No

3. Has the **consent** of the individuals concerned been acquired prior to the processing of their personal data?
- If “yes”, is it based on the Project’s Informed Consent Form, either on the provided Template or on other attached herein Template?
 - If “no” is it based on a different legal basis?

N/A

4. In the event of processing of personal data, is the processing:
- obviously “**Fair and lawful**”, meaning executed in a fair manner and following consent of the individuals concerned or based on another - acknowledged as adequate and proportionate as per above - legal basis?
 - Performed for a **specific (project-related) cause** only?
 - Executed on the basis of the principle of **proportionality and data minimisation** (meaning that only data that are **necessary** for the processing purposes are being processed and such deductive reasoning is documented)?
 - Based on **high-quality, updated and precise personal data**?

N/A

5. Are there any provisions for a storage limitation period of the personal data-in case of storage- after which they must be erased?

N/A

6. Are all **other lawful requirements** for the processing of the data (for example, **notification of the competent Data Protection Authority(s)** or undergoing a **DPIA procedure** and consulting with the competent DPA, if and where applicable) adhered to and on what legislative basis are such notifications justified as necessary or dismissed as unnecessary?

N/A

7. Have individuals been **made aware of their rights** on the processing of the personal data as per the GDPR and the relevant and executive national legislation (particularly the rights to access, rectify and delete the personal data and their right to lodge a complaint with the relevant Competent Authority) and if yes, by what demonstrable means (e.g. the informed consent form as per above or as per other Templates, attached herein?)

N/A

8. Even if anonymized or pseudonymized or aggregated data are referred to, does the dataset contain **location data** that could potentially (even via the combined use of other datasets) be **traced back to individuals**? If yes, what specific measures are taken to ensure this data (i) is anonymized or pseudonymized and (ii) cannot be used to track individuals without their consent? If no, what is the scientific methodology used to collect and gather said data?

N/A

9. In the context of risk assessment, prediction and forecasting, as foreseen in the scope of the EMERALDS project, during traffic, population movement monitoring or weather events, **is there any risk** that personal data could be inadvertently revealed in the event of an **emergency** or **unusual event**, because of the dataset usage, either on its own or combined with other openly available datasets, triggering identification or unwanted disclosure of PII? What measures are in place to protect - still identifiable if the dataset allows such extraction - **personal data** in these circumstances?

NO

10. For the use case of Trip Characteristics Inference as per the EMERALDS project scope, are there specific measures to ensure that **inferences made about trip characteristics** cannot be linked back to **specific individuals** or reveal **sensitive information** about their **habits** or **routines** i.e. by identifying specific individuals' absence or presence routines whether in the home or in a professional environment or in other premises?

NO

11. Are there any potentially personal identifiable information (PII) in the datasets, **disclosable by combination with other datasets**, either open data or proprietary (e.g., E-tickets validation data)? If yes, how is PII adequately anonymized or pseudonymized or how other datasets that by combination may result in unwanted or illegal disclosures or identification before any processing takes place?

NO

B. DATA SECURITY

1. Have proportionate security measures been undertaken for protection of the data, taking into account project requirements and the nature of the data?
- If yes, brief description of such measures (including physical-world measures, if any)

- If yes, is there a data breach notification policy in place within your organization (including an Incident Response Plan to such a breach)?

NO. At the moment the platforms described in the document do not store any data.

2. Given the **large-scale nature** of some datasets, are there specific measures in place to protect **included personal data** at scale at the data source or in the possession of data processors?

N/A

3. Regardless of personal data, in the case of Multi-modal integrated traffic management as defined under the EMERALDS scope, are there specific measures in place to ensure **the availability and integrity of data spanning multiple modes of transport** from being disclosed in other manners than the ones intended and covered under an open data scheme?

N/A

4. Are there specific measures in place to secure **sensitive infrastructure data**, if present?

YES. Access tokens for private repositories and access credentials for remote servers shall be stored on secure and encrypted storage facilities.

C. DATA TRANSFERS

1. Are personal data transfers beyond project partners going to take place for this deliverable?

- If “yes”, do these include transfers to third (non-EU) countries and if what policies apply?

NO

2. Are personal data transfers to public authorities going to take place for this deliverable?

NO

3. Do any state authorities have direct or indirect access to personal data processed for this deliverable?

NO

3. Taking into account that the Project Coordinator is the “controller” of the processing and that all other project partners involved in this deliverable are “processors” within the same contexts, are there any other personal data processing roles further attributed to any third parties for this deliverable? And if any, do they conform to the GDPR provisions?

NO

4. Given the geographical diversity of the datasets, are there measures in place to ensure compliance with specific personal data protection regulations **in different jurisdictions** i.e. at the place of the data source establishment as well as at the place of the establishment of a Data processor?

N/A

5. Are there additional protocols for data transfers involving **sensitive infrastructure data**, if present?

N/A

D. ETHICS AND RELATED ISSUES

1. Are personal data of children going to be processed for this deliverable (ie. “underage” signified e-tickets)?

NO

2. Is **profiling** of identifiable individuals in any way enabled or facilitated for this deliverable?

NO

3. Are **automated decisions** for identifiable individuals made or enabled on the basis this deliverable?

NO

4. Have partners for this deliverable taken into consideration system architectures of **privacy by design** and/or **privacy by default**, as appropriate?

YES

5. Have partners for this deliverable taken into consideration gender equality policies or is there an explicit reasoning that dismisses such risk as unsubstantiated or such need as irrelevant as per the methodology of work and production of the deliverable?

YES

6. Have partners for this deliverable taken into consideration means of protecting the confidentiality of the dataset if it is not signified as open data?

N/A

7. Are there additional considerations around the collection and processing of **location data** and data **that could potentially be used to infer patterns about individuals’ movements**?

NO

8. Have partners identified any **additional ethical issues** related to the processing of sensitive infrastructure data?

NO

9. Are shared economy (ie. “Uber” transfer services or “Lime” Scooters or other solution) or other shared mobility infrastructures used by the data sources? If yes, are there measures in place to ensure that the processing of **shared mobility data** respects privacy rights?

NO

10. In the context of Traffic Flow Data Analytics, are there specific considerations to ensure that the **analysis of traffic flow data** does not infringe on privacy rights or reveal sensitive information about individuals’ movements or routines?

N/A

11. Is the Project taking into account the need for an all people-inclusive policy in the future within its overall goals and not only the “tech-savvy” (i.e. elderly people not familiar with some tech devices) and does it entail possible proposals for that?

YES

References

ⁱ Roman, Dumitru, Radu Prodan, Nikolay Nikolov, Ahmet Soylu, Mihhail Matskin, Andrea Marrella, Dragi Kimovski et al. "Big data pipelines on the computing continuum: tapping the dark data." *Computer* 55, no. 11 (2022): 74-84

ⁱⁱ <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10143384/>

Vaño, R., Lacalle, I., Sowiński, P., S-Julián, R., & Palau, C. E. (2023). Cloud-native workload orchestration at the edge: A deployment review and future directions. *Sensors*, 23(4), 2215. Accessed on 15 May 2024 from: <https://www.mdpi.com/1424-8220/23/4/2215>

<https://training.linuxfoundation.org/blog/opportunities-and-challenges-in-edge-computing-under-kubernetes/>