TRACE

inTegration & haRmonizAtion
of logistiCs opErations

D3.3 TRACE platform (alpha release)

Horizon Innovation Actions | Project No. 101104278

Call HORIZON-CL5-2022-D6-02

Co-funded by
the European Union

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or CINEA. Neither the European Union nor CINEA can be held responsible for them

Dissemination level	Public (PU)
Type of deliverable	Other
Work package	WP3 – Platform Design and Integration
Status - version, date	Final v1.0, 29/11/2024
Deliverable leader	INTRA
Contractual date of delivery	30/11/2024
Actual date of delivery	30/11/2024

List of authors

Author Name	Organisation
Themistoklis Anagnostopoulos, Konstantina Papachristopoulou	INTRA
Sheila Sánchez Rodríguez, Christian Chavez Vasquez	ROB
Tejas Bhagat	BC5
Anestis Papakotoulas	NKUA
Theofilos Triommatis, Ioannis Papamichail	TUC
Kylafas Christos, Tymplalexis Nikolaos, Fountas Panagiotis	UTH
Martin Sénéclauze, Sareh Saeedi	CSEM
Savvas Apostolidis	CERTH
Georgios Andronikidis	SID
Alessio Masola	UNIMORE
Spyros Thermos	CDW
Shameem Puthiya Parambath	UGLA

Version History

Version	Date	Author	Description of changes
0.1	16/07/2024	INTRA	Initial ToC

Version	Date	Author	Description of changes
0.2	22/08/2024	INTRA	Input on Section 3
0.3	16/10/2024	ROB, with contributions from BC5, NKUA, TUC	Input on Section 5
0.4	27/11/2024	INTRA	Final edits, consolidation of inputs and creation of the final draft version for internal review
1.0	29/11/2024	INTRA	Creation of final version addressing internal review comments

Peer Review

	Reviewer Name	Organisation	Date
	Sarantis Paskalis	NKUA	28/11/2024
	Kostas Kolomvatsos	UTH	28/11/2024

Quality Manager Review

	Reviewer Name	Organisation	Date
	Ioannis Neokosmidis	INCITES	30/11/2024

Legal Disclaimer

The information in this document is provided “as is”, and no guarantee or warranty is given that it is fit for any specific purpose. The TRACE project Consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.

Executive Summary

The TRACE platform alpha release represents a significant milestone in developing an integrated logistics operations platform. This deliverable details the technical implementation, integration methodology, and validation of the platform's first major release.

The platform is built on a robust cloud infrastructure hosted on Hetzner Cloud, providing secure and scalable hosting for all platform components. At its core, the platform utilizes a comprehensive CI/CD stack that incorporates Jenkins for automation and continuous integration, Harbor for container image registry, Portainer for container management, and Keycloak for centralised user management and authentication.

The integration approach follows a structured plan spanning from M8 to M36, with development cycles divided into multiple sprints with clear milestones. The platform employs automated deployment processes through CI/CD pipelines, supported by a comprehensive issue tracking system for effective development and integration management.

Testing and validation employ a requirement-based testing methodology, ensuring alignment with user needs. The platform includes detailed validation matrices for all components, including blockchain components, event handling systems, interoperability layer, scheduler and route optimiser, user interfaces, and vehicle support services. This comprehensive testing approach ensures robust functionality across all platform elements.

Security is paramount in the platform's design, implementing multiple security layers including firewalls and VPN access. The system employs role-based access control (RBAC), encrypted communications, DDoS protection, and SSH key authentication to ensure secure operations across all components and user interactions.

The alpha release provides a solid foundation for further development and integration of logistics operations, with particular emphasis on security, scalability, and maintainability. The platform demonstrates readiness for initial deployment and testing in real-world scenarios, paving the way for the subsequent beta release.

Table of Contents

Table of figures

Table of tables

Definitions, Acronyms and Abbreviations

Abbreviation	Definition
AES	Advanced Encryption Standard
CA	Collaboration agreement
CDMS	Cloud-based Data Management System
CI/CD	Continuous Integration / Continuous Development
CLI	Command Line Interface
CNCF	Cloud Native Computing Foundation
DDoS	Distributed Denial of Service
DEK	Data Encryption Key
dNFT	dynamic Non-Fungible Token
DNS	Domain Name System
DSL	Domain-Specific Language
EMM	Event Management Module
Env	Environment
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
IAM	Identity and Access Management
IDM	Intrusion Detection Module
IPv4/IPv6	Internet Protocol version 4 / 6
KEK	Key Encryption Key
KES	Key Encryption Service
RBAC	Role-Based Access Control
RBT	Requirement-Based Testing
RTM	Requirement Traceability Matrix
SH	StreamHandler
SSH	Secure Shell
SSO	Single Sign-On
SYN	Synchronise
UAV	Unmanned Aerial Vehicle

Abbreviation	Definition
UDP	User Datagram Protocol
UI	User Interface
V2V	Vehicle-to-Vehicle
VCS	Version Control System
VM	Virtual Machine
VPN	Virtual Private Network
YAML	Yet Another Markup Language

1 Introduction

1.1 Scope of deliverable

Deliverable D3.3 entitled "TRACE platform (alpha release)" has been prepared in the framework of WP3 "Platform Design and Integration". It is the accompanying report of the software deliverable and provides information on the setting up and the operation of the initial release of the TRACE platform, including the methodology to be followed for the testing and validation of each module and component. This is the first part of three releases, with D3.4 being the beta release at M32 (after the Amendment to the Grant Agreement takes effect), and D3.5 being the final TRACE Platform delivery at the end of the project (M36).

Deliverable D3.3 is directly connected to two tasks, and therefore presents the work performed and the results achieved in:

- T3.5 Platform Integration
- T3.6 Platform Testing and Validation.

1.2 Relation with other work packages/deliverables

Based on the Description of the Action (DoA), D3.3 has several relationships with other Work Packages and deliverables. In particular, it builds upon the technical specifications regarding the reference architecture described in D3.1 [1] and the ecosystem development and use cases presented in D2.4 [2], prepared in the framework of WP2 "Conceptual Framework”. Additionally, D3.3 informs the deliverables of WP4 "Infrastructure & Ecosystem" referring to the infrastructural elements (D4.1, D4.2) and the synchromodal operations (D4.3, D4.4). Since D3.3 is part of a series with D3.4 (TRACE platform beta release) and D3.5 (TRACE platform final release), there is a direct dependency. Also, there is a strong connection with the second iteration of the TRACE Reference Architecture, namely D3.2, which will include feedback from the initial pilot demonstrations.

1.3 Intended audience

This deliverable is primarily targeted at three key audience groups, each with specific interests in the TRACE platform's technical implementation:

- **Technical Teams**: The detailed technical specifications, CI/CD stack configurations, and validation matrices make this document essential for software developers, system administrators, and DevOps engineers directly involved in the platform's development and maintenance. The comprehensive documentation of deployment procedures, security features, and testing methodologies provides these teams with the necessary guidance for implementing and maintaining the platform components.
- **Project Stakeholders**: Technical project managers and platform architects will find value in understanding the overall infrastructure design, integration methodology, and validation approach. The executive summary and high-level architectural descriptions help inform strategic decisions about platform development and resource allocation. Quality assurance teams will particularly benefit from the detailed testing and validation frameworks outlined in the document.
- **External Technical Audiences**: IT consultants, technical evaluators, and future platform maintainers can use this document to understand the platform's technical foundation, security measures, and operational procedures. The public dissemination level of this deliverable makes it accessible to these external audiences, facilitating knowledge transfer and potential platform adoption in other contexts.

1.4 Deliverable structure

This deliverable is organized into six main sections, providing a comprehensive overview of the TRACE platform's alpha release:

Section 1 introduces the document's scope, its relationship with other work packages and deliverables, and identifies the intended audience. This section provides the necessary context for understanding the document's purpose and relevance within the broader TRACE project.

Section 2 details the TRACE Cloud Platform, describing the hosting infrastructure on Hetzner Cloud, including security features and deployment environments. This section provides essential information about the platform's technical foundation and operational environment.

Section 3 presents the TRACE CI/CD Stack, covering user management, version control system, container image registry, CI/CD server, and Portainer. Each component is described in detail, explaining its role in the automated development and deployment pipeline.

Section 4 outlines the Integration Methodology and Plan, describing the approach to integrating various platform components and the development timeline. This section includes the issue tracking system and development workflows that support the integration process.

Section 5 comprises the Platform Testing and Validation framework, presenting a comprehensive set of validation matrices for each platform component. This section details the requirement-based testing approach and provides specific test cases for various platform modules.

Section 6 concludes the document and presents the future steps, pertaining to the beta release of the TRACE Platform.

The deliverable includes an additional section (Annex A) which provides a detailed CI/CD Stack User Guide, offering practical guidance for technical teams working with the platform's development and deployment tools.

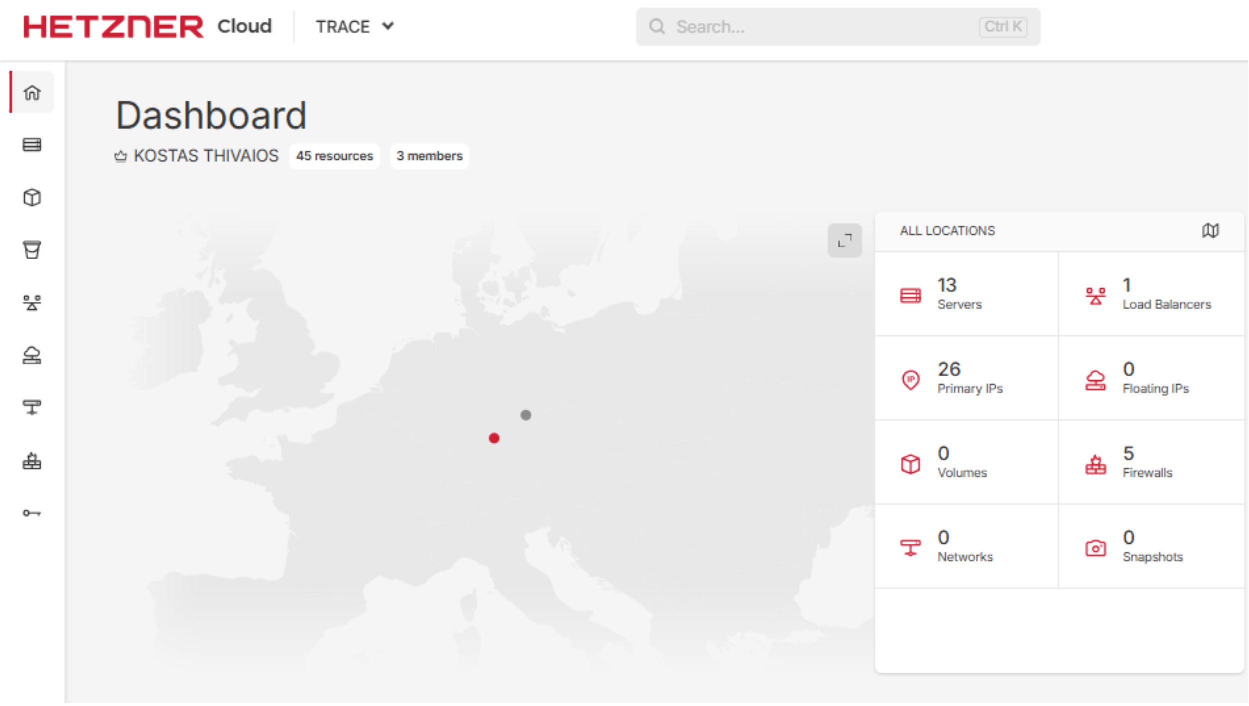
2 TRACE Cloud Platform

This Section provides information about the Cloud infrastructure that is used to host the TRACE Cloud Platform. Additionally, it presents the multiple deployment environments instantiated in the project.

2.1 Hosting Infrastructure

Hetzner Cloud was chosen as the public cloud provider to host the components of the TRACE cloud platform. Hetzner Cloud is a reliable and high-performance cloud service based in Germany that specialises in providing cloud servers (i.e., VMs), storage solutions, and networking services. As a European provider, Hetzner complies with strict GDPR regulations, ensuring secure data handling and privacy. All the TRACE Cloud Platform releases (alpha, beta, and final) will be hosted on Hetzner Cloud, offering robust infrastructure, seamless scalability, and reliable, secure, and efficient operation across its components.

Figure 1 summarises the resources that have been purchased on Hetzner Cloud at their primary data centre in Nuremberg, Germany, to host the components and services of the alpha release.



Dashboard

KOSTAS THIVAIOS 45 resources 3 members

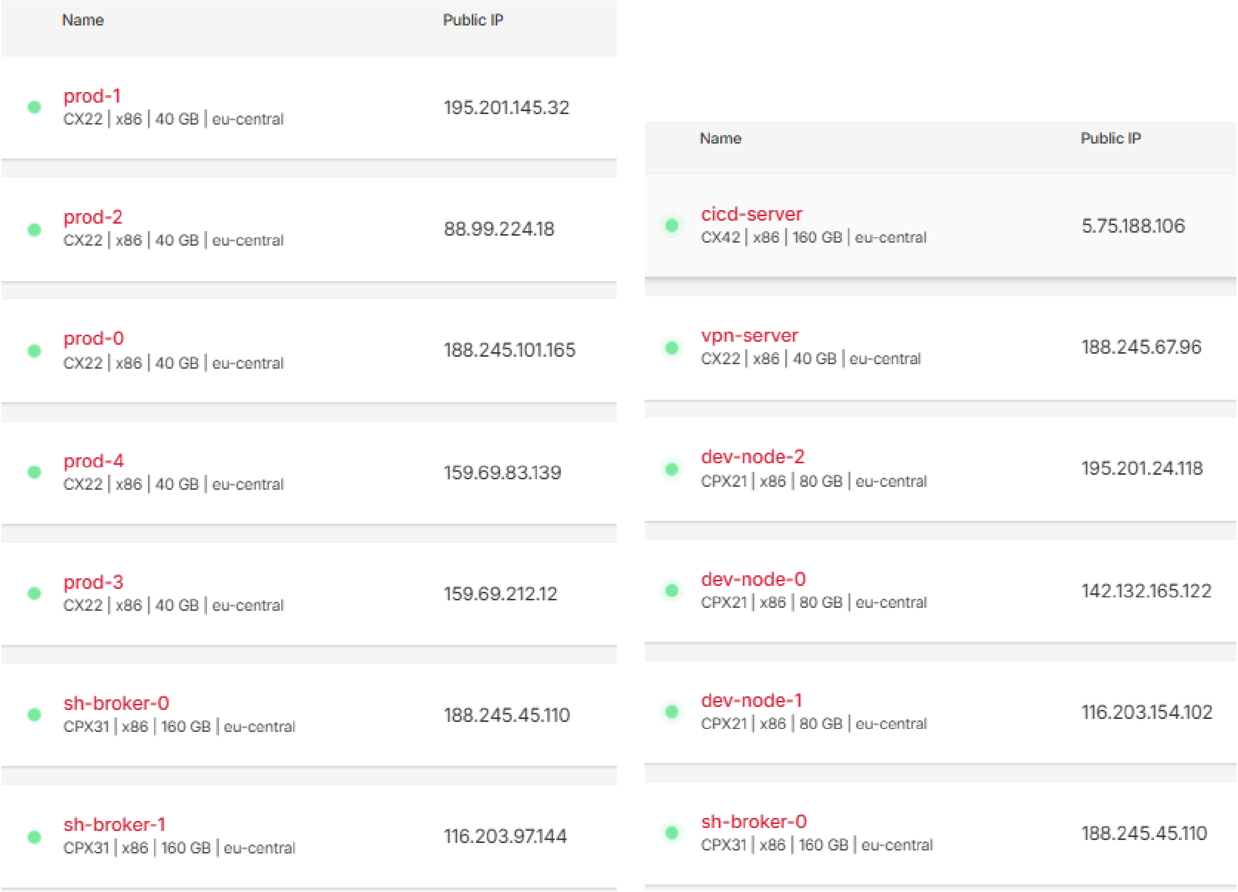
13 Servers	1 Load Balancers
26 Primary IPs	0 Floating IPs
0 Volumes	5 Firewalls
0 Networks	0 Snapshots

Figure 1: Hetzner Cloud Dashboard

These resources are:

- **Virtual Machines**: They are used to host the two platform environments, the CI/CD Stack components, and a VPN server that allows secure connectivity to external users and services.
- Pairs of **IPv4** and **IPv6** primary addresses: They allow external connectivity to the above VMs.
- A **Load Balancer**: Responsible for distributing the incoming traffic to the multiple nodes operating on the cloud platform.
- A group of **firewalls** that protects the instantiated VMs and permits traffic only to authorised sources.
- **Snapshots** of the VMs with different timestamps that allow fault recovery in case of a failure.

The Virtual Machines are the key elements of the platform that are used to host the various TRACE components. Figure 2 is a snapshot of all the deployed VMs of the system, while Table 1 provides further details about their resources and scope. The provided resources of these VMs can be up-scaled or down-scaled depending on the demands. For all VMs, the Ubuntu server (release 22.04) is the open-source operating system that has been installed.



	Name	Public IP
●	prod-1 CX22 | x86 | 40 GB | eu-central	195.201.145.32
●	prod-2 CX22 | x86 | 40 GB | eu-central	88.99.224.18
●	prod-0 CX22 | x86 | 40 GB | eu-central	188.245.101.165
●	prod-4 CX22 | x86 | 40 GB | eu-central	159.69.83.139
●	prod-3 CX22 | x86 | 40 GB | eu-central	159.69.212.12
●	sh-broker-0 CPX31 | x86 | 160 GB | eu-central	188.245.45.110
●	sh-broker-1 CPX31 | x86 | 160 GB | eu-central	116.203.97.144

	Name	Public IP
●	cicd-server CX42 | x86 | 160 GB | eu-central	5.75.188.106
●	vpn-server CX22 | x86 | 40 GB | eu-central	188.245.67.96
●	dev-node-2 CPX21 | x86 | 80 GB | eu-central	195.201.24.118
●	dev-node-0 CPX21 | x86 | 80 GB | eu-central	142.132.165.122
●	dev-node-1 CPX21 | x86 | 80 GB | eu-central	116.203.154.102
●	sh-broker-0 CPX31 | x86 | 160 GB | eu-central	188.245.45.110

Figure 2: Cloud Platform VMs

Table 1: Cloud Platform VMs

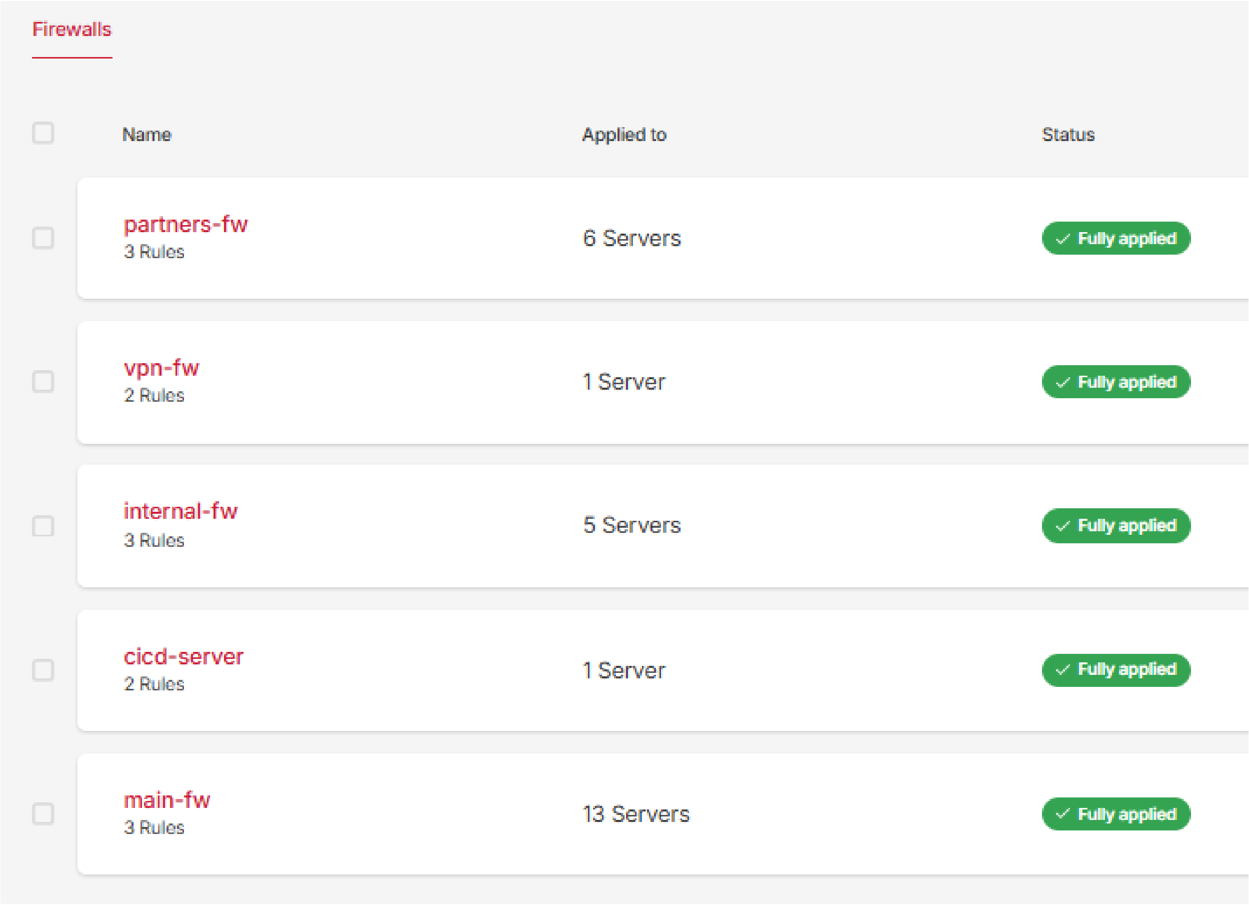
VM Name	Scope	vCPUs	RAM (GBs)	Storage (GBs)
cicd-server	CI/CD Stack	8	16	160
vpn-server	OpenVPN Server	2	4	40
dev-node-0	Staging Env	3	4	80
dev-node-1	Staging Env	3	4	80
dev-node-2	Staging Env	3	4	80
sh-broker-0	Prod Env | StreamHandler	4	8	160
sh-broker-1	Prod Env | StreamHandler	4	8	160

sh-broker-2	Prod Env | StreamHandler	4	8	160
prod-0	Prod Env | CDMS	2	4	40
prod-1	Prod Env | Scheduler RouteOptimiser EventManager	2	4	40
prod-2	Prod Env | Interoperability	2	4	40
prod-3	Prod Env | User Interface	2	4	40
prod-4	Prod Env | Blockchain APIs IDM	2	4	40

2.1.1 Security Features

Various security features in different layers have been applied to the TRACE cloud platform in order to ensure that the deployed software components and tools, as well as their data operations, are secured and protected.

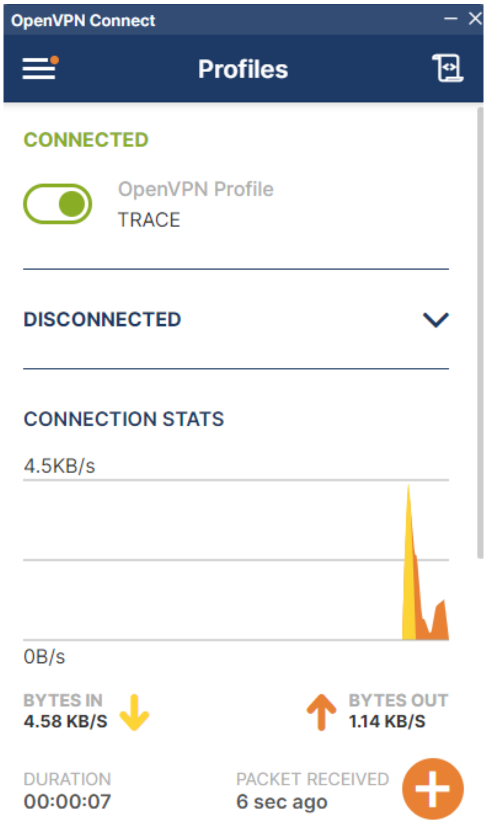
The overall TRACE Cloud Platform is protected by a set of virtual firewalls (a snapshot of which is depicted in Figure 3), ensuring that every interaction, data ingress, and egress, is closely monitored and filtered to prevent any unauthorised or malicious activities. Only authorised users can securely connect to this workspace through an OpenVPN server. This VPN gateway is hosted on a dedicated VM of the infrastructure as a standalone and independent service. It ensures encrypted and secure communication, allowing component administrators, testers and developers to access the environment safely from any location while keeping potential threats at bay.



□	Name	Applied to	Status
□	partners-fw 3 Rules	6 Servers	✓ Fully applied
□	vpn-fw 2 Rules	1 Server	✓ Fully applied
□	internal-fw 3 Rules	5 Servers	✓ Fully applied
□	cicd-server 2 Rules	1 Server	✓ Fully applied
□	main-fw 3 Rules	13 Servers	✓ Fully applied

Figure 3: Cloud Platform Firewalls

On the other hand, components and services deployed on the stakeholders' domains can access the Cloud Platform either through the VPN tunnel or by whitelisting source IP addresses through the firewalls. Platform administrators can provide access to the system through configurable OpenVPN files that users can import into an OpenVPN client, similar to the one depicted in Figure 4:



CONNECTED

OpenVPN Profile
TRACE

DISCONNECTED

CONNECTION STATS

4.5KB/s

0B/s

BYTES IN
4.58 KB/S

BYTES OUT
1.14 KB/S

DURATION
00:00:07

PACKET RECEIVED
6 sec ago

Figure 4: OpenVPN Client Application

Hetzner Cloud enables VM protection by design, as all VMs are protected by the built-in DDoS protection automated system. The system recognises the majority of the most common attack patterns (e.g., DNS reflection, NTP reflection, UDP floods, SYN floods, DNS floods, and invalid packets) in advance, allowing it to block the attacks and effectively mitigate the risk.

Finally, access to the platform VMs is allowed only with the use of SSH public key authentication through cryptographic keys. This approach provides strong and encrypted verification and communication and prevents password attacks (e.g., phishing, keylogging, dictionary attacks, brute-force attacks) or password leaks, as password-based authentication is disabled.

2.2 Deployment Environments

For the needs of the project, two identical environments have been instantiated, namely, the staging and production. These environments span across multiple VMs, ensuring that services running in different environments are kept isolated and secure.

- The **staging environment** is a pre-production setup that closely mirrors the production environment in terms of architecture, configuration, and scale. It acts as a sandbox in which the individual TRACE components are extensively tested before being deployed to the production environments. It allows technical teams to work on new features, fix bugs, and test functionality in an isolated setup without impacting end users or other components. TRACE developers have complete access and can push updates regularly, testing their work to ensure functionality and stability before moving to the next stage.
- The **production environment** is the live, user-facing environment where the platform is fully deployed and accessible to end-users. This environment will be used to host the stable versions A and B of the TRACE components that will be used in the context of the three demonstrators.

Docker is an ideal tool for implementing both deployment environments of the TRACE platform due to its ability to create consistent, portable, and scalable application environments. It allows each component of the to be packaged into a container. Each container includes the actual application, its dependencies, libraries, and runtime, ensuring that it runs consistently across different environments (development, testing, and production). Each component module (e.g., Event Filtering, Brokers, Connectors, IDM, etc.) runs in an isolated container, allowing for independent updates and scaling. These containers can be deployed on any VM of the cloud infrastructure. Moreover, Docker Compose is used to bundle together multiple containers that need to be deployed together and operate on top of common virtual resources, like Docker networks and volumes.

3 TRACE CI/CD Stack

This Section provides detailed information about the TRACE CI/CD Stack that has been deployed to support the development, integration, testing, and deployment of the TRACE components in the alpha release. In addition to the description of the tools included here, *Annex A: CI/CD Stack User Guide* provides the detailed user guide of the CI/CD Stack that technical teams have followed during the alpha release of the platform. Table 2 provides a list of all the URLs of the CI/CD Stack tools. External users can request an SSO account by opening an issue on the public TRACE issue-tracking platform on GitHub. A system administrator will be notified and create the respective account.

Table 2: CI/CD Tools URLs

Tool	URL
GitHub Organisation	https://github.com/orgs/trace-project-eu/dashboard
Keycloak	https://keycloak.trace.rid-intrasoft.eu/
Jenkins	https://jenkins.trace.rid-intrasoft.eu/
Harbor	https://harbor.trace.rid-intrasoft.eu/
Portainer	https://portainer.trace.rid-intrasoft.eu/

3.1 User Management

Keycloak [3] is the tool that was selected, installed, and configured to act as the centralised user-management service of the TRACE CI/CD Stack. Keycloak is an open-source identity and access management (IAM) solution that provides authentication, authorisation, and user management. Dedicated accounts were created for all the platform developers. These are added to groups corresponding to different technical teams of the consortium. The Keycloak groups have different access permissions on the CI/CD Stack resources, following the RBAC model.

User accounts are used for user authentication across the multiple CI/CD Stack tools. When users try to log into any of these tools, they are redirected to the Keycloak log-in page to enter their credentials, as presented in Figure 5. TRACE Keycloak was further integrated with GitHub, providing an extra SSO option to users, allowing them to use their GitHub account to log into the system.

New or external users can request access to the CI/CD stack services through the dedicated issue tracking platform described in Section 4.2.

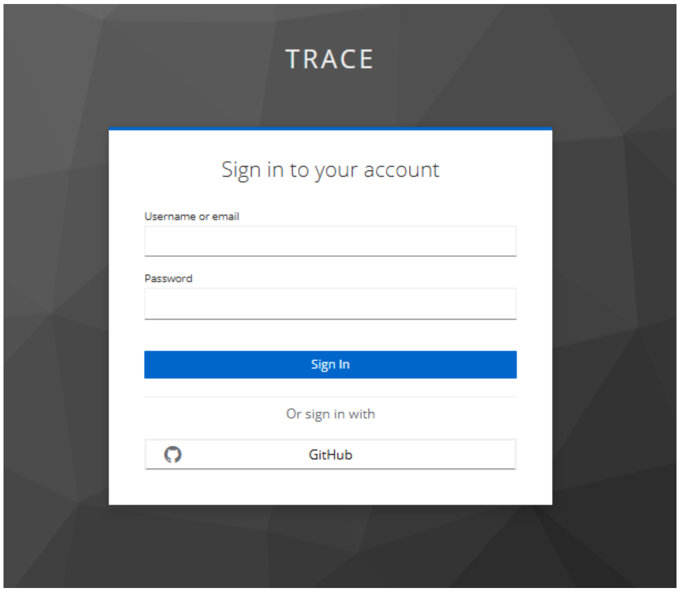


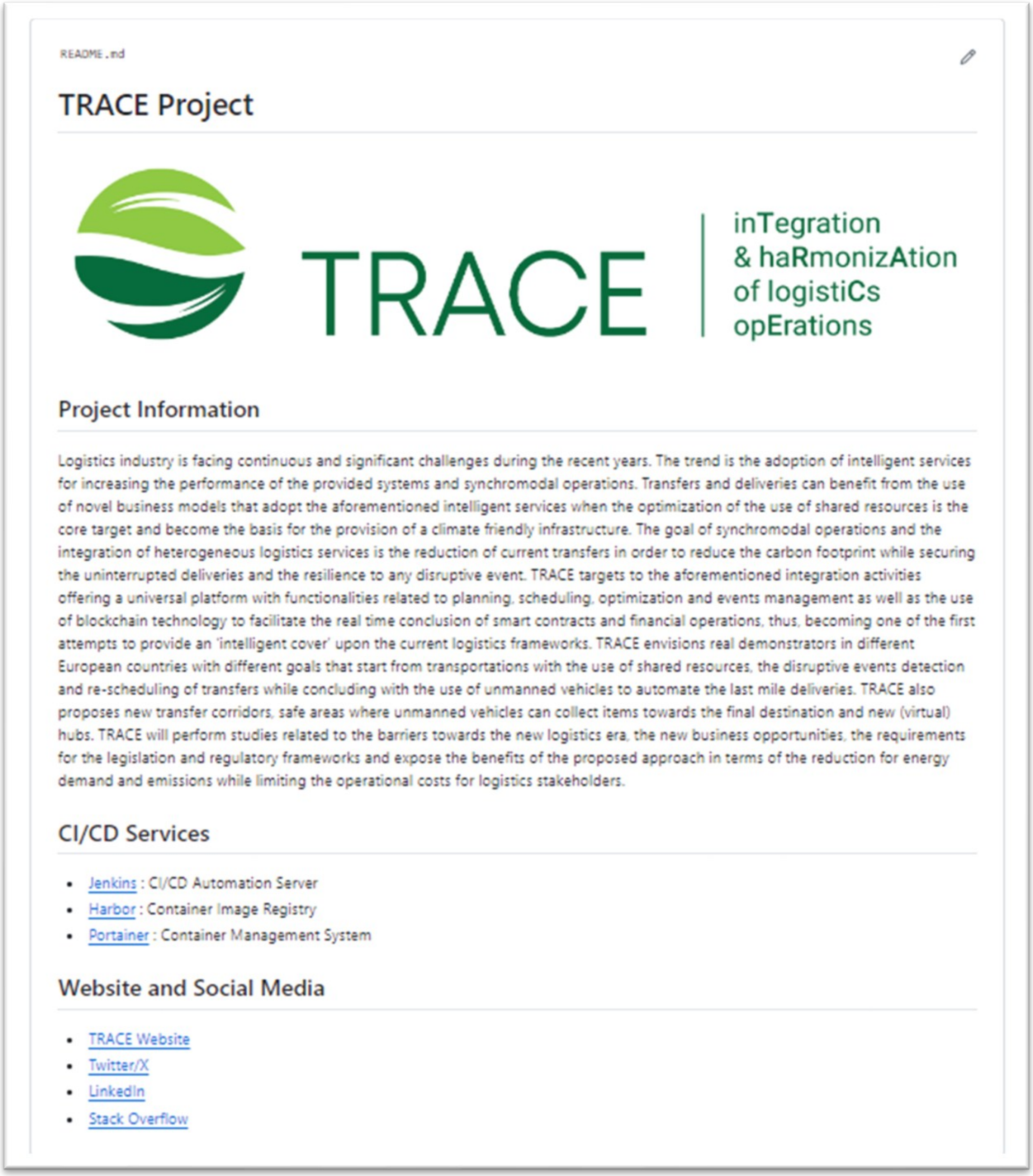
Figure 5: Log in Page in CI/CD Stack

3.2 Version Control System

Version control, also called source control, is the process of keeping track of changes made to software code. Version control systems (VCS) are software tools that assist developers in recording every change made to the code in code snapshots called commits, along with their associated metadata (e.g., the author of the commit, the time of the change, a message describing the changes, etc.). Each commit has a unique ID number, allowing developers to examine earlier versions of the code, compare changes, and, if needed, revert to previous versions. This approach enables multiple developers to work on the same codebase simultaneously by tracking their individual code updates.

In the context of TRACE, GitHub [4] has been selected to implement the VCS in the CI/CD Stack. It comprises a web-based platform that operates on top of Git, allowing development teams to host their code repositories, share them with others, and work together on projects by contributing, reviewing, and merging code. Moreover, it comes with an extended set of features for the development procedure, such as issue tracking, code reviews, wiki, etc. It also provides extensive branching capabilities: Typically, there is a main branch in a repository hosting the latest stable version of the source code, and dedicated feature or bugfix branches, where developers work on specific tasks. Once developers complete their source code changes, they merge back into the main branch.

A dedicated organisation has been created for hosting and organising the available code repositories, as depicted in Figure 6.



README.md

TRACE Project

TRACE inTegration & haRmonizAtion of logistiCs opErations

Project Information

Logistics industry is facing continuous and significant challenges during the recent years. The trend is the adoption of intelligent services for increasing the performance of the provided systems and synchromodal operations. Transfers and deliveries can benefit from the use of novel business models that adopt the aforementioned intelligent services when the optimization of the use of shared resources is the core target and become the basis for the provision of a climate friendly infrastructure. The goal of synchromodal operations and the integration of heterogeneous logistics services is the reduction of current transfers in order to reduce the carbon footprint while securing the uninterrupted deliveries and the resilience to any disruptive event. TRACE targets to the aforementioned integration activities offering a universal platform with functionalities related to planning, scheduling, optimization and events management as well as the use of blockchain technology to facilitate the real time conclusion of smart contracts and financial operations, thus, becoming one of the first attempts to provide an 'intelligent cover' upon the current logistics frameworks. TRACE envisions real demonstrators in different European countries with different goals that start from transportations with the use of shared resources, the disruptive events detection and re-scheduling of transfers while concluding with the use of unmanned vehicles to automate the last mile deliveries. TRACE also proposes new transfer corridors, safe areas where unmanned vehicles can collect items towards the final destination and new (virtual) hubs. TRACE will perform studies related to the barriers towards the new logistics era, the new business opportunities, the requirements for the legislation and regulatory frameworks and expose the benefits of the proposed approach in terms of the reduction for energy demand and emissions while limiting the operational costs for logistics stakeholders.

CI/CD Services

- Jenkins : CI/CD Automation Server
- Harbor : Container Image Registry
- Portainer : Container Management System

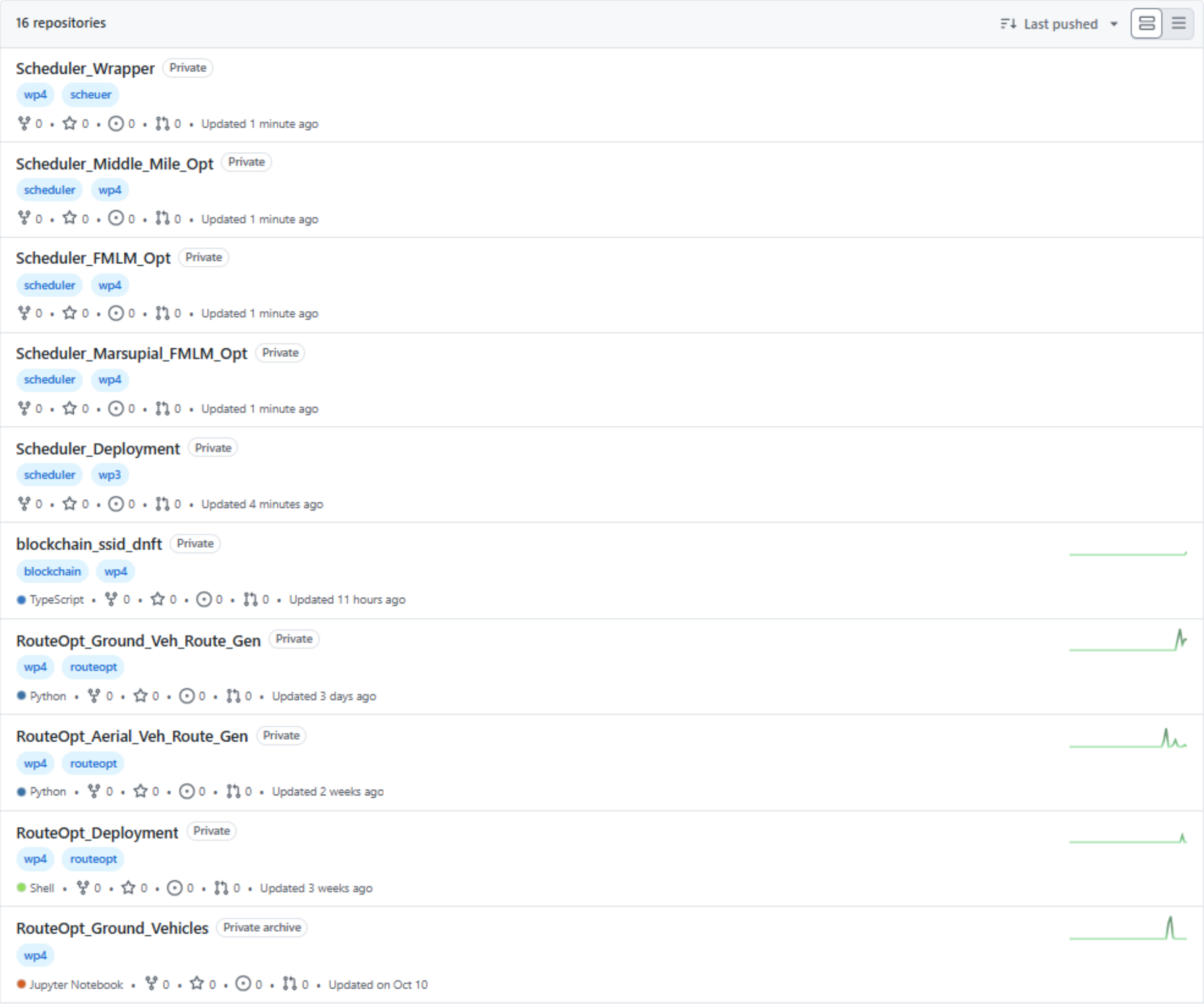
Website and Social Media

- TRACE Website
- Twitter/X
- LinkedIn
- Stack Overflow

Figure 6: TRACE GitHub Organisation

Each component in the TRACE platform is associated with one or more repositories, depending on the number of individual modules comprising the component. Specific tags have been added to each repository to group them per component and WPs, as depicted in Figure 7. By clicking a tag, users can filter repositories and get a list only of those that are associated with this specific tag. The developers of each partner were invited to the organisation and added to the respective team. From there, they are able to access some repositories according to the permissions granted to their team. Each GitHub repository must include the following files to allow the automated deployment through the CI/CD services:

- **README:** A short description of the component and its main functionalities, along with instructions about its deployment.
- **Jenkins files:** Configuration files with the definitions of the Jenkins [5] pipelines associated with the component necessary for automated deployment, termination, and testing.
- **Docker** [6] **Compose YAML file:** Configuration file used for the deployment of software components using the Docker compose plugin [7].
- Other **runtime configuration files**, such as environmental variables.



16 repositories

Scheduler_Wrapper Private
wp4 scheuer
0 • ☆ 0 • ⊙ 0 • 0 • Updated 1 minute ago

Scheduler_Middle_Mile_Opt Private
scheduler wp4
0 • ☆ 0 • ⊙ 0 • 0 • Updated 1 minute ago

Scheduler_FMLM_Opt Private
scheduler wp4
0 • ☆ 0 • ⊙ 0 • 0 • Updated 1 minute ago

Scheduler_Marsupial_FMLM_Opt Private
scheduler wp4
0 • ☆ 0 • ⊙ 0 • 0 • Updated 1 minute ago

Scheduler_Deployment Private
scheduler wp3
0 • ☆ 0 • ⊙ 0 • 0 • Updated 4 minutes ago

blockchain_ssid_dnft Private
blockchain wp4
● TypeScript • 0 • ☆ 0 • ⊙ 0 • 0 • Updated 11 hours ago

RouteOpt_Ground_Veh_Route_Gen Private
wp4 routeopt
● Python • 0 • ☆ 0 • ⊙ 0 • 0 • Updated 3 days ago

RouteOpt_Aerial_Veh_Route_Gen Private
wp4 routeopt
● Python • 0 • ☆ 0 • ⊙ 0 • 0 • Updated 2 weeks ago

RouteOpt_Deployment Private
wp4 routeopt
● Shell • 0 • ☆ 0 • ⊙ 0 • 0 • Updated 3 weeks ago

RouteOpt_Ground_Vehicles Private archive
wp4
● Jupyter Notebook • 0 • ☆ 0 • ⊙ 0 • 0 • Updated on Oct 10

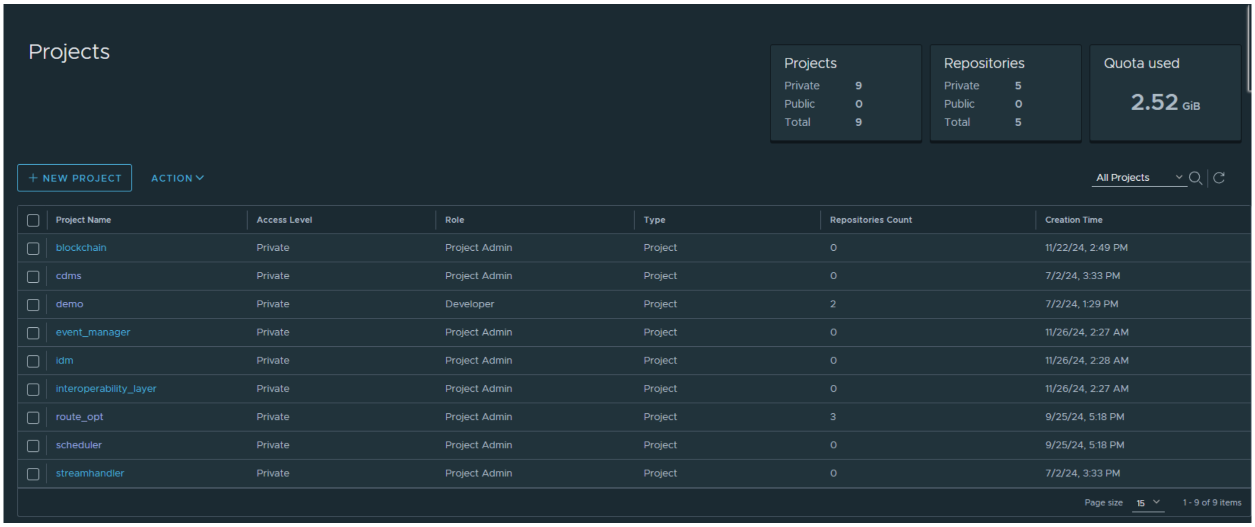
Figure 7: GitHub Repositories

3.3 Container Image Registry

Harbor [8]is an open-source container image registry developed as a Cloud Native Computing Foundation (CNCF) project. It stores and secures artefacts with policies and role-based access control, emphasising security, compliance, and performance. It supports storing, signing, scanning, and distributing container images. Harbor is integrated with the other tools of the TRACE CI/CD stack.

Different Harbor projects (registries) are created per technical component and are mapped to the platform building blocks. Each project acts as a registry used to store one or more container repositories. Developers with a TRACE Keycloak account have image push/pull permissions to specific projects (only associated with their developments). Additionally, they can access the web GUI and create repositories under the projects they manage to host their container images.

Figure 8 presents a snapshot of the Harbor Web UI and the different projects created to host the packages of the alpha release. Through this UI, users can view the different tags (i.e., versions) of the docker images, pull a specific tag, upload new images, and delete tags that are not needed anymore. Moreover, they can initiate security scans on their stored images to detect vulnerabilities in the bundled code.



Projects	
Private	9
Public	0
Total	9

Repositories	
Private	5
Public	0
Total	5

Quota used
2.52 GiB

□	Project Name	Access Level	Role	Type	Repositories Count	Creation Time
□	blockchain	Private	Project Admin	Project	0	11/22/24, 2:49 PM
□	cdms	Private	Project Admin	Project	0	7/2/24, 3:33 PM
□	demo	Private	Developer	Project	2	7/2/24, 1:29 PM
□	event_manager	Private	Project Admin	Project	0	11/26/24, 2:27 AM
□	idm	Private	Project Admin	Project	0	11/26/24, 2:28 AM
□	interoperability_layer	Private	Project Admin	Project	0	11/26/24, 2:27 AM
□	route_opt	Private	Project Admin	Project	3	9/25/24, 5:18 PM
□	scheduler	Private	Project Admin	Project	0	9/25/24, 5:18 PM
□	streamhandler	Private	Project Admin	Project	0	7/2/24, 3:33 PM

Page size 15 1 - 9 of 9 items

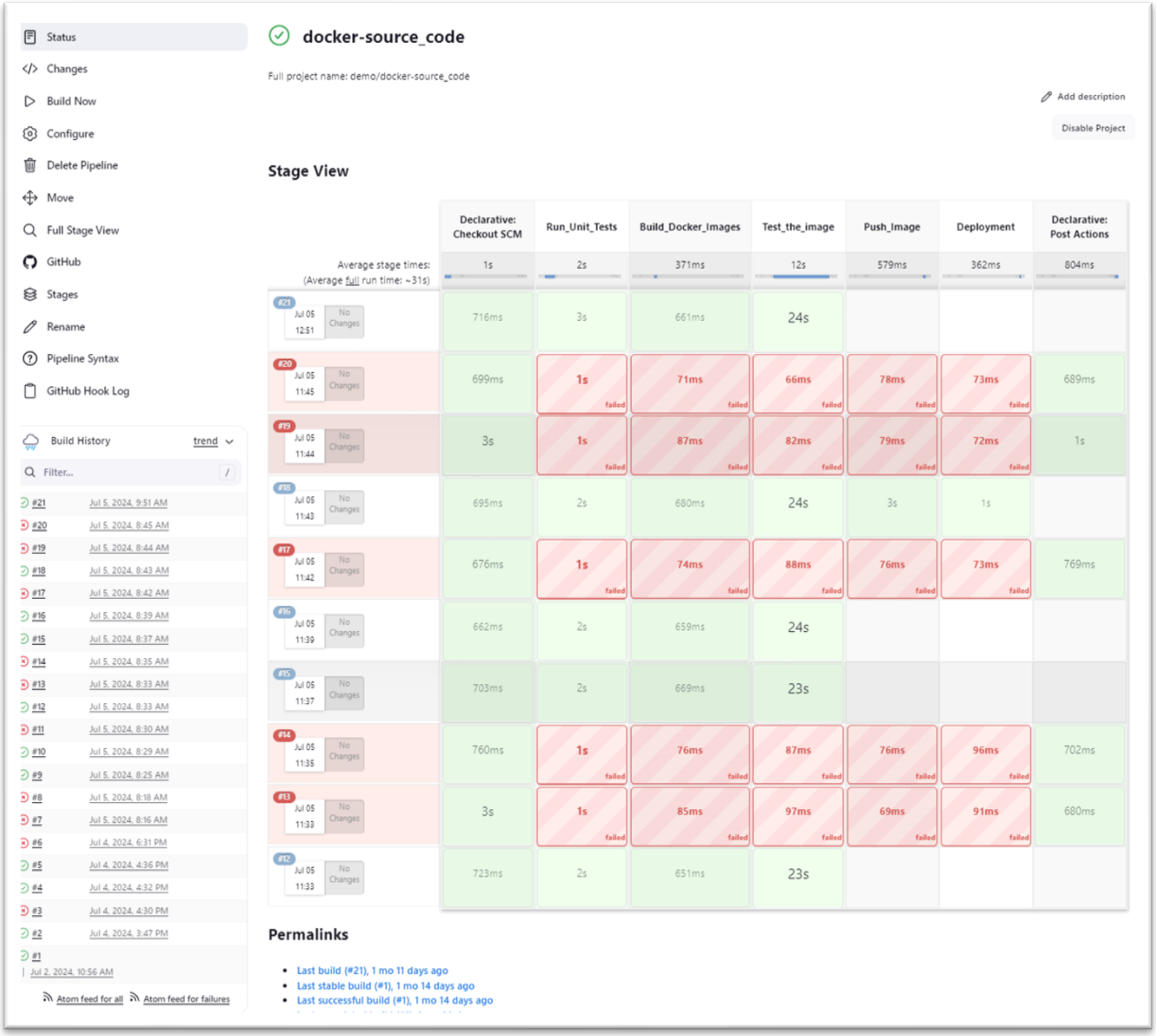
Figure 8: Harbor Container Registry

A retention policy for keeping the latest five docker image tags per project per repository has been applied for all the projects. This policy is automatically executed every hour. The stored images are pulled and deployed in the staging and production environments of the TRACE platform through Jenkins pipelines. For this purpose, a dedicated robot account has been configured for Jenkins, allowing the pipelines to pull the images from the required repository. For storing the built images, each developer is able to push packaged images either through Jenkins as part of a pipeline (automated pipeline) or from their own local system using a CLI token.

3.4 CI/CD Server

In the context of the TRACE project, Jenkins implements the role of the centralised CI/CD server. Jenkins [5] is an automation server that facilitates CI/CD pipelines, which include a sequence of actions and tasks.

These actions typically include pulling the most recent version of the code, building the Docker [6] images, checking for any issues, running unit and integration tests, and reporting any potential problems. After these checks, Jenkins releases the artefact for deployment and deploys the software component for further testing. Figure 9 demonstrates a project pipeline that includes multiple events and actions called stages.



docker-source_code

Full project name: demo/docker-source_code

Stage View

Average stage times: (Average full run time: ~31s)	Declarative: Checkout SCM	Run_Unit_Tests	Build_Docker_Images	Test_the_image	Push_Image	Deployment	Declarative: Post Actions
	1s	2s	371ms	12s	579ms	362ms	804ms
#21 Jul 05 12:51 No Changes	716ms	3s	661ms	24s			
#20 Jul 05 11:45 No Changes	699ms	1s failed	71ms failed	66ms failed	78ms failed	73ms failed	689ms
#19 Jul 05 11:44 No Changes	3s	1s failed	87ms failed	82ms failed	79ms failed	72ms failed	1s
#18 Jul 05 11:43 No Changes	695ms	2s	680ms	24s	3s	1s	
#17 Jul 05 11:42 No Changes	676ms	1s failed	74ms failed	88ms failed	76ms failed	73ms failed	769ms
#16 Jul 05 11:39 No Changes	662ms	2s	659ms	24s			
#15 Jul 05 11:37 No Changes	703ms	2s	669ms	23s			
#14 Jul 05 11:35 No Changes	760ms	1s failed	76ms failed	87ms failed	76ms failed	96ms failed	702ms
#13 Jul 05 11:33 No Changes	3s	1s failed	85ms failed	97ms failed	69ms failed	91ms failed	680ms
#12 Jul 05 11:33 No Changes	723ms	2s	651ms	23s			

Build History trend

Filter...

#21	Jul 5, 2024, 9:51 AM
#20	Jul 5, 2024, 8:45 AM
#19	Jul 5, 2024, 8:44 AM
#18	Jul 5, 2024, 8:43 AM
#17	Jul 5, 2024, 8:42 AM
#16	Jul 5, 2024, 8:39 AM
#15	Jul 5, 2024, 8:37 AM
#14	Jul 5, 2024, 8:35 AM
#13	Jul 5, 2024, 8:33 AM
#12	Jul 5, 2024, 8:33 AM
#11	Jul 5, 2024, 8:30 AM
#10	Jul 5, 2024, 8:29 AM
#9	Jul 5, 2024, 8:25 AM
#8	Jul 5, 2024, 8:18 AM
#7	Jul 5, 2024, 8:16 AM
#6	Jul 4, 2024, 6:31 PM
#5	Jul 4, 2024, 4:36 PM
#4	Jul 4, 2024, 4:32 PM
#3	Jul 4, 2024, 4:30 PM
#2	Jul 4, 2024, 3:47 PM
#1	Jul 2, 2024, 10:56 AM

Atom feed for all Atom feed for failures

Permalinks

- Last build (#21), 1 mo 11 days ago
- Last stable build (#1), 1 mo 14 days ago
- Last successful build (#1), 1 mo 14 days ago

Figure 9: Jenkins Demo Pipeline

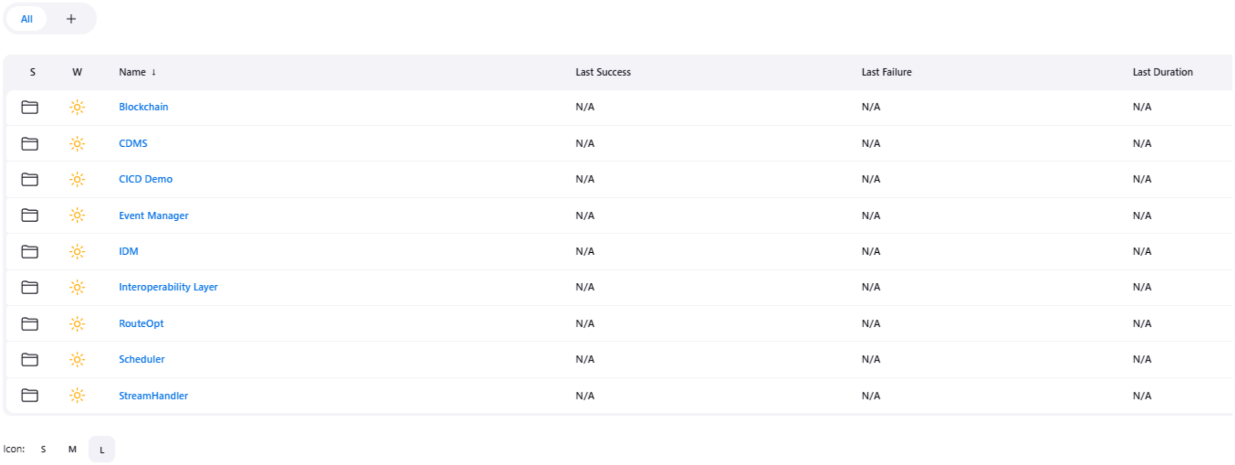
The CI/CD pipelines can be triggered in two ways:

- **Automatically**: In this scenario, developers make a local clone or copy of the component's source code from a remote repository and perform updates. Once the changes are ready, the developers commit and push the updated code to the centralised repository. The Jenkins server is automatically notified of these incoming changes and initiates the execution of a pipeline.
- **Manually**: In this scenario, a built container image is already pushed to the centralised container registry. Component developers or administrators log into the Jenkins Dashboard and manually initiate the execution of a pipeline that uses the said container image.

Jenkins provides an extensible toolset for modelling user-defined delivery pipelines "as code" using the Pipeline Domain-Specific Language (DSL) syntax in a configuration file called "Jenkinsfile". The Jenkinsfile is committed and stored in the component's GitHub repository, making the CI/CD process an integral part of the component. A part of the committed Jenkinsfile that defines the pipeline of Figure 9, is the following:

```
pipeline {
    agent {
        node {
            label 'dev02'
        }
    }
    environment {
        APP_NAME = "dummyrest"
        MAJOR_RELEASE = 0.1
        DOCKER_TAG = "${MAJOR_RELEASE}.${env.BUILD_NUMBER}"
        DOCKER_REG = "harbor.trace.rid-intrasoft.eu"
        DOCKER_REPO = "/demo/"
        DOCKER_REG_CREDS = "harbor-jenkins-creds"
    }

    stages {
        // *************************
        // *** RUN THE UNIT TESTS ***
        // *************************
        stage("Run_Unit_Tests"){

            steps{

                echo "***** Running Unit Tests *****"

                sh 'docker image build -t unittest:test SourceCode'

                sh 'docker container rm -f unittest || true'

                sh 'docker container run -e "PYTHONPATH=/dummyrest" -P --name unittest
--rm unittest:test pytest'

                sh 'docker image rm unittest:test'

            }

        }


        // *************************

        // *** IMAGE BUILD STAGE ***

        // *************************

        stage("Build_Docker_Images"){

            steps {

                echo "***** Building Docker Image *****"

                sh 'DOCKER_TAG=test docker compose build'

            }

        }


        // **************************************

        // *** Functional & Integration Tests ***

        // **************************************

        stage("Test_the_image"){

            steps {

                echo "***** Running Functional Tests *****"

                sh 'docker container rm -f dummyrest'

                sh 'DOCKER_TAG=test docker compose up -d'

                sh '''

                HOST_URL=$(hostname -I | awk '{print $1}')

                sleep 20 && bash jenkins/tests/func_test.sh "${HOST_URL}:8000"

                '''

                sh 'DOCKER_TAG=test docker compose down --rmi all'

            }

        }
        // ****************************
        // *** Push Images In JFrog ***
        // ****************************
        stage("Push_Image"){
            when {
                environment name: "GIT_BRANCH", value: "origin/master"
            }
            steps {
                withCredentials([[$class: 'UsernamePasswordMultiBinding',
credentialsId: "${DOCKER_REG_CREDS}", usernameVariable: 'USERNAME', passwordVariable:
'PASSWORD']]){
                    echo "***** Push Docker Image *****"
                    sh 'docker compose build'
                    sh 'docker login ${DOCKER_REG} -u ${USERNAME} -p ${PASSWORD}'
                    sh 'docker image push
${DOCKER_REG}${DOCKER_REPO}${APP_NAME}:${DOCKER_TAG}'
                    sh 'DOCKER_TAG="latest" docker compose build'
                    sh 'docker image push
${DOCKER_REG}${DOCKER_REPO}${APP_NAME}:latest'
                }
            }
        }

        // **************
        // *** Deploy ***
        // **************
        stage("Deployment"){
            when {
                environment name: "GIT_BRANCH", value: "origin/master"
            }

            steps {
                withCredentials([[$class: 'UsernamePasswordMultiBinding',
credentialsId: "${DOCKER_REG_CREDS}", usernameVariable: 'USERNAME', passwordVariable:
'PASSWORD']]){
                    echo "***** Deploy Application *****"
                    sh 'docker login ${DOCKER_REG} -u ${USERNAME} -p ${PASSWORD}'
                    sh 'docker compose pull'

                    sh 'docker compose up -d'

                    sh 'docker ps'

                }

            }

        }

    }

    post{

        failure{

            // slackSend (color: "#FF0000", message: "Job FAILED: '${env.JOB_NAME} [${env.BUILD_NUMBER}]' (${env.BUILD_URL})")

            sh 'docker image rm ${APP_NAME}:test &> /dev/null || true'

            sh 'DOCKER_TAG=test docker compose down --rmi all'

        }


        // success{

        //     slackSend (color: "#008000", message: "Job SUCCESSFUL: '${env.JOB_NAME} [${env.BUILD_NUMBER}]' (${env.BUILD_URL})")

        // }

    }

}
```

Dedicated workspaces in Jenkins have been created to host various pipelines for each of the TRACE components. These workspaces are accessible only by teams responsible for the development of the respective components, following the Role-Based Access Control model applied to the whole CI/CD Stack. Figure 10 captures a snapshot of the available workspace for the alpha release of the TRACE platform. Each workspace contains one or more pipelines associated with the internal modules of each component.



All +

S	W	Name ↓	Last Success	Last Failure	Last Duration
📁	☀	Blockchain	N/A	N/A	N/A
📁	☀	CDMS	N/A	N/A	N/A
📁	☀	CICD Demo	N/A	N/A	N/A
📁	☀	Event Manager	N/A	N/A	N/A
📁	☀	IDM	N/A	N/A	N/A
📁	☀	Interoperability Layer	N/A	N/A	N/A
📁	☀	RouteOpt	N/A	N/A	N/A
📁	☀	Scheduler	N/A	N/A	N/A
📁	☀	StreamHandler	N/A	N/A	N/A

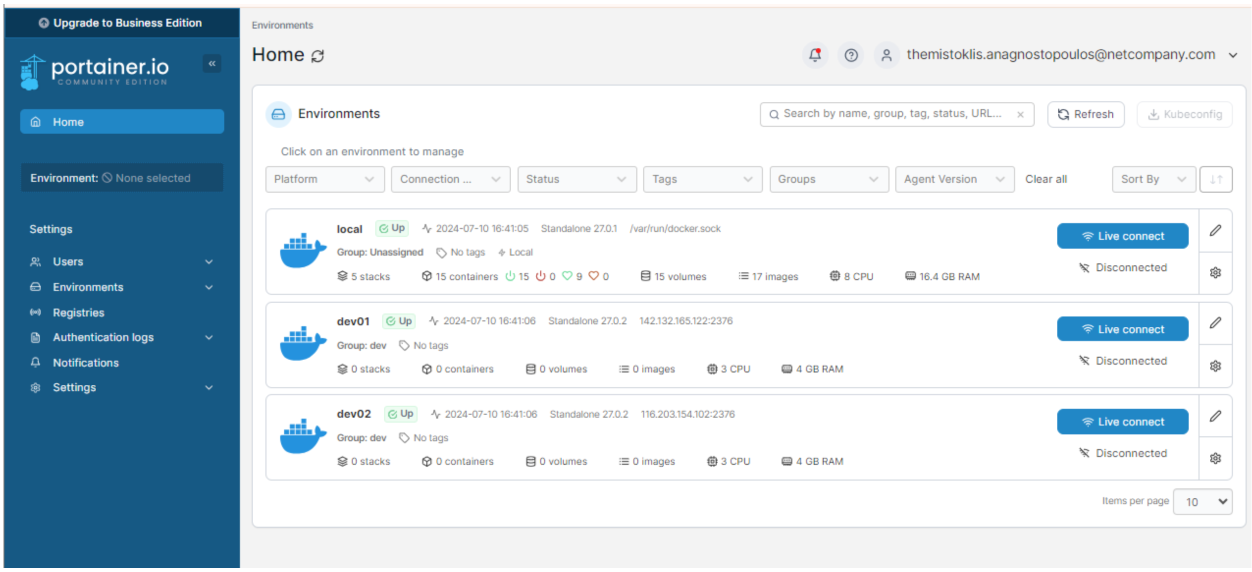
Icon: S M L

Figure 10: Jenkins Pipelines Workspaces

3.5 Portainer

Portainer [9] is a lightweight management UI designed to simplify the management of Docker environments. It offers a user-friendly web interface, making it accessible to both beginners and experienced professionals. With Portainer, users can manage individual Docker hosts or Docker Swarm clusters, deploy and configure applications, and handle containers, images, networks, and volumes with ease. It provides comprehensive monitoring tools and access to logs, facilitating performance monitoring and troubleshooting. Additionally, Portainer includes role-based access control for security in multi-user environments and supports deployment on various platforms, including Linux, Windows, and macOS. Extensible through support for extensions, Portainer allows users to customise and enhance its capabilities to meet specific needs. Overall, Portainer is a powerful tool for deploying, monitoring, and managing containerised applications.

A snapshot of the Portainer dashboard is shown in Figure 11. Users can see the list of the hosts comprising the deployment environments. By clicking on a specific host, they can see and investigate the Docker resources running on that host.



Click on an environment to manage

local Up 2024-07-10 16:41:05 Standalone 27.0.1 /var/run/docker.sock
Group: Unassigned No tags Local
5 stacks 15 containers 15 0 9 0 15 volumes 17 images 8 CPU 16.4 GB RAM
Disconnected

dev01 Up 2024-07-10 16:41:06 Standalone 27.0.2 142.132.165.122:2376
Group: dev No tags
0 stacks 0 containers 0 volumes 0 images 3 CPU 4 GB RAM
Disconnected

dev02 Up 2024-07-10 16:41:06 Standalone 27.0.2 116.203.154.102:2376
Group: dev No tags
0 stacks 0 containers 0 volumes 0 images 3 CPU 4 GB RAM
Disconnected

Figure 11: Portainer Dashboard

4 Integration Methodology and Plan

In the context of TRACE, a modern CI/CD methodology combined with some fundamental Agile principles was applied, aspiring to break down the development, testing, integration, and deployment of the various platform elements into smaller, manageable sprints. To achieve that, the technical teams of the consortium utilised the CI/CD Stack described in the previous section to enable the continuous delivery of the platform components and services by automating their build, test, and deployment process throughout each sprint. The outcome of this approach is multiple pipelines that enable frequent release updates, enhancing the Agile approach of rapid, incremental delivery. The primary goal of this methodology is to deliver the thoroughly tested and integrated TRACE cloud platform in two major releases on months 18 and 32 of the project, as defined by the project work plan.

4.1 Integration Plan

The integration process of the TRACE platform began with the definition of the integration plan. This plan defines periods with concrete timelines and goals, acting as a roadmap for all the subsequent development, testing, integration, and deployment activities, with the aim of releasing two major platform versions. The development of the integration plan followed the identification of the platform components and the design of the architecture as the final step of the architecture development methodology presented in D3.1 [1].

Table 3 summarises the integration plan:

Table 3: TRACE Integration Plan

Period	Dates	Phase	Description
M8 - M11	01/2024 - 04/2024	Preparation - CI/CD Deployment	Preparation, deployment, and configuration of the centralised TRACE CI/CD Platform
M4 - M16	09/2023 - 09/2024	1st Dev Cycle	Components developments in all technical tasks; Unit tests, functional tests, bilateral integration tests; The period will be divided into multiple sprints
M16	09/2024	Components stable v1 release	Milestone: A stable v1 release for each component is expected in the project registry
M17-18	10/2024 - 11/2024	End-to-end platform integration	Execution of end-to-end platform functional and integration test

Period	Dates	Phase	Description
M18	**11/2024**	**TRACE Platform alpha release**	**Milestone: 1st integrated platform release -> D3.3**
M19 - M30	12/2024 - 11/2025	2nd Dev Cycle	Components developments in all technical tasks; Unit tests, functional tests, bilateral integration tests; The period will be divided into multiple sprints
M30	11/2025	Components stable v2 release	Milestone: A stable v1 release for each component is expected in the project registry
M31 - M32	12/2025 - 01/2026	End-to-end platform integration	Execution of end-to-end platform functional and integration test
M32	**01/2026**	**TRACE Platform beta release**	**Milestone: 2nd integrated platform release -> D3.4**
M33- M36	02/2026 - 05/2026	Platform Validation & Finalisation	Feedback from end-users and pilots; Fixes and updates based on the feedback
M36	**05/2026**	**TRACE Platform final release**	**Milestone: TRACE final platform release -> D3.5**

The integration plan comprises the following phases:

- **First and second development cycles (Months 04-16 and 19-30)**: during these periods, all technical teams proceed with the core development activities of the various platform elements. Additionally, this period involves a wide range of tests to ensure the robust operation of each individual component. These tests include unit tests for testing specific parts of the source code, functional tests for verifying the expected functionality of a particular building block, and bilateral integration tests for testing APIs, communication channels, etc. The progress of each component is updated during the weekly WP3 meetings, where all the technical teams get together and monitor/report the advancements of the platform. At the end of each of these phases, a milestone will be achieved with the release of a stable package of the components comprising each significant release of the platform. These packages are stored in the centralised artifactory registry, using specific tags, as described in Section 3.3.
- **End-to-end platform integration (Months 17-18 and 31-32)**: these two-month periods are dedicated to conducting end-to-end functional and integration tests for the platform as a whole, preparing the deployment and operation on the pilots. The main goal of this phase is to detect

potential errors and bugs that need to be fixed before the actual release of the platform. These phases will end with the two integrated releases of the TRACE cloud platform in months 18 and 33.

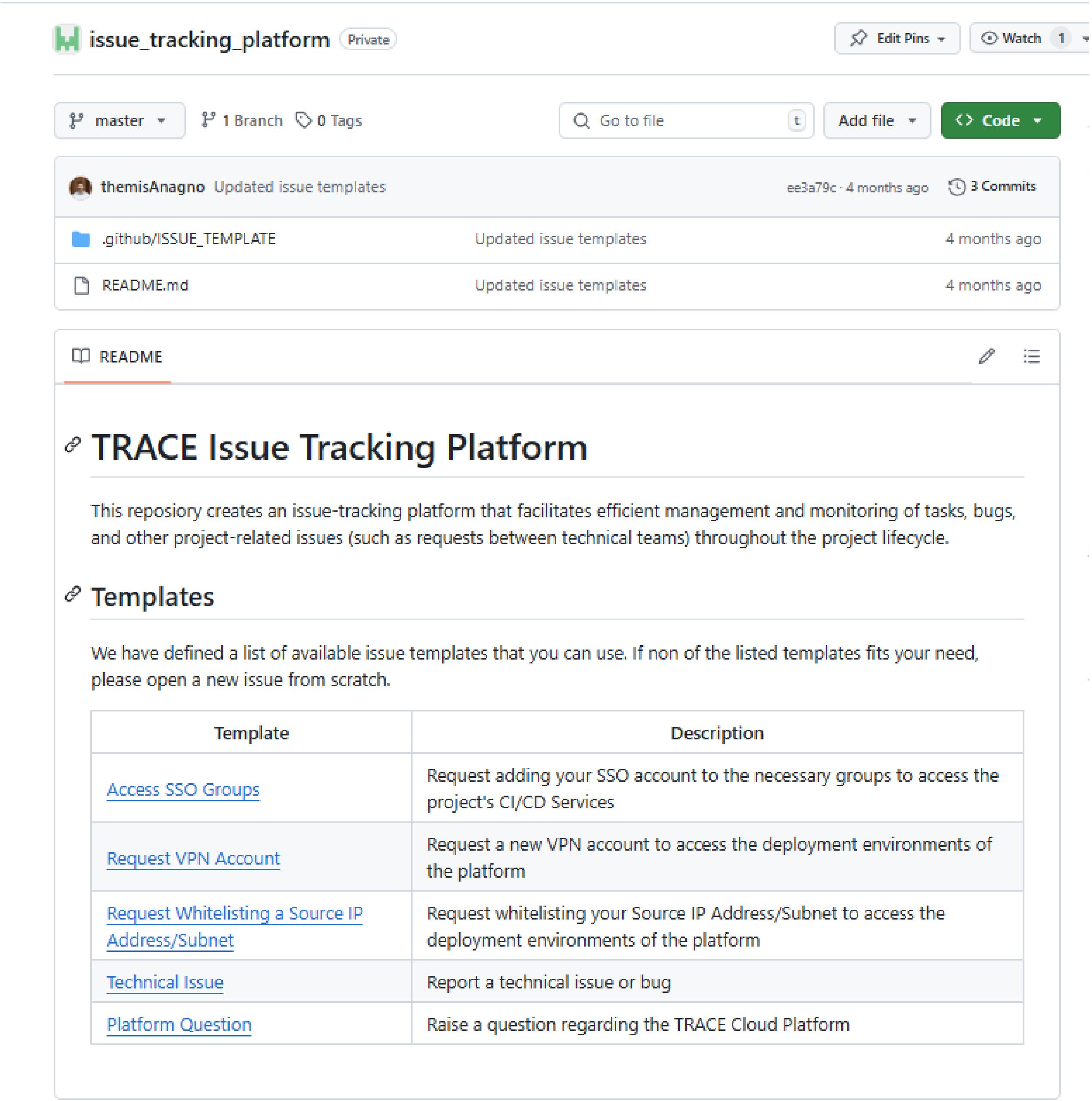
- **Platform validation and finalisation (Months 33-36)**: during this phase, the TRACE technical teams will continuously receive feedback from the platform end-users. Based on this feedback, they will proceed with bug fixes and feature updates on the involved components and services. This phase will end with the completion of the project and the last Milestone of the final integrated platform release.

4.2 Development and Integration Issue Tracking

An issue-tracking platform has been deployed and configured to record, manage, and monitor issues that arise during the development and integration phases of the project. This platform facilitates efficient management and monitoring of tasks, bugs, and other project-related issues (such as requests between technical teams) throughout the project lifecycle.

The platform is implemented in the dedicated GitHub repository shown in Figure 12. GitHub Issues allows platform developers and administrators to create detailed issue tickets for bugs, feature requests, or other tasks. These tickets include a title, description, and optional markdown formatting to add checklists, code snippets, or images for context. Issues can also be linked directly to specific lines of code in a TRACE component's repository, making it easy to associate bugs with their source. Open Issues are marked with specific labels that are useful for more efficient organisation and prioritisation.

The repository uses the issue templates shown in Figure 12 to guide contributors in providing necessary details when reporting issues or suggesting features, ensuring consistent and high-quality reports. Issues created through these templates are automatically assigned to the appropriate consortium member who is responsible for resolving the issue. Automatic notifications keep everyone informed about updates, comments, or changes to assigned tasks.



themisAnagno Updated issue templates ee3a79c · 4 months ago 3 Commits

.github/ISSUE_TEMPLATE	Updated issue templates	4 months ago
README.md	Updated issue templates	4 months ago

TRACE Issue Tracking Platform

This reposiory creates an issue-tracking platform that facilitates efficient management and monitoring of tasks, bugs, and other project-related issues (such as requests between technical teams) throughout the project lifecycle.

Templates

We have defined a list of available issue templates that you can use. If non of the listed templates fits your need, please open a new issue from scratch.

Template	Description
Access SSO Groups	Request adding your SSO account to the necessary groups to access the project's CI/CD Services
Request VPN Account	Request a new VPN account to access the deployment environments of the platform
Request Whitelisting a Source IP Address/Subnet	Request whitelisting your Source IP Address/Subnet to access the deployment environments of the platform
Technical Issue	Report a technical issue or bug
Platform Question	Raise a question regarding the TRACE Cloud Platform

Figure 12: Issue Tracking Platform

5 Platform Testing and Validation

The objective of the Test and Validation process is to confirm the successful deployment of TRACE components and to evaluate their functionality on the TRACE platform.

To achieve this, we have created a Validation Matrix for each component under test, which includes the corresponding requirement name and the test result. These matrices are grouped according to the modules they belong to. Additionally, a separate table has been developed to consolidate and analyse the test results for each component, providing an overview of the overall status of each component as well as the entire platform, as seen in Table 4 below.

Table 4: TRACE Platform modules validation & testing

Module	Component	Short Name	Task	Lead	Requirements
BlockChain	Distributed Ledger Structure	DLT	T4.5	BC5/SID/UGLA	Blockchain
	Digital Wallet	SSID	T4.5	BC5/SID/UGLA	Blockchain
	PKI Ecosystem	PKI	T4.5	BC5/SID/UGLA	Blockchain
	Smart Contract	SC	T4.5	BC5/SID/UGLA	Blockchain
Event handler	Resources Monitoring and Events Manager	RMEManager	T4.3 / T4.4	UTH	Event Monitoring & Optimisation
	Event Management Module	EMModule	T4.3	UTH	Event
	Monitoring Module	MOModule	T4.3	UTH	Event Monitoring & Optimisation
	Fleet Monitoring Manager	FMManager	T4.3	NKUA	Vehicles & Sensors Event Monitoring & Optimisation
	Mitigation Manager	MIManager	T4.3	UTH/CERTH	Event
Interopera-bility layer	TRACE Semantic Framework - Data Model	TSFDM	T3.2	NKUA	Platform Data Management
	Input Transformer	InTransform	T3.2	UGLA/UTH /CERTH	Cloud Infrastructure Data Management
	Output Transformer	OutTransform	T3.2	UGLA/UTH /CERTH	Cloud Infrastructure Data Management

Module	Component	Short Name	Task	Lead	Requirements
	Graphical User Interface (GUI) for logistics companies	GUI	T3.2	UTH	Cloud Infrastructure Data Management GUI
Scheduler & Route Optimizer	Scheduler	Scheduler	T4.4	UTH/TUC /CERTH	Monitoring & Optimisation Events
	Route Optimizer	Route Optimizer	T4.4	TUC/CERTH /UTH/UNISYS	Monitoring & Optimisation
Stream-Handler	Stream-Handler		T3.3	INTRA	Cloud Infrastructure Data Management
Trace Storage & Encryption	Data encryption	DataStoreCrypt	T4.6	SID	Cloud Infrastructure Data Management
	Cloud-based System	CDMS	T3.3	INTRA	Cloud Infrastructure Data Management
User Interface	Virtual Cockpit	Virtual Cockpit	T4.7	CDW/NKUA	Virtual Cockpit GUI Reporting & Analytics
	GUI	GUI	T3.5		GUI Reporting & Analytics
	API GateWay	API GW	T3.5	INTRA	Reporting & Analytics
	Monitoring and Logging Infrastructure	MonLog	T3.5	INTRA	Reporting & Analytics
Vehicle Support Services	Smart Bike Autonomous Driving System	SBADS	T4.1	UNIMORE	Vehicles & Sensors
	V2V com security	V2VcomSEC	T4.6	CSEM	Vehicles & Sensors
	Secure boot and firmware update for the V2V communication nodes	V2VcomBOOT	T4.6	CSEM	Vehicles & Sensors
	V2V low power communication System	V2VlpCOM	T3.4	CSEM	Vehicles & Sensors
	RB-VOGUI Interface	RB-VOGUI Interface	T4.2	ROBOTNIK	Vehicles & Sensors
	DIFLY LifeBox		T4.1	DIFLY	Vehicles & Sensors

Module	Component	Short Name	Task	Lead	Requirements
	Intrusion Detection Module and Events Management	IDM	T4.6	UNISYS	Safety & Reliability
	Bike Connection Box	BCB	T4.1		
	Low-power vision system for 3D scene reconstruction & obstacle detection	VisionAI 3D	T4.1	CSEM	Vehicles & Sensors
Vehicles	Cargo Bike		T4.1	OLV/UNIMORE	Vehicles & Sensors Safety & Reliability
	DIFLY Xplora2 UAS		T4.1	DIFLY	Vehicles & Sensors Safety & Reliability
	RB-VOGUI		T4.1	ROBOTNIK	Vehicles & Sensors Safety & Reliability
Platform Interface	Sink Manager		T4.1	NKUA	Vehicles & Sensors Safety & Reliability
	Optimisation module		T4.1	NKUA	

5.1 Benefits of Requirement-Based Testing

Requirement-Based Testing (RBT) is a critical methodology in software and hardware development and quality assurance. By aligning testing activities directly with the specified requirements, RBT ensures that the final product meets user needs and business goals.

The main benefits of adopting RBT:

- ***Alignment with requirements***: all testing efforts are directly aligned with the documented requirements. This alignment guarantees that the software product fulfills all specified functional and non-functional requirements.
- ***Traceability:*** Requirement Traceability Matrix (RTM) provides a clear mapping between each requirement and its corresponding tests. This comprehensive traceability ensures complete test coverage, making it easy to identify any requirements that may not have been tested.

- ***Early defect detection and risk mitigation*****:** Analysing requirements early in the development lifecycle facilitates the early identification of ambiguities, inconsistencies, and missing requirements. Detecting and addressing these issues early reduces the cost and effort associated with fixing defects later in the development process.
- ***Regulatory compliance***: RBT helps ensure compliance by validating that all specified requirements, including regulatory and safety requirements, are met.
- ***Enhanced user satisfaction****:* By ensuring that functional and non-functional requirements are met, RBT leads to higher user satisfaction.
- ***Efficient resource utilisation***: Focusing testing efforts on the specified requirements ensures that resources are used efficiently.

In summary, requirement-based testing ensures that the software product aligns with the specified requirements, leading to higher quality, improved user satisfaction, better risk management, and efficient use of resources. This methodical approach helps in delivering a reliable and compliant product that meets the business and user needs.

5.2 Codification

To facilitate the testing and validation process, we have established a naming convention for the component tests. This codification ensures consistency and clarity across all test names, making it easier to identify and track each test throughout the validation process. Each test name will follow a predefined format: **VALIDATION_METHOD_TYPE_MODULE_PRODUCT_XXX.** Below is a detailed explanation of the codification scheme:

- **VALIDATION** refers to the method used for validating the component. It can be:
  - **TEST**: if the component will be tested individually or in conjunction with another component.
  - **DEM**: if the validation will occur during the demonstrator phase with the entire system.
- **METHOD** refers to the methodology employed to validate the component. It can be:
  - **MAN**: if the validation will be conducted manually.
  - **AUTO**: if the validation will be performed automatically using tools like Jenkins.
- **TYPE** refers to the type of requirement being tested. It can be:

    - **F**: if it tests a functional requirement.
    - **N**: if it tests a non-functional requirement.
    - **I**: if it tests the interaction between the component and the TRACE platform.
- **MODULE** refers to the module where the component resides.
- **PRODUCT** refers to the specific component that you are testing.
- **XXX** serves as a numerical identifier to distinguish between multiple tests performed on the same component.

As an example, to name a test where we want to manually validate whether the RB_VOGUI robot, which belongs to the Vehicles module, meets a functional requirement, we will use: TEST_MAN_F_VEHICLES_RB-VOGUI_001.

By adhering to this standardised format, we will streamline the organisation and analysis of test results, improve communication among team members, and ensure consistency across all components.

5.3 Components tests validation matrix

Each component has a validation matrix, as a separate file, that comprehensively details all the tests required to verify that the component meets its specified requirements. This file provides a structured overview of the testing process and includes the following columns:

- **Responsible**: name of the partner who is responsible for the component.
- **Validation Type**: the method used for validating the component. It can be:
    - **Test**: if it will be tested individually or in conjunction with another component.
    - **Demonstrator**: if the validation will occur during the demonstrator phase with the entire system.
- **Execution Method**: the methodology that will be used to validate the component. It can be:
    - **Manual**: if it will be validated manually.
    - **Automated**: if it will be validated automatically using tools like Jenkins.
- **Test / Demo ID**: the unique identifier (codification name) for that component.
- **Component**: the name of the component that is being tested. It should be the same as the file name.
- **Scenario**: the pilots where it is going to be tested (e.g., Italy, Greece, Slovenia).

- **Requirement Covered**: the name of the requirement that this test addresses.
- **Description of Fit Criterion**: a description of the success criteria for the test.
- **Initial Conditions**: the initial state of the component.
- **Final Conditions**: the final state that the component must/should/may achieve.
- **Step Description**: a detailed definition of the steps to follow to meet the requirements.
- **STATUS**: the current status of the test. It can be:
  - NO RUN: if the test has not started yet.
  - RUNNING: if the test is in progress.
  - BLOCKED: if the test has stopped due to external conditions.
  - PASSED: if the test has been completed successfully. (finished successfully)
  - FAILED: if the test has been finished with a failure result.
- **NOTES**: additional notes that are considered as necessary.

5.4 Validation matrices

In this subsection, all the established test and validation files are compiled, each of which captures the platform's requirements in a separate matrix. In this initial version, the tests defined correspond to the components developed within each module of the architecture. For each test, a detailed definition is provided, outlining the aspects to be evaluated in subsequent versions. While this version focuses on individual components, a future iteration will implement a comprehensive global test to assess the architecture's performance as a fully integrated system, combining all the components currently under evaluation.

5.4.1 Blockchain

5.4.1.1 Digital Wallet

Table 5: Digital Wallet component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
BC5	Test	Manual	TEST_MAN_F_BLK_BC5_SEC-KEY_001	BC5-BLK-SEC-KEY	Greek, Slovenian, Italian	BLK-FUN-004-Secure Storage of Public Keys	TRACE should enhance security by securely storing public keys associated with logistics companies offline, used for verifying transactions	User registration with TRACE. User will be assigned an Blockchain base wallet	Wallet Public key will get store in Trace central Database	1	User starts the process of registration. In the process, the wallet will get create automatically and its private key will be shared with user only.
										2	Trace will store the public address of the wallet into the secure database.
BC5	Test	Manual	TEST_MAN_F_BLK_BC5_IGT-CON-BLK-TNX_002	BC5-BLK-CON-BLK-TNX	Greek, Slovenian, Italian	BLK-PRM-005-Integrity and Consensus in Blockchain Transactions	The TRACE shall ensure the integrity and consensus of transactions, and wallets public keys, providing trust in the system.				

5.4.1.2 Distributed Ledger Structure

Table 6: Distributed Ledger Structure component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
BC5	Test	Manual	TEST_MAN_F_BLK_ BC5-TNX-VER_001	BC5-DNFT	Greek, Slovenian, Italian	BLK-FUN-003-Transaction ID Capture and Verification	TRACE should capture and retrieve the transaction ID from the Algorand blockchain once a transaction has been processed, facilitating tracking and verification.	Transaction will get initialise	Data will be shared with the platform	1	A smart contract will get initiated.
										2	On receiving the response, status of the process with get captured
										3	Transaction will be retrieved from the from response and will get store in the blockchain
BC5	Test	Manual	TEST_MAN_F_BLK_BC5-SHD-MNG_002	BC5-BLK-SDH-MNG	Greek, Slovenian, Italian	BLK-FUN-007 - Blockchain-Based Scheduling Management	TRACE will automate and secure logistics operations by managing collaborative scheduling agreements on the blockchain.	Scheduler will pass the collaboration agreement hash to the blockchain	hash will get store into the Dynamic NFT meta data.	1	Collaboration agreement (CA) will get prepared by scheduler
										2	Hash of the CA will get store in

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											key value pair on respective Dynamic NFT
BC5	Test	Manual	TEST_MAN_F_BLK_BC5-IM-TNX_REC_003	BC5-BLK-IM-TNX-REC	Greek, Slovenian, Italian	BLK-FUN-006-Immutable Transaction Records	TRACE should maintain a transparent and immutable record's transactions Id in centralised storage which is accessible via the platform interface	Waiting on transaction status	Made the record accessible via platform	1	TRACE will map the transaction ID with appropriate shipment record.
BC5	Test	Manual	TEST_MAN_NF_BLK_BC5-BNP_004	BC5-BLK-BNP	Greek, Slovenian, Italian	BLK-PRM-001-Blockchain Network Performance Optimization	The system shall optimise the performance of the blockchain network to ensure timely execution of transactions and smart contracts.				

5.4.1.3 PKI Ecosystem

Table 7: PKI Ecosystem component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
BC5	Test	Manual	TEST_MAN_F_BLK_BC5-AUTH_001	BLK-FUN-002	Greek, Slovenian, Italian	BLK-FUN-002 - User Authentication on Blockchain	TRACE will seamlessly trigger blockchain transactions upon successful login, enhancing automation and	Waiting for user to interact	Authentication status will be shared with TRACE	1	User will interact with authentication using front end

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
							security in logistics operations.				
										2	Automatic transaction will get triggered, and details will be shared with TRACE
BC5	Test	Manual	TEST_MAN_NF_BLK_BC5-AUTH-SEC-002	BLK-PRM-009	Greek, Slovenian, Italian	BLK-PRM-009-Enhancing Authentication Security with Blockchain	The TRACE should utilise the immutability and decentralised nature of the Algorand blockchain to enhance the security of the authentication process.				

5.4.1.4 Smart Contract

Table 8: Smart Contract component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
BC5	Test	Manual	TEST_MAN_F_BLK_BC5-DNFT_001	BC5-DNFT	Greek, Slovenian, Italian	BLK-FUN-001 - dNFT Smart Contract	TRACE will update the status of the shipment using smart contracts and dNFTs, ensuring up-to-date	Create Dynamic NFT Data	Store final details of the shipment and other	1	The smart contract API will get the data from the scheduler.

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
							and accurate record of information.		document in decentralised storage		
										2	Data will get parse into json format. parsed json will be inscribe in dynamic NFT meta data
										3	Dynamic NFT will get created on decentralized IPFS store and on Blockchain.
BC5	Test	Manual	TEST_MAN_F_BLK_BC5-GEN-DNFT_002	BC5-GEN-DNFT	Greek, Slovenian, Italian	BLK-FUN-005-dNFT Generation for Logistics Assets	TRACE should generate a unique dNFT for each shipment/parcel, allowing digital tracking and management of logistic assets	Create Dynamic NFT ID	Assign the Dynamic NFT to the shipment	1	Create Dynamic NFT on Blockchain
										2	Get Id of dNFT and assign it to shipment
BC5	Test	Manual	TEST_MAN_F_BLK_BC5-LOG-EVNT_003	BC5-LOG-EVNT	Greek, Slovenian, Italian	BLK-FUN-009-Logging Significant Events on Blockchain	TRACE may log all significant events related to the delivery process on the	Get update on a shipment and retrieve the Dynamic	Update the Dynamic NFT data	1	Retrieve Id of the existing dNFT.

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
							blockchain, ensuring that all actions are transparent and immutable	NFT ID to Update	with latest update		
										2	Update the existing Dynamic NFT data with the latest one.

5.4.2 EventHandler

5.4.2.1 Event Management Module

Table 9: Event Management Module component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UTH	Test	Manual	TEST_MAN_F_EVENTHANDLER_EMMODULE_001	Event Management Module	All	EVT-FUN-001: Events should be generated when a) a shipment is loaded onto a vehicle, b) when the shipment is delivered to the platform or logistics company's system in real-time	Real-time event generation when a shipment is loaded or delivered, with alert logs accessible in TRACE platform	Shipment data available, vehicle loading initiated	Event generated, recorded, and accessible in real time upon loading or delivery	1	Trigger event when shipment is loaded to vehicle
										2	Trigger event when shipment is delivered to

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											designated platform
UTH	Test	Manual	TEST_MAN_F_EVENTHANDLER_EMMODULE_002	Event Management Module	All	EVT-FUN-004: TRACE platform shall be capable of detecting various types of events, such as delays, disruptions, or anomalies, in real-time	Real-time detection and alerting of events like delays, disruptions, and anomalies	System monitoring enabled on all incoming data streams	Alerts generated for each detected event type, accessible in real time	1	Analyse incoming data for event markers
UTH	Demonstrator	Automated	DEM_AUTO_F_EVENTHANDLER_EMMODULE_003	Event Management Module (EMM)	All	EVT-FUN-002: Event Management Module shall integrate with other Modules	We will ensure the proper functionality of EMM and the proper integration with other components	The Event Management Module is deployed and initialised, and other components are operational and awaiting interaction.	The components must integrate properly with EMM	1	Test and verify communication and data exchange between EMM and other TRACE components.
UTH	Test	Manual	TEST_MAN_F_EVENTHANDLER_EMMODULE_004	Event Management Module	All	EVT-FUN-013: When an unmanned vehicle has a mechanical or electronic breakdown, TRACE shall trigger a fault event	Fault event should be triggered when unmanned vehicle has a mechanical or electronic breakdown.	Vehicle has a mechanical or electronic breakdown	Warning generated and alert sent to TRACE	1	Event Management Module identifies the fault.
										2	Verify that the fault event is generated and

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											sent to the TRACE platform as an alert.
UTH	Test	Manual	TEST_MAN_F_EVENTHANDLER_EMMODULE_005	Event Management Module	All	EVT-FUN-010: TRACE platform should use machine learning algorithms to identify patterns in event occurrences	The identification of patterns in event occurrences using machine learning will be checked.	Identification of patterns in event occurrences	Warning generated and alert sent to TRACE	1	Train the machine learning algorithms to recognise patterns
UTH	Test	Manual	TEST_MAN_F_EVENTHANDLER_EMMODULE_006	Event Management Module	All	EVT-FUN-005: TRACE platform shall prioritise events based on predefined criteria to ensure appropriate response and resource allocation	Prioritise events based on predefined criteria to allocate resources effectively	Criteria for event prioritisation defined and stored	Prioritised event list and response plan triggered	1	Assign priority based on event type and criteria

5.4.2.2 Fleet Monitoring Manager

Table 10: Fleet Monitoring Manager component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UTH	Test	Manual	TEST_MAN_F_EVENTHANDLER_FMMANAGER_001	Fleet Monitoring Manager	All	EVT-FUN-012: When an unmanned vehicle approaches the limits of its	Warning triggered when unmanned vehicle approaches operating area limits	Vehicle within proximity to area limit	Warning generated and alert sent to TRACE	1	Track vehicle's location for boundary proximity

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						authorised operating area, TRACE shall trigger a warning event					
NKUA	Test	Manual	TEST_MAN_F_EVENTHANDLER_FMMANAGER_002	Fleet Monitoring Manager	All	MON-FUN-005: Platform may integrate with maintenance management systems	To optimising fleet availability regarding a specified area	GPS data of vehicles	Feet availability	1	Filtering the fleet availability

5.4.2.3 Mitigation Manager

Table 11: Mitigation Manager component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UTH	Test	Manual	TEST_MAN_F_EVENTHANDLER_MIMANAGER_001	Mitigation Manager	All	EVT-FUN-006: TRACE platform shall provide notification and alerting capabilities to inform users about important events, updates, or changes in transportation operations, ensuring timely	Send notifications and alerts for major events to ensure timely response by users	User notification configurations set	Alerts sent to users promptly on relevant events	1	Configure alert and notification parameters

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						awareness and response					
UTH	Test	Manual	TEST_MAN_F_EVENTHANDLER_MIMANAGER_002	Mitigation Manager	All	EVT-PRM-001: TRACE platform shall support automated responses to certain types of events, such as triggering re-allocations or updating schedules	Automated responses enabled for specific events, such as schedule updates or reallocations	Automation rules configured and system initialised	Responses triggered automatically as per configuration	1	Define automation rules for event response
UTH	Test	Manual	TEST_MAN_F_EVENTHANDLER_MIMANAFER_003	Mitigation Manager	All	EVT-FUN-011: When a low-battery alert event is raised from an unmanned vehicle, TRACE should force it to return to consolidation center for a battery charging or switching.	Mitigation manager should apply an appropriate plan to return the vehicle to consolidation center for a battery charging or switching.	Vehicle has a mechanical or electronic breakdown	Warning generated and alert sent to TRACE	1	When a low-battery event takes place the Mitigation Manager triggered and make the vehicle to return to the consolidation center and logs the battery charging or switching

5.4.2.4 Monitoring Module

Table 12: Monitoring Module component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UTH	Test	Manual	TEST_MAN_F_EVENTHANDLER_MOMODULE_001	Monitoring Module	All	EVT-FUN-005: TRACE platform shall prioritise events based on predefined criteria to ensure appropriate response and resource allocation	Prioritise events based on predefined criteria to allocate resources effectively	Criteria for event prioritisation defined and stored	Prioritised event list and response plan triggered	1	Assign priority based on event type and criteria
UTH	TEST	Manual	TEST_MAN_N_EVENTHANDLER_MOMODULE_002	Monitoring Module	All	MON-PRM-001: TRACE platform may support scenario modeling and what-if analysis to evaluate the impact of different operational strategies, route optimisations, or resource allocations on performance metrics	Scenario modeling and what-if analysis should be implemented for the evaluation of the impact of various operational strategies, route optimisations or resource allocations on performance metrics.	Monitoring Module is operational and receives real-time data inputs from the TRACE platform.	Monitoring Module is operational and receives real-time data inputs from the TRACE platform.	1	Simulate operational strategies, route optimisations, and resource allocations, then analyse the impacts on performance metrics.

5.4.2.5 Resources Monitoring and Events Manager

Table 13: Resources Monitoring and Events Manager

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UTH	Test	Automated (Jenkins)	TEST_AUTO_I_EVENTHANDLER_RMEMANAGER_001	Resources Monitoring and Events Manager	All	EVT-FUN-009: TRACE platform should have an adequate track and trace system of events available in real time.	REManager processes and updates events data received in real time.	Real-time data received from sensors and EMModule.	Updated allocations/routes sent to Scheduler	1	Simulate real-time data updates from sensors and events through EMModule.
										2	REManager triggers Scheduler for new allocations or route changes based on updated data.
										3	Confirm that the updated allocations/routes are logged for traceability.
UTH	Test	Manual	TEST_AUTO_I_EVENTHANDLER_RMEMANAGER_002	Resources Monitoring and Events Manager	All	EVT-FUN-014: TRACE platform shall operate area limit fault events.	REManager detects out-of-bounds events and triggers mitigation actions	Vehicle leaves its designated operating area.	Fault event detected and mitigation actions triggered.	1	REManager forwards the event to the Scheduler for route adjustments or reallocation.
										2	Verify the mitigation plan

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											is logged and executed appropriately.

5.4.3 Interoperability layer

5.4.3.1 Graphical User Interface

Table 14: Graphical User Interface component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_001	Graphical User Interface (GUI) for logistics companies	All	DM-FUN-001: TRACE platform shall integrate data from different sources and formats to provide a unified view of the transportation system	TRACE platform shall integrate data from different sources and formats to provide a unified view of the transportation system	TRACE platform running with access to multiple data sources in various formats	TRACE platform displays a unified view incorporating all integrated data	1	Verify that the GUI displays a unified, cohesive view with data from multiple sources, all accessible through the TRACE platform
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_002	Graphical User Interface (GUI) for logistics companies	All	DM-FUN-002: TRACE platform should support both structured and unstructured data	TRACE platform should support both structured and unstructured data	TRACE platform is connected to sources providing both structured and unstructured data formats	TRACE platform's GUI displays both types of data seamlessly, with no data loss or format issues	1	Verify that the GUI correctly displays both structured and unstructured data from connected sources, ensuring data

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											consistency and completeness
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_003	Graphical User Interface (GUI) for logistics companies	All	INT-FUN-001: GUI should incorporate predictive analytics capabilities	GUI should incorporate predictive analytics to forecast trends, demand, and performance based on historical data.	TRACE platform is connected to data sources and has historical data available	Predictive analytics capabilities are accessible in the GUI for proactive decision-making	1	Verify that the GUI displays predictive analytics forecasts based on historical data, aiding decision-making
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_004	Graphical User Interface (GUI) for logistics companies	All	INT-FUN-002: GUI may incorporate collaborative features	GUI should offer shared workspaces, comments, and notifications for teamwork and communication.	GUI is operational, and multiple user accounts are available for testing collaboration features	Collaborative tools (workspaces, comments, notifications) are accessible in the GUI	1	Verify that users can access and utilise shared workspaces, comment features, and notifications within the GUI
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_005	Graphical User Interface (GUI) for logistics companies	All	INT-FUN-006: GUI should support different levels of users	GUI should offer at least 3 user roles: Requestor, Executor, and Admin.	TRACE platform is operational with role-based access control enabled	GUI allows users with different roles to access appropriate data and functions	1	Verify that each user role (Requestor, Executor, Admin) has correct access to GUI features and data
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_011	Graphical User Interface (GUI) for logistics companies	All	INT-FUN-010: TRACE platform shall be scalable for growing data	TRACE platform should scale as data volume and user numbers increase.	TRACE platform is set to monitor performance	GUI performance remains stable and responsive	1	Verify that the TRACE platform GUI maintains performance

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						volumes and user loads		with increasing data loads	under increased data volumes and user loads		and responsiveness as data volumes and user loads grow
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_012	Graphical User Interface (GUI) for logistics companies	All	INT-FUN-011: TRACE shall allow recipients to book real-time/same-day deliveries	GUI should offer a booking feature for recipients to schedule real-time/same-day deliveries.	TRACE platform operational with booking functionality enabled	GUI allows recipients to book real-time/same-day deliveries and manages vehicle allocation accordingly	1	Verify that the GUI enables recipients to book same-day deliveries, and that vehicle allocation updates accordingly

5.4.3.2 Input Transformer

Table 15: Input Transformer component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UGLA	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_013	User Interface	All	INT-FUN-013: Secure Login and Data Upload	GUI should offer secure login through blockchain wallet and upload data	Data availability from stakeholders	Integrated data	1	A user/ software push data to the platform using the Southbound API.

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										2	Interface validates the user through blockchain API
UGLA	Demonstration		DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_014	User Interface	All	INT-FUN-014: Data Integrity Checking and Parsing	Input transformer carries out data integrity checking and parsing	Data availability from stakeholders	Integrated data	1	Transforms carries out different integrity checks to verify that correct types of data is uploaded
										2	The uploaded data is parsed and stored in backend engines depending on the type of data

5.4.3.3 Output Transformer

Table 16: Output Transformer component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UGLA	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY	User Interface	All	INT-FUN-015: Data Transformation	Output transformer carries out any	Data stored in the backend engines	User requested data	1	Output transformer converts user

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
			_LAYER_GUI_015				relevant data transformation				uploaded data to specific format depending on the downstream tasks
										2	Output transformer allows stakeholders to securely modify and download the data

5.4.3.4 TRACE Semantic Framework – Data Model

Table 17: TRACE Semantic Framework – Data Model component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
NKUA	Test	Manual	TEST_MAN_F_PLT_TSFDM_001	TRACE Semantic Framework - Data Model (TSFDM)	Greek, Italian, Slovenian	PLT-FUN-012 - TRACE platform shall support interoperability with hardware,	To validate the requirement: Capability to retrieve data from logistic companies	Data availability from logistic companies	Integrated data	1	A user/ software requests data from the TSFusing the

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						software, and external APIs					Northbound API.
										2	The request is translated int several queries for each underlying data source
										3	The data is collected from all sources
										4	The collected data is merged
										5	The collected data is returned as a response to the user/ module made the request
NKUA	Test	Manual	TEST_MAN_F_PLT_TSFDM_002	TRACE Semantic Framework - Data Model (TSFDM)	Greek, Italian, Slovenian	DM-FUN-001 - TRACE platform shall integrate data from various sources for a unified view	To validate the requirement: Capability to retrieve data from logistic companies	Data availability from logistic companies	Integrated data	1	A user/software requests data from the TSFusing the Northbound API.
										2	The request is translated int

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											several queries for each underlying data source
										3	The data is collected from all sources
										4	The collected data is merged
										5	The collected data is returned as a response to the user/module made the request

5.4.4 Scheduler and Route Optimiser

5.4.4.1 Route Optimiser

Table 18: Route Optimiser component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
TUC	Test	Manual	TEST_MAN_F_SCHEDULER_ROUTE_OPTIMIZER_001	Route Optimizer	All	PLT-FUN-006 - TRACE platform should enable the planning of necessary measures to prevent or mitigate the negative impacts of future events	The Route Optimiser provides alternative routes when a road is closed in an urban environment without changing the allocation of the shipments to vehicles.	An optimal route is calculated without considering any closed roads	A route is recalculated after excluding a road segment as no go zone to represent that it is closed.	1	The Route Optimiser calculates the best route for the vehicle
										2	Recalculate the optimal route by excluding a specified road to represent that it is closed
TUC	Test	Manual	TEST_MAN_F_SCHEDULER_ROUTE_OPTIMISER_002	Route Optimiser	All	MON-FUN-010 - When generating trajectories and paths for unmanned vehicles, TRACE shall generate trajectories and paths crossing exclusively the authorised operating areas	The produced trajectories do not cross any no-fly or no-go zones, and stay within the boundaries of the operational area	The boundaries of the operational area alongside the boundaries of the no-fly and no-go zones are provided to the Route Optimiser	The route optimiser returns the computed routes	1	Provide the polygons of the operational area alongside the polygons of no-go zones and no-fly zones

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										2	Route optimiser returns the appropriate routes

5.4.4.2 Scheduler

Table 19: Scheduler component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
TUC	Demonstration	Manual	DEM_MAN_I_SCHEDULER_SCHEDULER_001	Scheduler	All	PLT-FUN-008 - TRACE platform shall dynamically adjust resource allocations, such as vehicle assignments or route schedules, based on real-time performance data and predictive insights to optimise efficiency and responsiveness	The Scheduler successfully, assigns shipments to vehicles and vehicles to routes by updating TRACE's database and Blockchain	A request is forwarded to the Scheduler	Scheduler completes all the procedures following the request	1	Retrieve Data from the Interoperability layer
										2	Calculate appropriate Schedules

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										3	Update TRACE's storage, Blockchain, and Logistics Partners' databases through Interoperability Layer
TUC	Test	Manual	TEST_MAN_N_SCHEDULER_SCHEDULER_002	Scheduler	All	MON-PRM-002 - TRACE platform shall consider cost optimisation objectives alongside performance metrics, balancing operational efficiency with cost-effectiveness to maximise overall value and profitability	The Scheduler adapts the optimal solution as the users selects different values that emphasise the importance of delivery time and cost	The Scheduler receives multiple identical requests except from the importance of factors	The Scheduler sends the calculated Schedule and Routes for approval to the UI.	1	The Scheduler receives the nearly identical requests and calculates the respective optimal schedule and the routes
										2	The Scheduler provides the calculated routed and schedules

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
TUC	Test	Auto	TEST_AUTO_N_SCHEDULER_SCHEDULER_003	Scheduler	All	MON-PRM-003 - TRACE platform should allow users to define performance thresholds and alerting rules to notify them when performance metrics exceed or fall below acceptable levels, triggering proactive intervention	The Scheduler returns a notification alongside the schedule for approval and the calculated routes if the solution exceeds specified thresholds	The user specifies within the request thresholds regarding the cost of delivery and delivery time	The Scheduler sends the calculated Schedule and Routes for approval to the UI.	1	The Scheduler receives the request with the provided thresholds and calculates the optimal schedule and the routes
										2	The Scheduler returns also a notification if performance indices exceed the provided threshold
TUC	Test	Manual	TEST_MAN_N_SCHEDULER_SCHEDULER_004	Scheduler	All	RA-PRM-001 - TRACE should provide comprehensive performance tracking	The Scheduler reports the optimal cost using a breakdown list of components that affect the choice of an optimal solution	A request is sent to the Scheduler	The Scheduler sends the calculated routes for approval with a cost analysis with the specified KPIs	1	The Scheduler receives the request with the provided thresholds and calculates the optimal schedule and the routes

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										2	The Scheduler sends the calculated routes and schedule for approval alongside an analytics breakdown of the specified KPIs
TUC	Test	Manual	TEST_MAN_N_SCHEDULER_SCHEDULER_005	Scheduler	All	RA-PRM-002 - TRACE may allow benchmarking and performance comparison	The Scheduler uses the report analytics and the selection parameters to find optimal solutions according to benchmark criteria and compare them with the specified user's request	A request is sent to the Scheduler	The Scheduler sends the calculated routes for approval with a cost analysis with the specified KPIs	1	The Scheduler receives the request with the provided thresholds and calculates the optimal schedule and the routes
										2	The Scheduler provides a comparison between the benchmarked solution and the calculated one

5.4.5 StreamHandler

5.4.5.1 StreamHandler

Table 20: StreamHandler component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
INTRA	Test	Auto (Jenkins)	TEST_AUTO_F_SH_001	StreamHandler (SH)	All		This test will validate that the StreamHandler is successfully deployed through Jenkins pipelines and operates as expected.	The SH is not deployed.	The SH is deployed on multiple VMs on the cloud.	1	Trigger the Jenkins pipelines to deploy the core SH services
										2	Check the health status of the Dockerised services
										3	Execute a GET API cCloud and ensure the response code is 200
INTRA	Test	Auto (Jenkins)	TEST_AUTO_F_SH_002	StreamHandler (SH)	All	PLT-FUN-012 COM-FUN-016 DM-FUN-002 DM-PRM-003 DM-PRM-005	This test will validate that other services can send and receive messages on a testing StreamHandler topic.	The SH is deployed on multiple VMs on the cloud.	Messages are exchanged through a testing topic.	1	Create a topic on SH named "testing"

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										2	On a Jenkins agent, create a producer, connect to the cluster, and send test messages on the testing topic.
										3	On another Jenkins agent, create a consumer, connect to the cluster, and read Cloud the messages on the testing topic. Ensure that the previously sent messages are included.
INTRA	Test	Manual	TEST_MAN_F_SH_003	StreamHandler (SH)	All	PLT-FUN-011 EVT-FUN-005 EVT-FUN-009	This test will validate that the StreamHandler admin console is working correctly.	The SH is deployed on multiple VMs on the cloud.	The admin web UI is reachable and works as expected.	1	A platform administrator is connected to the admin UI through a browser, and checks that everything is working as expected and

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											no errors are reported.
INTRA	Test	Manual	TEST_MAN_F_SH_004	StreamHandler (SH)	All	PLT-PRM-005 PLT-PRM-010 INT-PRM-008 CLD-PRM-005 CLD-PRM-006	This test will validate that the StreamHandler is fault-tolerant and highly available.	The SH is deployed on multiple VMs the cloud. A single VM goes down.	The SH is deployed on multiple VMs on the cloud. Its services are not affected.	1	Create a test local producer and consumer and send messages to the testing topic.
										2	Manually bring down a VM hosting the SH modules.
										3	Ensure that the messages continue to go through the cluster and are received by the consumer.
										4	Manually start again the VM.
										5	Ensure that the SH modules are automatically recovered once the VM gets started.

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
INTRA	Test	Manual	TEST_MAN_F_SH_005	StreamHandler (SH)	All	RA-FUN-003 SEC-PRM-003 DM-PRM-001	This test will validate that the StreamHandler will discard messages that exceed the specified retention period.	The SH is deployed on multiple VMs on the cloud. The data retention policy is set.	Messages that exceed the retention period are dropped.	1	Set the data retention period to one hour.
										2	Create a test local producer and consumer and send messages to the testing topic.
										3	Wait an hour and check the messages in the testing topic. Ensure that the sent messages are deleted.
INTRA	Test	Manual	TEST_MAN_F_SH_006	StreamHandler (SH)	All	SEC-FUN-001 SEC-FUN-002 SEC-FUN-003 SEC-PRM-002 SEC-PRM-003	This test will validate that only authorised users can access topics	The SH is deployed on multiple VMs on the cloud.	Unauthorised users cannot write/read to/from topics	1	Create local SH producers and consumers
										2	Try to write and read to/from a specific SH topic. Ensure

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											that an unauthorised message is received.

5.4.6 Trace Storage and Encryption

5.4.6.1 Cloud-based System

Table 21: Cloud-based system component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
INTRA	Test	Auto (Jenkins)	TEST_AUTO_F_CDMS_001	CDMS	All	PLT-FUN-011 DM-FUN-001 DM-FUN-002 DM-PRM-003 DM-PRM-005	This test will validate that the Database is successfully deployed through Jenkins pipelines and operates as expected.	The CDMS is not deployed on the cloud.	The CDMS is deployed on the cloud and accepts requests.	1	Deploy the Dockerised CDMS modules through Jenkins pipelines.
										2	Check the health status report of MongoDB, ensuring that it is healthy.
										3	Check the deployment status of the CDMS API GW.

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										4	Send an API call to ingest some test data into the system.
										5	Query and aggregate the test data. Check that the received data match the expected ones.
										6	Delete test data. Receive a successful deletion code.
INTRA	Test	Manual	TEST_MAN_F_CDMS_002	CDMS	All	PLT-PRM-005 PLT-PRM-010 INT-PRM-008 CLD-PRM-005 CLD-PRM-006	This test will validate that the CDMS is fault-tolerant and highly available.	The CDMS is deployed on the cloud and accepts requests. An instance of the CDMS goes down.	The CDMS is deployed on the cloud and accepts requests.	1	Manually bring down a VM hosting the CDMS modules.
										2	Continue to send API requests to the CDMS. Ensure that the requests are received successfully,

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											and the responses are as expected.
										3	Manually start again the VM.
										4	Ensure that the CDMS modules are automatically recovered once the VM gets started.
INTRA	Test	Auto (Jenkins)	TEST_AUTO_F_CDMS_003	CDMS	All	DM-PRM-004	This test will validate that the CDMS will recover from a disaster without any data loss, using a backup volume.	The CDMS is not deployed on the cloud.	The CDMS is deployed on the cloud and accepts requests. All the data have been restored from a backup volume.	1	Deploy the Dockerised CDMS modules through Jenkins pipelines. Attach to the CDMS module a test volume that has been used to store backup test data from a previously operational system.
										2	Check that the CDMS modules are up and running.

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										3	Send a LIST API call to the system to ensure that the CDMS contains some data.
										4	Send a GET API call to the system to retrieve the data. Ensure that the received data match the expected ones.
INTRA	Test	Manual	TEST_MAN_F_CDMS_004	CDMS	All	PLT-PRM-006 RA-FUN-003 SEC-FUN-002 SEC-FUN-003 SEC-PRM-002 SEC-PRM-003	This test will validate that only authorised users can access a database.	The CDMS is deployed on the cloud and accepts requests.	The CDMS is deployed on the cloud and accepts requests. A user receives unauthorised response when trying to access a DB where they have no permissions.	1	Create testing DB and user. Do not provide access permissions to the user.
										2	Using the new user, try to read from the

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											new database. Ensure that the response code 401 is received.
INTRA	Test	Manual	TEST_MAN_F_CDMS_005	CDMS	All	VS-FUN-016 DM-PRM-001	This test will validate that the CDMS is working as expected through the tool's admin console.	The CDMS is deployed on the cloud and accepts requests.		1	Connect to the platform admin console and ensure that all the DBs are operational

5.4.6.2 Data store encryption

Table 22: Data store encryption component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
SID	Test	Manual	TEST_MAN_DATASTORECRYPT_001	DataStoreCrypt	All	SEC-FUN-002: TRACE platform shall provide robust security features, such as role-based access control (RBAC), encryption (at rest and in transit), and multi-factor authentication	DataStoreCrypt provides robust security features and role-based access control as well as data encryption for data in transit and at rest through RBAC managed by KES	Component should have access to the data that will be encrypted as well as the storage and KES and the encryption keys should be created	Encrypted data stored in queue or database	1	Create key encryption key (KEK): create the key on KES service and store it in Hashicorp Vault which has an S3 Bucket as internal storage.

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										2	Encrypt data using KEK or DEK (Data encryption key). In transit send data to the appropriate KES endpoint for encryption. At rest ask for DEK from KES, decrypt DEK with appropriate KEK and then encrypt data using decrypted DEK.
										3	Store data in Kafka topic for in-transit and in database for at-rest
SID	Test	Manual	TEST_MAN_DATASTORECRYPT_002	DataStoreCrypt	All	SEC-FUN-003: TRACE system shall perform access control mechanisms to protect data integrity and prevent	DataStoreCrypt provides access control mechanisms to protect data integrity and unauthorised access	KES RBAC provides security regarding unauthorised decryption	Decrypted data	1	Use key Id and correct certificates to access the key on KES to decrypt the data. Use correct key id

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						unauthorised access					provided in Kafka message for decrypting message while using correct certificate to access KES endpoints
SID	Test	Manual	TEST_MAN_DATASTORECRYPT_003	DataStoreCrypt	All	SEC-PRM-001: TRACE platform shall implement encryption algorithms to secure personal data and business-sensitive data	DataStoreCrypt utilise encryption algorithms in order to secure personal data and business-sensitive data	Component should have access to the data that will be encrypted using AES encryption	Encrypted data stored in queue or database	1	Get encrypted DEK (data encryption key) from KES: Generate a DEK using KES endpoints. KES will return encrypted DEK which needs decryption for future usage
										2	Decrypt DEK using KEK: Decrypt DEK by using KES endpoints
										3	Encrypt data using decrypted DEK with AES encryption: Use the decrypted DEK in order to

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											encrypt the data and store them.

5.4.7 User interface

5.4.7.1 API GateWay

Table 23: API GateWay component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
INTRA	Test	Auto (Jenkins)	TEST_AUTO_F_APIGW_001	API GW	All	PLT-FUN-011 PLT-FUN-012	This test validates that the API GW is successfully deployed through Jenkins Pipelines	The API GW is not deployed.	The API GW is deployed on the cloud and is ready to accept requests.	1	Trigger a new deployment of the API GW via an automated Jenkins pipeline.
										2	Check the status of the Dockerised API GW modules and ensure that they are healthy.
										3	Send a request to a testing endpoint. Make sure that the response

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											matches the expected one.
INTRA	Test	Auto (Jenkins)	TEST_AUTO_F_APIGW_002	API GW	All	PLT-PRM-007 INT-PRM-008	This test validates that the API GW can communicate with the backend platform services.	The API GW is deployed on the cloud and is ready to accept requests.	The API GW has responded to the API calls.	1	As part of a pipeline, automatically send requests to the API GW that require communication with all the platform backend components. Ensure that the received the code is 200 for all the API calls.

5.4.7.2 GUI

Table 24: GUI component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_001	Graphical User Interface (GUI) for logistics companies	All	DM-FUN-001: TRACE platform shall integrate data from different sources and formats to provide a unified	TRACE platform shall integrate data from different sources and formats to provide a unified view of the	TRACE platform running with access to multiple data sources in	TRACE platform displays a unified view incorporating all integrated data	1	Verify that the GUI displays a unified, cohesive view with data from multiple sources, all

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						view of the transportation system	transportation system	various formats			accessible through the TRACE platform
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_002	Graphical User Interface (GUI) for logistics companies	All	DM-FUN-002: TRACE platform should support both structured and unstructured data	TRACE platform should support both structured and unstructured data	TRACE platform is connected to sources providing both structured and unstructured data formats	TRACE platform's GUI displays both types of data seamlessly, with no data loss or format issues	1	Verify that the GUI correctly displays both structured and unstructured data from connected sources, ensuring data consistency and completeness
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_003	Graphical User Interface (GUI) for logistics companies	All	INT-FUN-001: GUI should incorporate predictive analytics capabilities	GUI should incorporate predictive analytics to forecast trends, demand, and performance based on historical data.	TRACE platform is connected to data sources and has historical data available	Predictive analytics capabilities are accessible in the GUI for proactive decision-making	1	Verify that the GUI displays predictive analytics forecasts based on historical data, aiding decision-making
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_004	Graphical User Interface (GUI) for logistics companies	All	INT-FUN-002: GUI may incorporate collaborative features	GUI should offer shared workspaces, comments, and notifications for teamwork and communication.	GUI is operational, and multiple user accounts are available for testing	Collaborative tools (workspaces, comments, notifications)	1	Verify that users can access and utilize shared workspaces, comment features, and

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
								collaboration features	are accessible in the GUI		notifications within the GUI
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_005	Graphical User Interface (GUI) for logistics companies	All	INT-FUN-006: GUI should support different levels of users	GUI should offer at least 3 user roles: Requestor, Executor, and Admin.	TRACE platform is operational with role-based access control enabled	GUI allows users with different roles to access appropriate data and functions	1	Verify that each user role (Requestor, Executor, Admin) has correct access to GUI features and data
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_006	Graphical User Interface (GUI) for logistics companies	All	INT-FUN-010: TRACE platform shall be scalable for growing data volumes and user loads	TRACE platform should scale as data volume and user numbers increase.	TRACE platform is set to monitor performance with increasing data loads	GUI performance remains stable and responsive under increased data volumes and user loads	1	Verify that the TRACE platform GUI maintains performance and responsiveness as data volumes and user loads grow
UTH	Demonstration	Manual	DEM_MAN_F_INTEROPERABILITY_LAYER_GUI_007	Graphical User Interface (GUI) for logistics companies	All	INT-FUN-011: TRACE shall allow recipients to book real-time/same-day deliveries	GUI should offer a booking feature for recipients to schedule real-time/same-day deliveries.	TRACE platform operational with booking functionality enabled	GUI allows recipients to book real-time/same-day deliveries and manages vehicle allocation accordingly	1	Verify that the GUI enables recipients to book same-day deliveries, and that vehicle allocation updates accordingly

5.4.7.3 Monitoring and Logging Infrastructure

Table 25: Monitoring and Logging Infrastructure component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
INTRA	Test	Auto (Jenkins)	TEST_AUTO_F_MON_001	MonLog	All		This test validates that the Monitoring and Logging Stack can be deployed automatically through Jenkins.	The MonLog component is not deployed.	The MonLog component is deployed on the cloud. It collects metrics and logs from other systems.	1	Trigger the automated deployment of the MonLog component through a Jenkins pipeline.
										2	Check the status of the Dockerised modules and make sure they are healthy.
										3	Check the environmental variables that define the monitoring targets.
INTRA	Test	Manual	TEST_MAN_F_MON_002	MonLog	All		This test validates that the Monitoring and Logging Stack is collecting monitoring data and logs from the platform.	The MonLog component is deployed on the cloud.	The MonLog component collects metrics and logs.	1	Navigate through the MonLog menus and ensure that it collects metrics and

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											logs from all the backend platform components.
INTRA	Test	Manual	TEST_MAN_F_MON_003	MonLog	All	INT-PRM-008	This test validates that the Monitoring and Logging Stack will detect a critical issue in the platform.	The MonLog component is deployed on the cloud.	The MonLog component sends a notification alert about a critical system error.	1	Bring a backend component down manually.
										2	Wait for the MonLog to automatically detect the failure in a short period of time.
										3	Wait for the MonLog to send a notification alert to the platform administrator about the error.

5.4.7.4 Virtual Cockpit

Table 26: Virtual Cockpit component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
CD W	Test	Manual	DEM_MAN_I_USERINTERFACES_VC_001	Virtual Cockpit	Greek, Italian, Slovenian	VR-PRM-001, VR-PRM-002, VR-PRM-003	This test will validate that the Virtual Cockpit can monitor the UAV remotely in real-time	The Virtual Cockpit is not connected to the TRACE platform	Monitor and control (emergency) a UV	1	Create a consumer, connect to the Kafka cluster, and read Cloud the messages on the appropriate topic
										2	Start consuming information and render it to the corresponding application components
										3	Visualise UAV streaming information
										4	Send an emergency action to the UAV and control it
CD W	Test	Manual	TEST_MAN_I_USERINTERFACES_VC_001	Virtual Cockpit	Cloud	VR-FUN-001, VR-FUN-002, VR-FUN-003, VR-FUN-004,	This test will validate the successful communication of	The Virtual Cockpit is not connected to	The Virtual Cockpit connected to	1	Create a consumer, connect to the Kafka cluster,

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						VR-PRM-004, VR-PRM-005	the Virtual Cockpit with the Stream Handler	the TRACE platform	the TRACE platform		and read Cloud the messages on the appropriate topic
										2	Start consuming information and render it to the corresponding application components
										3	Visualize a test stream
CD W	Test	Manual	DEM_MAN_I_USERINTERFACES_VC_002	Virtual Cockpit	Greek, Italian, Slovenian	VR-PRM-006	Same description with DEM_MAN_I_USERINTERFACES_VC_001 but without the Head Mounted Display	The Virtual Cockpit is not connected to the TRACE platform	Monitor and control (emergency) a UAV	1	Create a consumer, connect to the Kafka cluster, and read Cloud the messages on the appropriate topic
										2	Start consuming information and render it to the corresponding application components

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										3	Visualise UAV streaming information
										4	Send an emergency action to the UAV and control it

5.4.8 Vehicle Support Services

5.4.8.1 VisionAI 3D- Low-power vision system for 3D scene reconstruction & obstacle detection

Table 27: VisionAI 3D - Low-power vision system for 3D scene reconstruction & obstacle detection component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
CSEM	Test	Manual	TEST_MAN_F_VEHICLES_VAI3D_001	VisionAI 3D	Italian	VS-FUN-004	Ensure that the integrated vision system reliably detects obstacles for cargo bikes using a coarse 3D scene reconstruction.	Evaluating multiple optical layouts for a low-power 3D scene reconstruction system.	The prototype is built and tested for real-time detection of obstacles and their position.	1	The selected optical layout is evaluated through simulation studies. Once the design is finalized, its functionality is tested in the laboratory. This is followed by the production of a prototype,

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											which is subsequently tested to ensure reliability and efficiency in obstacle detection for the required distance ranges.
CSEM	Test	Manual	TEST_MAN_F_VEHICLES_VAI3D_002	VisionAI 3D	Italian	VS-FUN-004	Ensure that information regarding the obstacle's distance and size is communicated in real time to the bike communication system for further actions.	No scene reconstruction system is currently integrated for the cargo bikes.	The vision system is integrated into the cargo bikes within a platooning framework (and/or on the driver’s helmet) to enhance obstacle avoidance and situational awareness for the driver.	2	The final test involves installing the system on bikes, where it verifies the communication of obstacle information to the BCB.

5.4.8.2 Smart Bike Autonomous Driving System

Table 28: Smart Bike Autonomous Driving System component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_SBADS_002	SBADS	Italian	COM-FUN-005 - Single bike drivers	To validate the requirement: Ensure that a single operator can control and manage a platoon of autonomous bikes through the TRACE platform.	Operator has access to TRACE system with control of a platoon.	Operator successfully directs multiple bikes autonomously.	1	Assign a single operator to manage a platoon of bikes through the TRACE platform. Conduct a series of deliveries, confirming that the operator can monitor and control the platoon without manual intervention.
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_BIKE-SBADS_TRACE_004	SBADS	Italian	COM-FUN-007 - Bike platoon reorganisation	To validate the requirement: Ensure that the TRACE system enables platoon reorganisation and autonomously directs empty bikes back to the hub.	platoon is engaged in deliveries, and an empty bike is identified.	The empty bike is reorganised in the platoon and returns to the hub autonomously.	1	Conduct deliveries with a platoon of cargo bikes. When a bike completes its delivery or it is outside on the road and it is empty, test the system's ability to

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											autonomously reorganise the platoon and return the empty bike to the hub.
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_SBADS_006	SBADS	Italian	COM-FUN-009 - Bike-to-drone exchange of parcels	To validate the requirement: Ensure the TRACE system supports parcel transfers between bikes and drones for last-meter delivery.	Bike is in an area without bike lanes or with difficult road conditions.	Drone successfully receives parcels and completes the final delivery leg and come back to the bike.	1	Simulate a scenario where a bike encounters a road condition that prevents further travel. Trigger the drone to autonomously pick up the parcel from the bike and complete the delivery. Verify the transfer and successful delivery.
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_SBADS_007	SBADS	Italian	COM-FUN-010 - Inter-bike movement of parcels	To validate the requirement: Ensure the TRACE system facilitates the transfer of parcels between bike platoons.	wo bike platoons are in communication at a designated transfer point.	Goods are successfully transferred between platoons.	1	Simulate a scenario where two bike platoons meet at a transfer point. Test the transfer of parcels

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											between the two platoons and confirm that the TRACE system handles the switch without manual intervention.
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_SBADS_008		Italian	COM-FUN-011 - Bikes and drones loading	To validate the requirement: Ensure that the TRACE system manages the efficient loading of parcels onto bikes and drones from the Modena hub.	Parcels are ready for delivery at the Modena hub.	Parcels are correctly loaded onto bikes and drones for dispatch.	1	At the Modena hub, prepare bikes and drones for parcel delivery. Load parcels onto both vehicles, ensuring that the system assigns the correct destinations to each vehicle and confirms that they are properly loaded.
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_SBADS_009	SBADS	Italian	COM-FUN-012 - Usage of bike platoons	To validate the requirement: Ensure multiple cargo bikes can be organised into platoons for the	Multiple bikes are ready to be organised into a platoon.	A platoon is successfully formed (loaded with parcels) and	1	Organise a group of cargo bikes into a platoon and initiate a delivery

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
							transportation of shipments.		ready for delivery.		operation. Confirm that the TRACE system can manage the platoon, ensuring the efficient movement of parcels to their respective destinations.
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_SBADS_010	SBADS	Italian	COM-FUN-013 - Bike sends live data to MASA	To validate the requirement: Ensure that live data from bikes, such as GPS and parcel delivery status, is communicated to MASA via 4G/5G antennas.	Bike is en route with active 4G/5G connection.	Data from the bike is successfully transmitted to MASA.	1	Equip the bike with GPS and delivery status tracking. While the bike is en route, verify that real-time data (GPS location, delivery progress) is being sent to MASA via 4G/5G antennas. Check Data will arrive at MASA servers
										2	Data will be redirected from MASA to

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											TRACE platform
UNIMORE	Demonstration	Manual	DEMO_MAN_F_VEHICLES_SBADS_014	SBADS	Italian	COM-PRM-001 - The routing shall consider the available charging infrastructure	To validate the requirement: Ensure that the TRACE platform takes into account vehicle charging stations, battery life, and route length when planning routes.	Vehicle is en route, and charging stations are available along the route.	Route is optimised to include available charging infrastructure as needed.	1	Simulate a delivery route using an electric vehicle, ensuring that the TRACE platform accounts for available charging stations. Monitor the system's ability to reroute the vehicle if battery levels drop below a critical threshold and verify that the route includes charging stops as necessary.
UNIMORE	Demonstration	Manual	DEMO_MAN_F_VEHICLES_SBADS_015	SBADS	Italian	COM-PRM-003 - Clear Marking of Bike Lanes Through Intersections for Cargo Bike Navigation	To validate the requirement: Ensure that bike lanes are clearly marked through intersections, allowing safe	Cargo bike approaches an intersection with bike lanes.	Bike lanes remain clearly marked through the intersection, preventing	1	Perform a field test where a cargo bike navigates through intersections. Ensure that

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
							passage for cargo bikes.		conflicts with other vehicles.		the Autonomous Vehicle system recognises the clear marking of bike lanes across the intersections, allowing safe passage without conflicts with motor vehicles.
UNIMORE	Demonstration	Manual	DEMO_MAN_F_VEHICLES_BIKE-SBADS_016	SBADS	Italian	COM-PRM-029 - Cargo Bike buffer zone (potential if realistic)	To validate the requirement: Ensure that a buffer zone of at least 1.5 to 2 meters is maintained between cargo bikes and other vehicles.	Cargo bike is sharing the road with other vehicles.	Buffer zone is maintained, enhancing safety.	1	Set up a scenario where cargo bikes share the road with other vehicles. Measure the distance between the bike and surrounding traffic to ensure that a buffer zone of 1.5 to 2 meters is consistently maintained. Verify that this buffer enhances

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											safety and does not disrupt traffic flow.
UNIMORE	Demonstration	Manual	DEMO_MAN_F_VEHICLES_BIKE-017	No component on this requirement	Italian	COM-PRM-032 - Minimum Width for Bike Lane Accommodating Cargo Bikes	To validate the requirement: Ensure that bike lanes have a minimum width of 2 meters, ideally 2.6 meters, to accommodate cargo bikes.	Cargo bike is on a designated bike lane.	Bike lane width meets or exceeds 2 meters, allowing safe passage for cargo bikes.	1	Assess, if possible, the width of bike lanes in various sections of a city route. Verify that lanes meet the minimum requirement of 2 meters, ideally 2.6 meters. Ensure that cargo bikes can safely navigate these lanes without encroaching on motor vehicle traffic or compromising safety.

5.4.8.3 Bike Connection Box

Table 29: Bike Connection Box component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_BIKE-BCB_003	BCB	Italian	COM-FUN-006 - Intra-bike communications	To validate the requirement: Ensure that platoons of cargo bikes can communicate delivery location data at specific crossroads.	Cargo bikes are approaching a designated crossroad.	Information on delivery locations is successfully exchanged.	1	Deploy two platoons of cargo bikes at a designated crossroad. Ensure that communication is established between the bikes to exchange information on delivery destinations and verify that the data is transferred without errors.
										2	Data will be redirected from MASA to TRACE platform
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_BCB_011	BCB	Italian	COM-FUN-014 - V2V communication	To validate the requirement: Ensure vehicle-to-vehicle communication is established, allowing bikes to	Multiple bikes are operating within close proximity.	V2V communication between bikes is successfully established.	1	Set up multiple bikes in proximity to each other and test the vehicle-to-vehicle (V2V)

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
							communicate with each other.				communication. Ensure that the bikes can share delivery data and status updates between each other seamlessly.
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_012	BCB	Italian	COM-FUN-015 - Vehicles should have minimum 4G network access	To validate the requirement: Ensure all vehicles have reliable 4G/5G access for real-time updates.	Vehicle is operating in an area with network coverage.	4G/5G connection is active, and real-time updates are transmitted.	1	Equip a vehicle with 4G/5G network connectivity. Perform a series of delivery operations in different network environments and confirm that the vehicle remains connected and is able to send real-time updates throughout the process.
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_BCB_013	BCB	Italian	COM-FUN-016 - Platform should	To validate the requirement: Ensure the TRACE platform	Users attempt to communicate	Communication is successful across all	1	Test the TRACE platform's communicatio

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						support variable communications	supports multiple communication channels (text, voice, video).	through different channels.	supported channels.		n functionalities by initiating text-based, voice-based, and video-based interactions between users. Verify that all forms of communication are supported and function smoothly.

5.4.8.4 RB-VOGUI Interface

Table 30: RB-VOGUI Interface component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
ROBOTNIK	Test	Manual	TEST_MAN_I_VEHICLES_RB-VOGUI_003	RB_VOGUI	Greek, Slovenian	VS-FUN-006 - Each unmanned vehicle shall use a dedicated GPS receiver to measure its position	To validate the requirement: The GPS value signals will be defined, created and implemented	GPS installed	Values of GPS at TRACE platform	1	The communication between the vehicle and TRACE is tested
										2	The GPS signal is received

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
ROBOTNIK	Test	Manual	TEST_MAN_I_VEHICLES_RB-VOGUI_004	RB_VOGUI	Greek, Slovenian	VS-FUN-007 - Each vehicle shall transmit to TRACE its operational status in real-time (e.g., offline, online, operative, idle, etc.)	To validate the requirement: The communication and the frequency will be checked and validated		Values of operational state	1	The communication between the vehicle and TRACE is tested
										2	The status signal is received
ROBOTNIK	Test	Manual	TEST_MAN_I_VEHICLES_RB-VOGUI_005	RB_VOGUI	Greek, Slovenian	VS-FUN-009 - Each unmanned vehicle shall use a IMU to monitor their pose and movement.	To validate the requirement: The IMU value signals will be defined, created and implemented	IMU installed	Values of IMU at TRACE platform	1	The communication between the vehicle and TRACE is tested
										2	The IMU signal is received
ROBOTNIK	Test	Manual	TEST_MAN_I_VEHICLES_RB-VOGUI_006	RB_VOGUI	Greek, Slovenian	VS-FUN-014 - The platform should be able to integrate with existing vehicle-tracking/telematics solutions	To validate the requirement: The vehicle will include a teleoperation mode and will receive commands from TRACE platform	Motors can receive velocity commands from TRACE platform	The vehicle is teleoperated by TRACE platform	1	The communication between the vehicle and TRACE is tested
										2	The vehicle moves according with the goal sent

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											from the TRACE platform
ROBOTNIK	Test	Manual	TEST_MAN_I_VEHICLES_RB-VOGUI_007	RB_VOGUI	Greek, Slovenian	VS-FUN-024 - Vehicles shall support communication hardware and software to transmit and receive data over wireless network.	To validate the requirement: The vehicle will send data over a network to TRACE platform	Router installed	TRACE receives values over wireless network	1	The wireless communication between the vehicle and TRACE is tested
										2	The data is received
ROBOTNIK	Test	Manual	TEST_MAN_I_VEHICLES_RB-VOGUI_009	RB_VOGUI	Greek, Slovenian	VS-PRM-004 - Each vehicle shall transmit to TRACE its location with time elapsing as minimum between the time to travel 100 meters and 1 minute	To validate the requirement: The location signals will be defined, created and implemented	I_TEST_VEHIRBVO_3 completed	Location received on time	1	The location signal is received every 5 seconds
ROBOTNIK	Test	Manual	TEST_MAN_I_VEHICLES_RB-VOGUI_010	RB_VOGUI	Greek, Slovenian	VS-PRM-008 - The location information may provide additional fields such as velocity	To validate the requirement: The velocity signals will be defined, created and implemented	Encoders installed	Values of Velocity at TRACE platform	1	The communication between the vehicle and TRACE is tested

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										2	The velocity signal is received

5.4.8.5 V2V communication security

Table 31: V2V communication security component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
CSEM	Test	Manual	TEST_MAN_F_V2VcomSEC_001	V2VcomSEC	Italian	VS-FUN-024	The communication is encrypted using the public key and can be decrypted with the private key	Both keys are stored on the device	The message is decrypted on the Cargo bike main controller	1	The test of this component will be conducted as part of the test of the V2V communication System.

5.4.8.6 Secure boot and firmware update for the V2V communication nodes

Table 32: Secure boot and firmware update for the V2V communication nodes component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
CSEM	Test	Manuel	TEST_MAN_F_V2VcomBOOT_001	V2VlpCOM	Italian	VS-FUN-024	A new firmware can be sent, install and run to the V2V communication device	The V2V component is connected to the V2I through the	A new firmware is running on the V2V communication device.	1	The test of this component will be conducted as part of the test of the V2V

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
								Cargo bike controller			communication System.

5.4.8.7 V2V low power communication System

Table 33: V2V low power communication System component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
CSEM	Test	Manual	TEST_MAN_F_V2VlpCOM_001	V2VlpCOM	Italian	VS-FUN-024	Communication can be established between all the V2V communication devices that will be installed on the cargo bikes.	In Lab hardware ready	Secured Communication established	1	Secured communication Established
										2	Data exchange validated
										3	Firmware update validated
CSEM	Test	Manual	TEST_MAN_F_V2VlpCOM_001	V2VlpCOM	Italian	VS-FUN-024	The devices are installed on the cargo bikes and communication can be established between them	Component embedded on. cargo bikes and connected to bike main controller	Secured Communication established	1	Secured communication Established
										2	Data exchange validated

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										3	Firmware update validated

5.4.8.8 Intrusion Detection Module and Events Management

Table 34: Intrusion Detection Module and Events Management component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UNISYSTEMS	Test	Manual	TEST_MAN_F_VEHICLE_SUPPORT_SERVICES_IDM_001	Intrusion Detection Module and Events Management	All	SRY-FUN-005: TRACE platform shall define fail-safe mechanisms to minimise the risk of system failure and enhance system redundancy	INDIRECT FULLFILLMENT	Sensors and communication infrastructure functioning normally; no active intrusion detected.	Intrusion Detection Module detects anomalies and triggers mitigation actions or redundancy management mechanisms.	1	Monitor sensor and communication data streams for anomalies.
										2	Detect potential intrusions or failures in real-time.
										3	Trigger predefined fail-safe mechanisms or redundancy

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											protocols to maintain operational continuity.
UNI SYSTEMS	Test	Manual	TEST_MAN_F_VEHICLE_SUPPORT_SERVICES_IDM_002	Intrusion Detection Module and Events Management	All	SRY-FUN-004: TRACE platform shall account for various environmental factors, such as weather and road conditions, that may affect logistics operations	INDIRECT FULLFILLMENT	Sensors and environmental data streams are continuously monitored; no abnormal conditions detected.	IDM identifies deviations due to environmental factors and flags potential risks or disruptions.	1	Continuously monitor communication between sensors and autonomous vehicles for anomalies.
										2	Detect any data irregularities indicating a potential collision risk.
UNI SYSTEMS	Test	Manual	TEST_MAN_F_VEHICLE_SUPPORT_SERVICES_IDM_003	Intrusion Detection Module and Events Management	All	SRY-FUN-006: TRACE vehicles should set driving assistance systems, such as lane-keeping assistance and automatic braking, to enhance the degree of smoothness and	INDIRECT FULLFILLMENT	Vehicle equipped with sensors and communication systems; no active driving assistance mechanisms are triggered.	Driving assistance systems, such as lane-keeping and automatic braking, are active and functioning correctly.	1	Continuously monitor data streams from vehicle sensors (e.g., lane detection, obstacle sensors).

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						steadiness of automation					
										2	Detect potential deviations, such as lane drift or obstacles, that require driving assistance intervention.

5.4.9 Vehicles

5.4.9.1 Cargo Bike

Table 35: Cargo Bike component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_BIKE-Cargo Bike_001	Cargo Bike	Italian	COM-FUN-004 - Limited capacity for delivering parcels with UGVs	To validate the requirement: Ensure the TRACE platform can correctly process the capacity of UGVs, including the number of parcels each can carry.	UGV is operational and connected to the TRACE platform.	TRACE platform correctly identifies and reports the UGV's parcel capacity.	1	Do different critical tests (near the maximum UVG' capacity or maximum parcels volumes) in order to check if it is feasible process the planning based

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											on the UVG' capacity.
										2	Data will be redirected from MASA to TRACE platform
UNIMORE	Test	Manual	DEMO_MAN_F_VEHICLES_BIKE-Cargo Bike_005	Cargo Bike	Italian	COM-FUN-008 - Bike loading	To validate the requirement: Ensure cargo bikes are correctly loaded with parcels for different city areas.	Parcels are sorted and ready for delivery at the hub (based on a routing plan)	Bikes depart with the correct parcels loaded for designated city areas.	1	Load multiple parcels onto a cargo bike, ensuring that each parcel is correctly allocated to a specific delivery area. Confirm that the system checks the parcels against the destination areas before departure.

5.4.9.2 RB_VOGUI

Table 36: RB_VOGUI component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
ROBOTNIK	Test	Manual	TEST_MAN_F_VEHICLES_RB-VOGUI_001	RB_VOGUI	Greek, Slovenian	VS-FUN-002 - A battery management system should monitor the battery of the vehicles that base their movement on electricity	To validate the requirement: The batteries mounted on the vehicles will include a BMS and monitor the provided data	Battery installed	Values of battery	1	The battery includes a BMS
										2	The battery status is monitored
ROBOTNIK	Test	Manual	TEST_MAN_F_VEHICLES_RB-VOGUI_002	RB_VOGUI	Greek, Slovenian	VS-FUN-004 - Autonomous vehicles shall have obstacle sensors.	To validate the requirement: The sensor will detect the obstacle, get the distance and avoid the obstacle	1. Distance sensor installed 2. Navigation algorithms implemented	Obstacle avoided	1	The sensor detects the obstacle and gets the distance
										2	The controller decides if the obstacle is too close
										3	The navigation system avoids the obstacle
ROBOTNIK	Test	Manual	TEST_MAN_F_VEHICLES_RB-VOGUI_003	RB_VOGUI	Greek, Slovenian	VS-FUN-006 - Each unmanned vehicle shall use a	To validate the requirement: The GPS value signals	GPS installed	Values of GPS	1	The signal and the frequency are defined

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						dedicated GPS receiver to measure its position	will be defined, created and implemented				
										2	The GPS signal is implemented
ROBOTNIK	Test	Manual	TEST_MAN_F_VEHICLES_RB-VOGUI_004	RB_VOGUI	Greek, Slovenian	VS-FUN-007 - Each vehicle shall transmit to TRACE its operational status in real-time (e.g., offline, online, operative, idle, etc.)	To validate the requirement: The status signals will be defined, created and implemented		Values of operational state	1	The signal and the frequency are defined
										2	The status signal is implemented
ROBOTNIK	Test	Manual	TEST_MAN_F_VEHICLES_RB-VOGUI_005	RB_VOGUI	Greek, Slovenian	VS-FUN-009 - Each unmanned vehicle shall use a IMU to monitor their pose and movement.	To validate the requirement: The IMU value signals will be defined, created and implemented	IMU installed	Values of IMU	1	The signal and the frequency are defined
										2	The IMU signal is implemented
ROBOTNIK	Test	Manual	TEST_MAN_F_VEHICLES_RB-VOGUI_006	RB_VOGUI	Greek, Slovenian	VS-FUN-014 - The platform should be able to integrate with	To validate the requirement: The vehicle will include a teleoperation	Motors can receive velocity commands	The vehicle is teleoperated	1	The input command is defined

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
						existing vehicle-tracking/telematics solutions	mode and will receive commands from TRACE platform				
										2	The vehicle moves according with the input commands
ROBOTNIK	Test	Manual	TEST_MAN_F_VEHICLES_RB-VOGUI_007	RB_VOGUI	Greek, Slovenian	VS-FUN-024 - Vehicles shall support communication hardware and software to transmit and receive data over wireless network.	To validate the requirement: The vehicle will send data over a network to TRACE platform	Router installed	Values sent over wireless network	1	The signal and the frequency are defined
										2	The signal is implemented
ROBOTNIK	Test	Manual	TEST_MAN_NF_VEHICLES_RB-VOGUI_008	RB_VOGUI	Greek, Slovenian	VS-PRM-001 - 2D/3D LiDAR, camera sensor or 360 cameras should perform the mapping of the environment	To validate the requirement: The vehicle will get the sensor data and will create and store a map	LIDAR installed	Map of the environment	1	The sensor gets the data
										2	The map server creates a map

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										3	The map is correctly saved
										4	The map fits the reality
ROBOTNIK	Test	Manual	TEST_MAN_NF_VEHICLES_RB-VOGUI_009	RB_VOGUI	Greek, Slovenian	VS-PRM-004 - Each vehicle shall transmit to TRACE its location with time elapsing as minimum between the time to travel 100 meters and 1 minute	To validate the requirement: The location signals will be defined, created and implemented	F_TEST_VEHIRBVO_3 completed	Location transmitted on time	1	The location signal is transmitted every 5 seconds
ROBOTNIK	Test	Manual	TEST_MAN_NF_VEHICLES_RB-VOGUI_010	RB_VOGUI	Greek, Slovenian	VS-PRM-008 - The location information may provide additional fields such as velocity	To validate the requirement: The velocity signals will be defined, created and implemented	Encoders installed	Values of Velocity	1	The signal and the frequency are defined
										2	The velocity signal is implemented
ROBOTNIK	Test	Manual	TEST_MAN_F_VEHICLES_RB-VOGUI_011	RB_VOGUI	Greek, Slovenian	COM-FUN-015 - Vehicles should have minimum 4G network access	To validate the requirement: Ensure the vehicles have reliable 4G/5G access	Vehicle is operating in an area with network coverage.	4G/5G connection is active, and real-time updates are transmitted.	1	Equip a vehicle with 4G/5G network connectivity.
										2	Confirm that the vehicle

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											remains connected and is able to send data
ROBOTNIK	Test	Manual	TEST_MAN_F_VEHICLES_RB-VOGUI_012	RB_VOGUI	Greek, Slovenian	SRY_PRM_001 - The outdoor localisation accuracy of unmanned vehicles shall be lower than 10 meters on average (or less, if possible, within 1 meter)	To validate the requirement: Ensure the localisation reaches the desired accuracy	TEST_MAN_F_VEHICLES_RB-VOGUI_003 and TEST_MAN_I_VEHICLES_RB-VOGUI_003 completed	Localisation accuracy lower than 10 meters	1	Get the localisation accuracy

5.4.10 Platform interface

5.4.10.1 Sink Manager

Table 37: Sink Manager component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
NKUA	Test	Manual	TEST_MAN_F_VEHICLES_SINK_001	Sink Manager	Greek, Slovenian	VS-FUN-007 - Each vehicle shall transmit to TRACE its operational status in real-time (e.g., offline, online, operative, idle, etc.)	To validate that the data can be consolidated and transmitted to the TRACE back-end.	Onboard sensors are able to capture measurements	Consolidated data is sent to StreamHandler	1	Data from onboard sensors is collected

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
										2	Consolidated data is sent to the StreamHandler

5.4.10.2 Optimisation module

Table 38: Optimisation module component validation

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
NKUA	Test	Manual	TEST_MAN_F_VEHICLES_OPT_001	Optimisation Module	Greek	PLT-PRM-005 TRACE platform should be designed with fault tolerance mechanisms to handle failures decently and ensure system resilience.	To validate that the platform is resilient enough to re-arrange the task workload when required	A processing task is executed onboard the vehicle.	The system re-allocates resources to ensure the proper execution of the task.	1	A vehicle with limited onboard resources (e.g., an unmanned ground vehicle) starts executing a computationally intensive task locally. The system monitors the vehicle's battery life,

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											computational load, and network conditions in real time.
										2	The system detects that the vehicle's battery lifetime is decreasing rapidly and network conditions are stable. Based on this contextual information, the framework decides to offload the task to a nearby MEC node with better computational capacity.
										3	The system initiates task migration to the MEC node.
										4	During task execution on

Lead	Val. Type	Execution Method	Test / Demo ID	Component	Scenario	Requirement Covered	Description of Fit criterion	Initial Conditions	Final Conditions	Step	Step Description
											the MEC node, the system evaluates network conditions and determines the optimal moment to transmit intermediate results back to the vehicle or other network nodes. If conditions degrade, the system applies optimal stopping theory to delay the transmission until resources are available. The task is completed efficiently, and the results are integrated seamlessly back into the vehicle's operations.

6 Conclusions

The alpha release of the TRACE platform marks a significant achievement in establishing a comprehensive infrastructure for integrated synchromodal logistics operations. Through the successful implementation of the cloud platform, CI/CD stack, and extensive testing framework, several key objectives have been accomplished.

The deployment of the platform on Hetzner Cloud has proven to be both reliable and efficient, providing the necessary scalability and security features required for a robust logistics platform. The implemented security measures, including firewalls, VPN access, and role-based access control, ensure that the platform meets stringent security requirements while maintaining accessibility for authorised users.

The CI/CD stack, comprising Jenkins, Harbor, and Portainer, has demonstrated its effectiveness in streamlining the development and deployment processes. The integration of these tools has created a seamless workflow for developers, enabling automated testing, containerisation, and deployment of platform components. The centralised user management through Keycloak has successfully simplified access control while maintaining security standards.

The comprehensive testing and validation framework, utilising requirement-based testing methodology, has ensured thorough verification of all platform components. The detailed validation matrices have provided clear evidence of component functionality and integration success, setting a strong foundation for future development phases.

The platform's modular architecture and well-documented integration methodology provide a clear pathway for future enhancements and the upcoming beta release. The established issue tracking system and development workflows will facilitate continued improvement and maintenance of the platform.

While this alpha release represents a significant milestone, it also serves as a learning opportunity for further refinements in the beta release, which will be delivered on M32. The implemented infrastructure and processes provide a solid foundation for the platform's evolution, ensuring it can meet the growing demands of modern logistics synchromodal operations.

7 References

[1] TRACE Consortium, "D3.1 Report on reference architecture (A)," 2024.

[2] TRACE Consortium, "D2.4 – Ecosystem Development, Safety and Use Cases (A)," 2024.

[3] "Keycloak," [Online]. Available: https://www.keycloak.org/.

[4] "Github," [Online]. Available: https://github.com/.

[5] "Jenkins open source automation server," [Online]. Available: https://www.jenkins.io/.

[6] "Docker: Accelerated Container Application Development," [Online]. Available: https://www.docker.com/.

[7] "Docker Compose documentation," [Online]. Available: https://docs.docker.com/compose/.

[8] "Harbor: cloud native repository for Kubernetes," [Online]. Available: https://goharbor.io/.

[9] "Portainer: Container Management for Docker and Kubernetes," [Online]. Available: https://www.portainer.io/.

Annex A: CI/CD Stack User Guide

The table below presents the configured CI/CD services:

Table 39: TRACE CI/CD Services

Service	Role	URL
Jenkins	Automation Server	https://jenkins.trace.rid-intrasoft.eu
Harbor	Container Registry	https://harbor.trace.rid-intrasoft.eu
Portainer	Container Management UI	https://portainer.trace.rid-intrasoft.eu

A centralised SSO mechanism based on Keycloak has been configured to access these services. A login window on Keycloak will pop up when you try to access these services, as depicted in ***Figure 13***, where you will use your GitHub account to log in by clicking the "sign in with GitHub button".

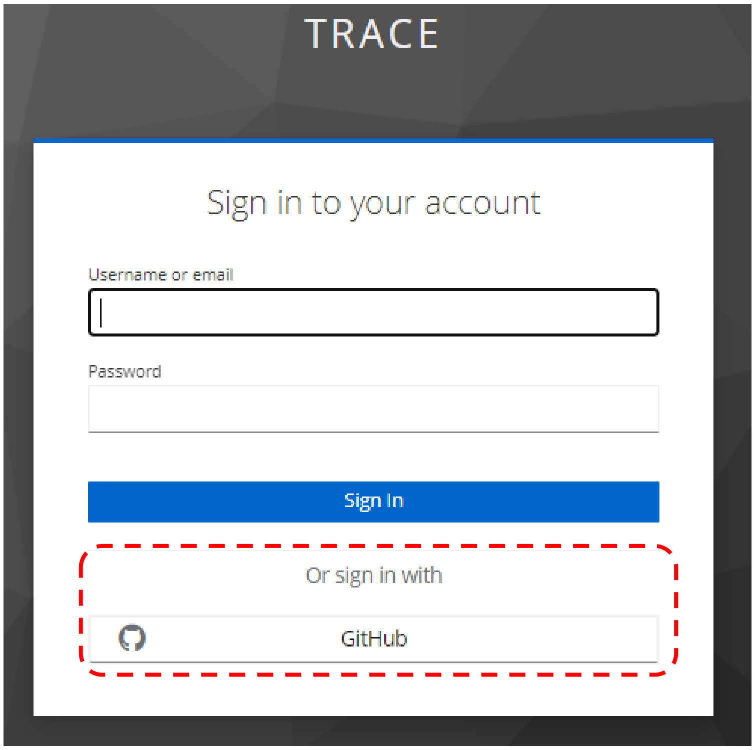


Figure 13: SSO Login Page

Note: *The first time, you will be redirected to GitHub, where you must authorise Keycloak to use your GitHub account.*

All these services are configured to support RBAC policies, ensuring that only authorised users can access specific resources based on their role within the project. Thus, when you log into the CI/CD services, you will be able to view resources that your user has permission to access. Once you have your CI/CD ready, you must request access to the respective groups of your organisation by opening a new issue on the Issue Tracking platform on GitHub.

Note: *When the platform admin adds your account to the necessary groups, you must log out and log back into the CI/CD services for the changes to apply.*

Jenkins

Jenkins is an open-source automation server widely used in the field of continuous integration and continuous delivery (CI/CD). It facilitates the automation of building, testing, and deploying software, enabling developers to integrate changes to their projects more frequently and easily. Jenkins supports various version control tools like Git and offers a vast ecosystem of plugins, allowing for the customisation and extension of its capabilities to suit diverse workflows and environments. With its user-friendly interface and extensive documentation, Jenkins provides a flexible platform for automating all phases of the software development lifecycle, from code integration to delivery.

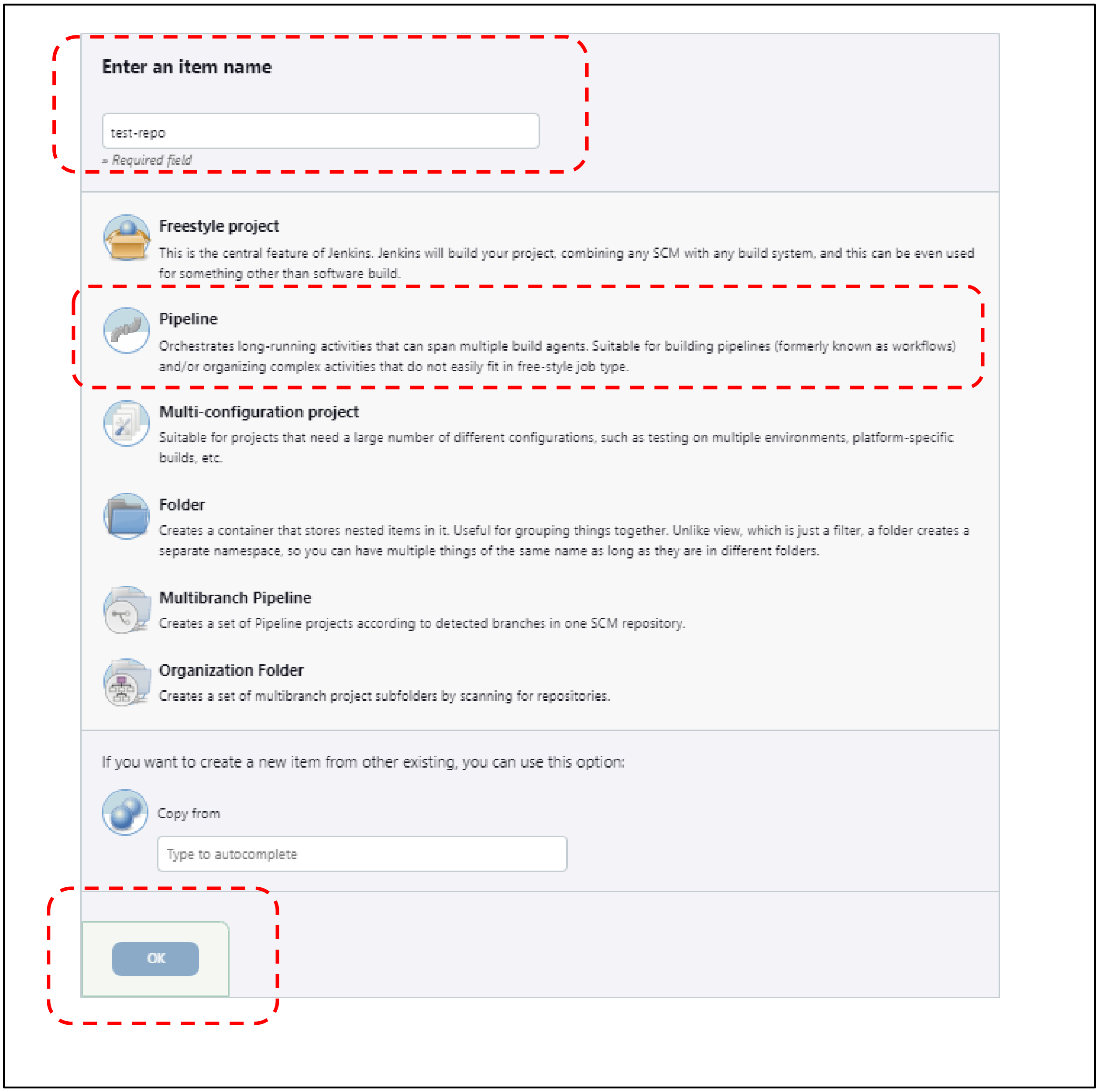
Jenkins Pipelines

Specific workspaces (folders) are created and mapped to the platform's component. Setting up a Jenkins pipeline with a trigger from a GitHub repository involves several steps:

1. Create a New Pipeline Job:

    - Go to the Jenkins dashboard, enter the appropriate folder, and click New Item from the left-side navigation pane.

    - Enter a name for your pipeline, select Pipeline, and click OK.



Enter an item name

test-repo

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

Organization Folder
Creates a set of multibranch project subfolders by scanning for repositories.

If you want to create a new item from other existing, you can use this option:

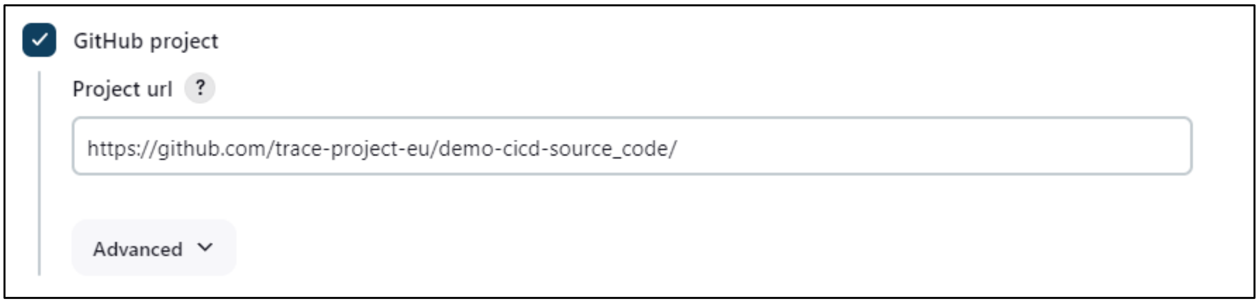
Copy from

Type to autocomplete

Figure 14: Create a Jenkins Pipeline

2. Configure the Pipeline:

- In the General section, add the URL of the GitHub project:



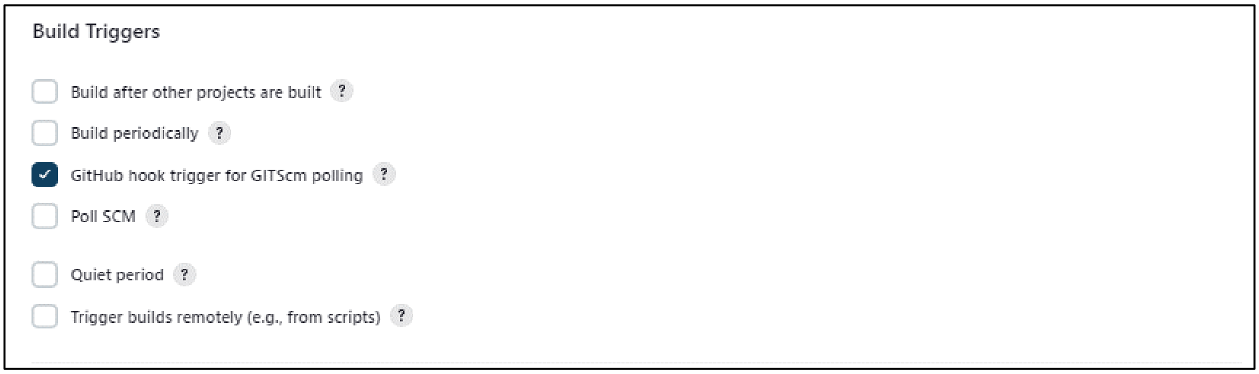
☑ GitHub project

Project url ?

https://github.com/trace-project-eu/demo-cicd-source_code/

Figure 15: Link a Pipeline with a GH Project

- In the Build Triggers section, select the GitHub hook trigger for GITScm polling option:

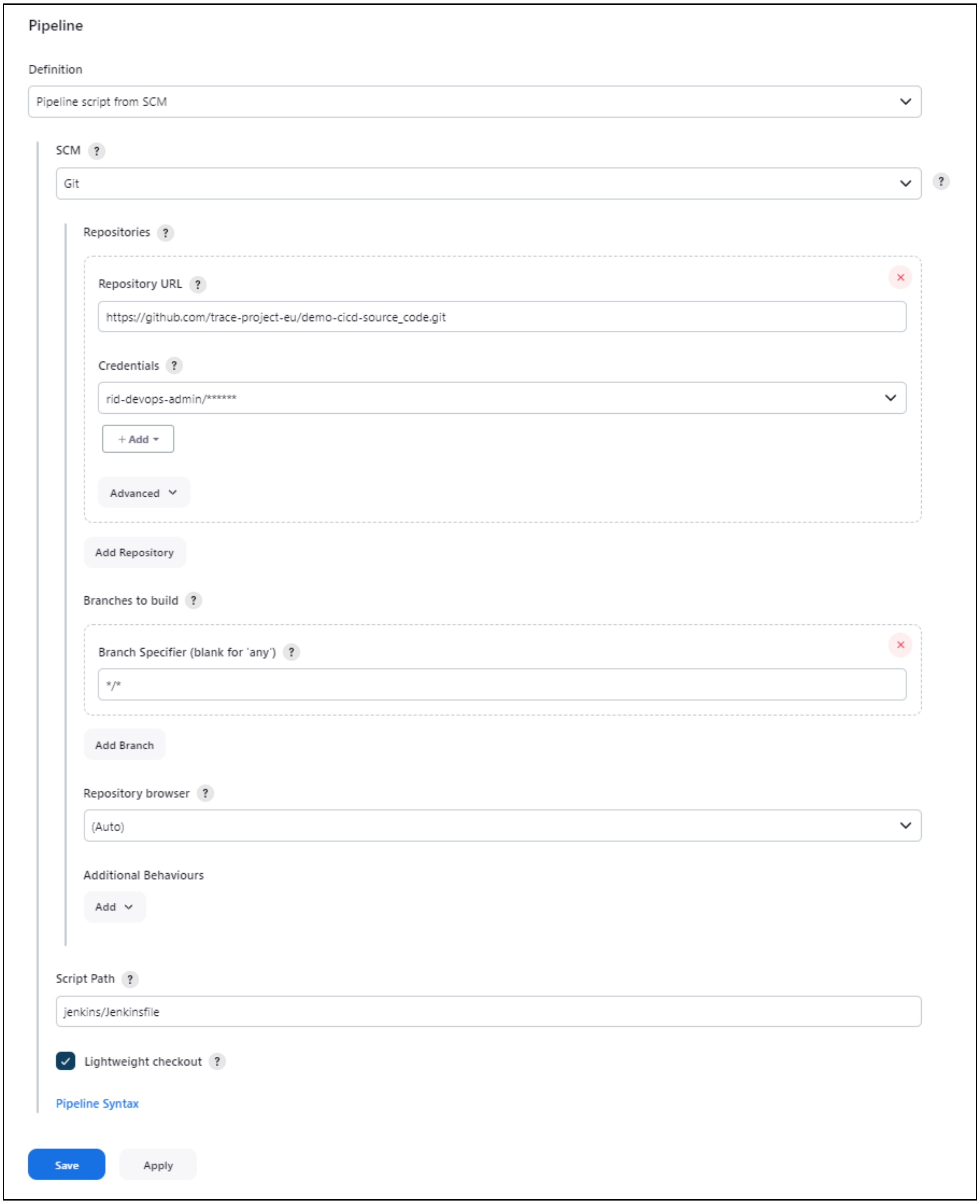


Build Triggers

□ Build after other projects are built ?
□ Build periodically ?
☑ GitHub hook trigger for GITScm polling ?
□ Poll SCM ?
□ Quiet period ?
□ Trigger builds remotely (e.g., from scripts) ?

Figure 16: Add Automated Build Triggers

- Scroll down to the Pipeline section and change the following settings:
  a. Change the Definition to Pipeline script from SCM.
  b. Set the SCM to Git.
  c. Enter your GitHub repository URL.
  d. Use the GH Access Token credentials (i.e., **gh-pull-repos**).
  e. In the Branch Specifier, enter the branch you want to build (e.g., */main).
  f. In the Script Path, enter the path to your Jenkinsfile (e.g., Jenkinsfile).



Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: https://github.com/trace-project-eu/demo-cicd-source_code.git

Credentials: rid-devops-admin/******

Branches to build

Branch Specifier (blank for 'any'): */*

Repository browser: (Auto)

Additional Behaviours

Script Path: jenkins/Jenkinsfile

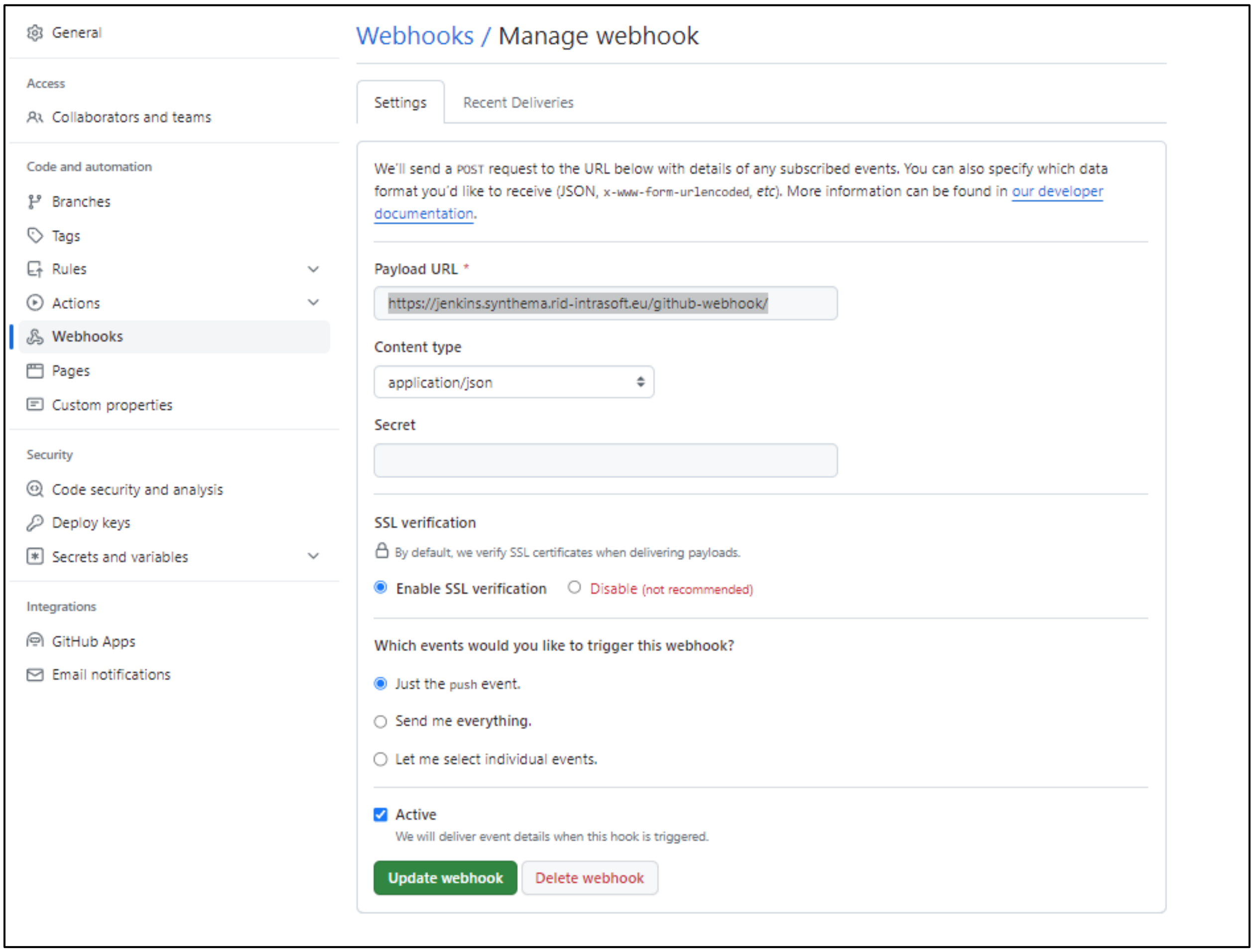
☑ Lightweight checkout

Pipeline Syntax

Figure 17: Define the Pipeline

3. Configure Webhooks for GitHub Trigger:

   - Go to your GitHub repository.
   - Navigate to Settings > Webhooks > Add webhook.
   - Set the Payload URL to your Jenkins environment followed by /github-webhook/ (i.e., https://jenkins.trace.rid-intrasoft.eu/github-webhook/).
   - Choose application/json for the content type.
   - Select Just the push event.
   - Ensure the webhook is active and click Add webhook.



Webhooks / Manage webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, *etc*). More information can be found in our developer documentation.

Payload URL *
https://jenkins.synthema.rid-intrasoft.eu/github-webhook/

Content type
application/json

Secret

SSL verification
By default, we verify SSL certificates when delivering payloads.

● Enable SSL verification ○ Disable (not recommended)

Which events would you like to trigger this webhook?

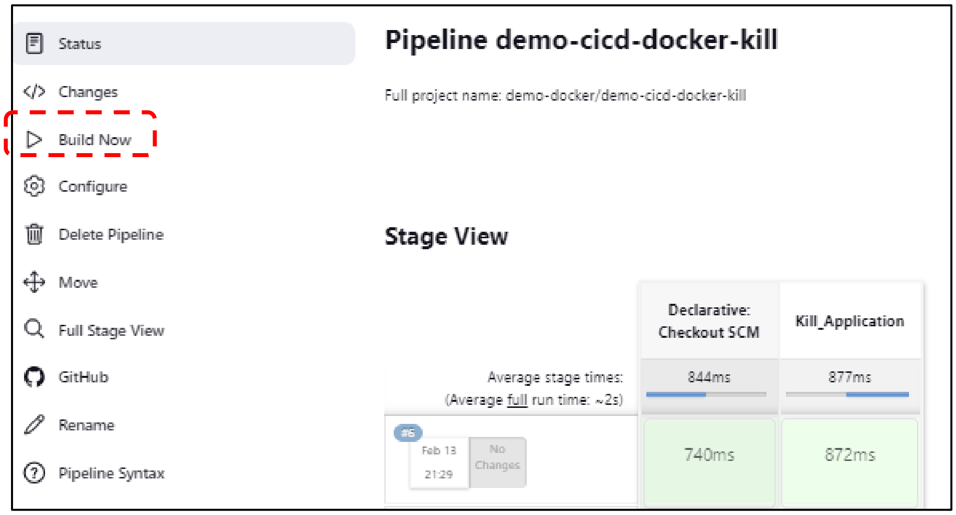
● Just the push event.
○ Send me everything.
○ Let me select individual events.

☑ Active
We will deliver event details when this hook is triggered.

Figure 18: GitHub Webhooks

Note: *In most cases, Jenkins automatically creates the webhooks shortly after creating the pipeline.*

In some cases, developers do not wish to automatically trigger the execution of pipelines, and they prefer to initiate them manually. This can be achieved through the Jenkins Dashboard, by opening the pipeline page and clicking on the "Build Now" button, as depicted in ***Figure 19***.



Pipeline demo-cicd-docker-kill

Full project name: demo-docker/demo-cicd-docker-kill

Stage View

	Declarative: Checkout SCM	Kill_Application
Average stage times: (Average full run time: ~2s)	844ms	877ms
#6 Feb 13 21:29 No Changes	740ms	872ms

Figure 19: Manual Pipeline Execution

Note: Due to a known Jenkins bug, in some cases, you must manually build the Jenkins pipeline once to allow the incoming webhooks to trigger subsequent builds automatically.

Jenkinsfile

A Jenkinsfile is a text file that contains the definition of a Jenkins Pipeline and is checked into source control. It follows the Groovy syntax and specifies the stages, steps, and environments that are used to automate the build, test, and deployment processes in a project.

The Jenkinsfile is used by Jenkins Pipeline to streamline and manage the continuous integration and delivery pipeline as code. This approach enables developers to codify their build, test, and deployment pipelines in a version-controlled document, promoting transparency, repeatability, and maintainability.

Typically, a Jenkinsfile includes:

- **Pipeline Stages:** Logical segments of the pipeline (e.g., Build, Test, Deploy) that organise the overall process.

- **Steps:** Individual tasks performed within a stage, such as executing a script, compiling code, or running tests.
- **Agents:** Instructions on where the pipeline will run, which could be on any available agent or a specific one configured for certain tasks.
- **Environment Variables:** Defines variables that can be used throughout the pipeline, such as credentials or configuration settings.
- **Post Actions:** Actions that occur after the stages are complete, such as sending notifications or cleaning up the workspace.

An example of a Jenkinsfile with multiple stages is available under the demo CI/CD repositories.

Harbor

Harbor is an open-source container image registry that secures artefacts with policies and role-based access control, ensuring images are scanned and free from vulnerabilities. It was developed as a Cloud Native Computing Foundation (CNCF) project, emphasising security, compliance, and performance. Harbor extends the capabilities of a standard image registry by providing advanced features such as image signing and scanning, user management, and replication services. It supports storing, signing, and scanning container images and Helm charts, making it a comprehensive solution for storing and managing container images securely and efficiently. Harbor is designed to be integrated into the CI/CD pipeline and is compatible with container orchestration platforms, providing a consistent application environment from development to production.

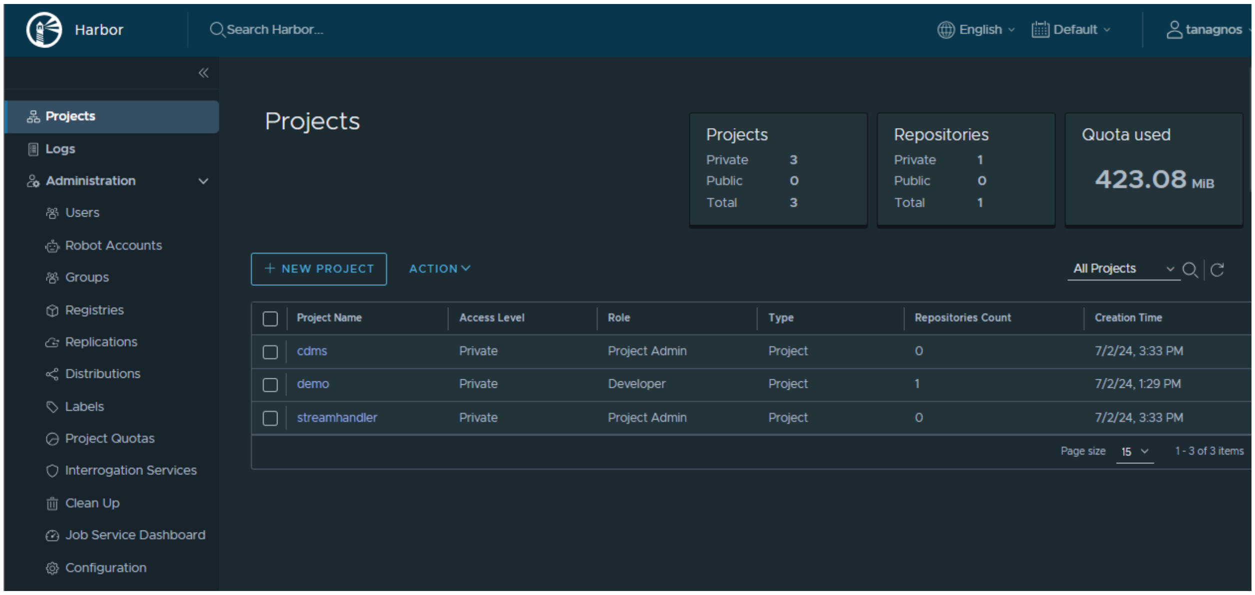
Harbor Repositories

Different Harbor projects will be created per technical component. Each project acts as a registry that can be used to store one or more container repositories. Colleagues with a Keycloak account can access the GUI while they have image push/pull permissions to specific projects (only associated with their developments). Under each project, each partner can create repositories to host their container images.

The Harbor GUI allows to:

- View the different versions of the docker images securely.
- View the history of the image.
- Delete tags that are not needed anymore.

A retention policy for keeping the latest five docker image tags per project per repository has been applied for all the projects. It is highly recommended that partners push their docker images to the private docker registry before the deployment. The stored images can be pulled and deployed in either TRACE's development or production environments. Each user will be able to push/pull images either through Jenkins (automated pipeline) or from their own remote hosts (see the following two sections).



Projects

Projects	
Private	3
Public	0
Total	3

Repositories	
Private	1
Public	0
Total	1

Quota used
423.08 MiB

☐	Project Name	Access Level	Role	Type	Repositories Count	Creation Time
☐	cdms	Private	Project Admin	Project	0	7/2/24, 3:33 PM
☐	demo	Private	Developer	Project	1	7/2/24, 1:29 PM
☐	streamhandler	Private	Project Admin	Project	0	7/2/24, 3:33 PM

Page size 15 1 - 3 of 3 items

Figure 20: Harbor Dashboard

Integration with Jenkins

You can use a specific Jenkins plugin to retrieve the Harbor credentials and pull or push container images. A snippet of the Jenkinsfile code is the following:

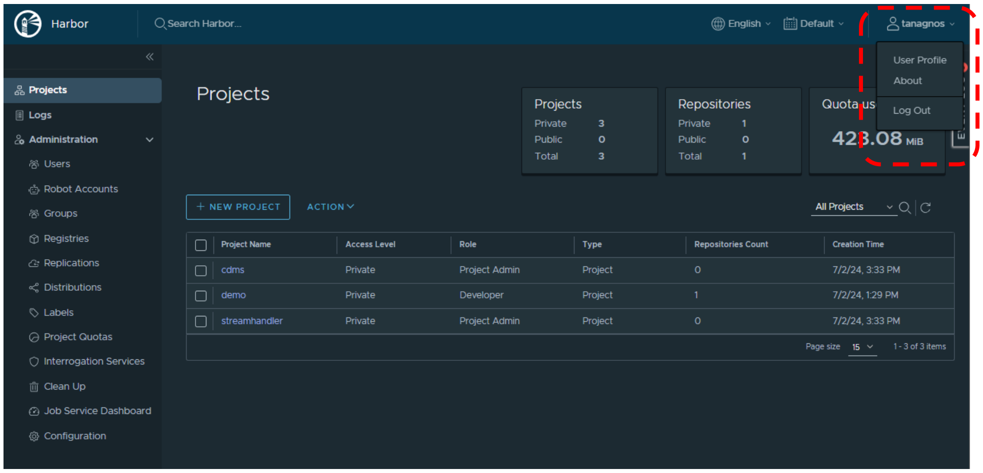
```
stage("Push_Image") {
            steps {
                withCredentials([[$class:              'UsernamePasswordMultiBinding',
credentialsId: 'harbor-jenkins-creds', usernameVariable: 'USERNAME', passwordVariable:
'PASSWORD']]) {
                    echo "***** Push Container Image *****"
                    // Login to the remote Docker Registry
                    sh 'docker login ${DOCKER_REG} -u ${USERNAME} -p ${PASSWORD}'
                    // Build the images
                    sh  'docker  image  tag  ${DOCKER_REG}${DOCKER_REPO}${APP_NAME}:test
${DOCKER_REG}${DOCKER_REPO}${APP_NAME}:latest'
                    sh               'docker                image                push
${DOCKER_REG}${DOCKER_REPO}${APP_NAME}:latest'
                }
            }
        }
```

The code uses the stored Jenkins credentials *'harbor-jenkins-creds'* to retrieve the username and password of the Harbor user and access the remote Harbor registry. You must always use these specific credentials (i.e., *'harbor-jenkins-creds'*) in your pipelines to access the Harbor registries through Jenkins.

Push from local CLI

Users can directly interact with Harbor from their local system. First, they need to log into the centralised Harbor registry. To do so, they need to retrieve their CLI Secret Key from the Harbor Dashboard by following these steps:

1. Log into the Harbor Dashboard using their credentials.
2. Access their user information by clicking their username on the top right corner and then the "User Profile" section:



Projects

Projects	
Private	3
Public	0
Total	3

Repositories	
Private	1
Public	0
Total	1

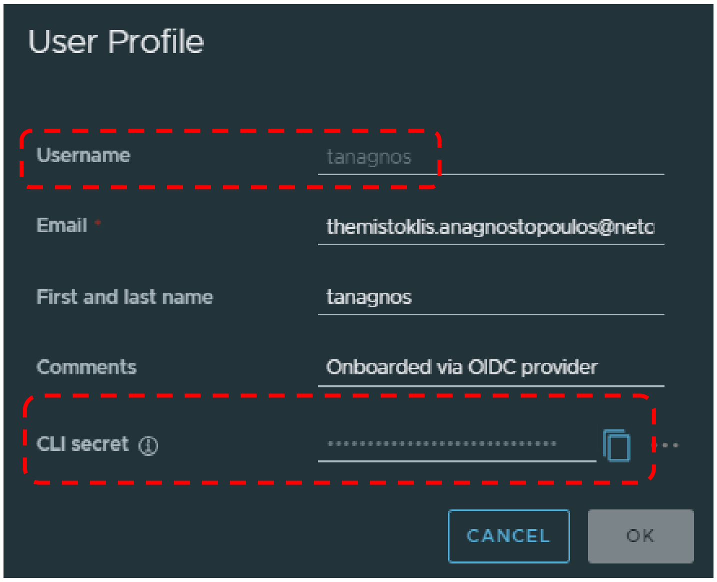
Quota us
423.08 MiB

□	Project Name	Access Level	Role	Type	Repositories Count	Creation Time
□	cdms	Private	Project Admin	Project	0	7/2/24, 3:33 PM
□	demo	Private	Developer	Project	1	7/2/24, 1:29 PM
□	streamhandler	Private	Project Admin	Project	0	7/2/24, 3:33 PM

Page size 15 1 - 3 of 3 items

Figure 21: Harbor Dashboard

3. Copy their username and CLI secret:



User Profile

Username: tanagnos
Email *: themistoklis.anagnostopoulos@netc
First and last name: tanagnos
Comments: Onboarded via OIDC provider
CLI secret: ••••••••••••••••••••••••••••••

Figure 22: Harbor User Information

They can use these credentials to log in from a local terminal:

```
themis@dpop:~/intra/projects/trace/demo_projects/docker_source$      docker      login
harbor.trace.rid-intrasoft.eu

Username: tanagnos
```

```
Password:

WARNING! Your password will be stored unencrypted in /home/themis/.docker/config.json.

Configure a credential helper to remove this warning. See

https://docs.docker.com/engine/reference/commandline/login/#credentials-store


Login Succeeded
```

Subsequently, they can build locally a new image of their component, tag it appropriately, and push it to Harbor:

```
themis@dpop:~/intra/projects/trace/demo_projects/docker_source$ docker image push
harbor.trace.rid-intrasoft.eu/demo/dummyrest2:latest

[…]

[+] Building 0.5s (11/11) FINISHED

themis@dpop:~/intra/projects/trace/demo_projects/docker_source$ docker image push
harbor.trace.rid-intrasoft.eu/demo/dummyrest2:latest

The push refers to repository [harbor.trace.rid-intrasoft.eu/demo/dummyrest2]

[…]

latest: digest: sha256:ea2795531833c6142834cf4f4d554db2d86320cd0fd7c6ec94778f87270059da
size: 3047
```

Portainer

Note: *In order for Portainer to be able to monitor Docker resources, we need to add a specific label to the Docker containers. An example is shown below:*

```
services:

    dummyrest:

        image: ${DOCKER_REG}${DOCKER_REPO}${APP_NAME}:${DOCKER_TAG}

        container_name: ${APP_NAME}

        build:

            context: ./SourceCode

        volumes:

            - "./data:/data"

        ports:

            - "8000:8000"
            - "8001:8001"
        environment:
            DB_FILE_PATH: /data/dummyrest.db
            SQLITE_DB_PATH: /data/dummyrest.db
            PYTHONPATH: /dummyrest
        command: ["gunicorn", "-w", "4", "-b", "0.0.0.0", "--access-logfile=-",
"app:create_app()"]
        labels:
            io.portainer.accesscontrol.teams: trace-all
```