European Connected Factory Platform for Agile Manufacturing

European Factory Platform

WP6: Integration and Deployment

D6.2: Integration and Deployment – Final Report

Vs: 1.0

Deliverable Lead and Editor: Mathias Axling, CNET

Date: 2022-06-30

Dissemination: Public

Status: <~~Draft~~ ¦ ~~Consortium Approved~~ ¦ EU Approved>

Short Abstract

The deliverable describes the development, integration and deployment architecture of the EFPF Ecosystem and the relevant components of the EFPF Platform. It describes methods and procedures for development, test, integration and deployment of the EFPF Ecosystem. The development, test and deployment environments used for the pilots and open calls are also described in this report.

Grant Agreement: 825075

Document Status

Deliverable Lead	Mathias Axling, CNET
Internal Reviewer 1	Usman Wajid, ICE
Internal Reviewer 2	Wernher Behrendt, SRFG
Type	Deliverable
Work Package	WP6: Integration and Deployment
ID	D6.2**:** Integration and Deployment – Final report
Due Date	2022-06-30
Delivery Date	2022-06-30
Status	<~~Draft~~ ¦ ~~Consortium Approved~~ ¦ EU Approved>

History

See Annexe A.

Status

This deliverable is subject to final acceptance by the European Commission.

Further Information

www.efpf.org

Disclaimer

Project Partners:

Executive Summary

This deliverable presents the final version of the development and deployment architecture of the EFPF ecosystem, which supports the integration of the four base platforms from the H2020 FoF-11-2016 cluster, namely NIMBLE, COMPOSITION, DIGICOR, and vf-OS. The owners of these base platforms have provided the support for the platforms' integration in EFPF. The main focus of this deliverable is on the status of integrations and deployments in EFPF during the final year of the project duration, including the integration and deployment pipeline and architecture.

The design decisions and implementation for the target development and deployment architecture are presented in their final form. The development and deployment platform has been successfully applied in pilots and open call experiments. The content in this deliverable will be used to inform future users and contributors about the relevant integration facts and methods for the EFPF ecosystem.

Table of Contents

0 Introduction

0.1 EFPF Project Overview

EFPF – European Connected Factory Platform for Agile Manufacturing – is a project funded by the H2020 Framework Programme of the European Commission under Grant Agreement 825075 and conducted from January 2019 until December 2022. It engages 30 partners (Users, Technology Providers, Consultants and Research Institutes) from 11 countries with a total budget of circa 16M€. Further information can be found at www.efpf.org.

To foster the growth of a pan-European platform ecosystem that enables the transition from "analogue-first" mass production, to "digital twins" and lot-size-one manufacturing, the EFPF project will design, build and operate a federated digital manufacturing platform. The platform will be bootstrapped by interlinking four base platforms from FoF-11-2016 cluster funded by the European Commission, early on. This will inform the design of the EFPF Data Spine and the associated toolsets to fully connect the existing user communities of the four base platforms. The federated EFPF platform will also be offered to new users through a unified Portal with value-added features such as single sign-on (SSO), user access management functionalities to hide the complexity of dealing with different platform and solution providers.

0.2 Deliverable Purpose and Scope

The purpose of this deliverable *D6.2: Integration and Deployment - Final Report* is to document the current integration and deployment views of the EFPF Ecosystem Enablers and relevant components of the EFPF Platform and base platforms participating in the EFPF Ecosystem. It describes methods and procedures for development, test, integration and deployment of the EFPF Ecosystem. The development, integration and deployment of EFPF Platform tools and services and the base platforms' services are, as specified in the Description of Action (DoA), the responsibility of the owners and managers of these components and are not detailed in the deliverable. However, guidelines for the versioning and management of exposed services are specified by EFPF. The development, test and deployment architectures used for the pilots and open calls are described in this report. Detailed installation and configuration procedures are provided in the documentation portal and administrative guides.

0.3 Target Audience

This document targets primarily project technical partners delivering tools and services for the EFPF platform: stakeholders involved in building, testing and maintaining software or performing system administration. This also includes potential future stakeholders and external organizations that will expose their platforms, tools or services to be integrated with EFPF either through pilots or open call experiments. A major audience for the document is EFF, who need to know how to plan for the operation and support of the EFPF Ecosystem in the future.

0.4 Deliverable Context

The background context of this document includes the architecture description (D3.1 “EFPF Architecture I”) outlining the targeted system design and organisation of functional architectural elements, the project plan (DoA) for the delivery of the relevant results. It is also related to deliverables on operational management and maintenance of the EFPF platform.

The document also refers to the technical infrastructure, tools and methods, used in development and deployment of the EFPF system.

Deliverable D7.2 “Operational Management and Maintenance of EFPF Platform-II”, will report the operational viewpoint e.g., monitoring, backup and support. Some design decisions reported in the deliverable are enablers for these operational features/capabilities. Security will be reported in D5.15 “EFPF Security and Governance - Final Report”.

0.5 Document Structure

This deliverable is broken down into the following sections:

- **Section 0 Introduction**: An introduction to this deliverable, including a general overview of the project, an outline of the purpose, scope, context, status, and target audience of the deliverable at hand.
- **Section 1 EFPF Integration and Deployment Update:** Provides an overview of the EFPF architecture following the process and method based on ISO/IEC/IEEE 42010:2011 “Systems and software engineering - Architecture description” [IEEE 42010, 2011]. The relevant Viewpoints and Quality Characteristics are outlined.
- **Section 2 Development Viewpoint:** Describes the Development Viewpoint including module organization and codeline organization of the EFPF Ecosystem and describes the deployment pipeline for the Data Spine and business critical Ecosystem and validation and testing procedures and tools.
- **Section 3 Deployment Viewpoint:** Documents the Deployment Viewpoint including the Runtime Platform, with the Development, Test and Production environments and dependencies.
- **Section 4 After EFPF: Recommendations for EFF and Further Work:** Summarizes Lessons Learned from the chosen development approach
- **Section 5 Conclusion and Outlook:** Discusses post project activities including the migration to the EFF.

Annexes:

- Annex A: Document History
- Annex B: References
- Annex C: Development Viewpoint Table
- Annex D: Deployment Viewpoint Table
- Annex E: dependency Table

0.6 Document Status

This document is classified as dissemination level “public”.

0.7 Document Dependencies

This document is the second part of two related deliverables, D6.1 “EFPF Integration and Deployment-I”, delivered in M18 and this report, D6.2: “EFPF Integration and Deployment-Final report”. The classification of ecosystem elements used is described in “D3.12 EFPF Data Spine Realization – Final Report” which also provides additional design descriptions of components.

0.8 Glossary and Abbreviations

A definition of standard terms related to EFPF, as well as a list of abbreviations, is available at https://www.EFPF.org/glossary

0.9 External Annexes and Supporting Documents

Annexes and Supporting Documents:

- Annex A: Document History
- Annex B: References
- Annex C: Development Viewpoint Table
- Annex D: Deployment Viewpoint Table
- Annex E: Dependency Table

0.10 Reading Notes

- The sections of the preceding D6.1 deliverable detailing component information have been moved to tables in annexes. The sections provide an overview of the development and deployment architecture.

1 EFPF Integration and Deployment Update

1.1 EFPF Architecture Context View

Figure 1 presents an overview of the high-level architecture of the EFPF ecosystem that consists of the Ecosystem Enablers and tools, services, and platforms from various providers. In contrast to the high-level architecture diagrams from the previous deliverable D3.11, the architecture in Figure 1 extends the central box of Data Spine to include more components which are collectively called as 'Ecosystem Enablers'. The Ecosystem Enablers are the core components that enable the creation and the functioning of the ecosystem. In addition, it consists of separate blocks for tools/services/data APIs indicating that the ecosystem enables the integration of individual tools and services together with full-fledged platforms.

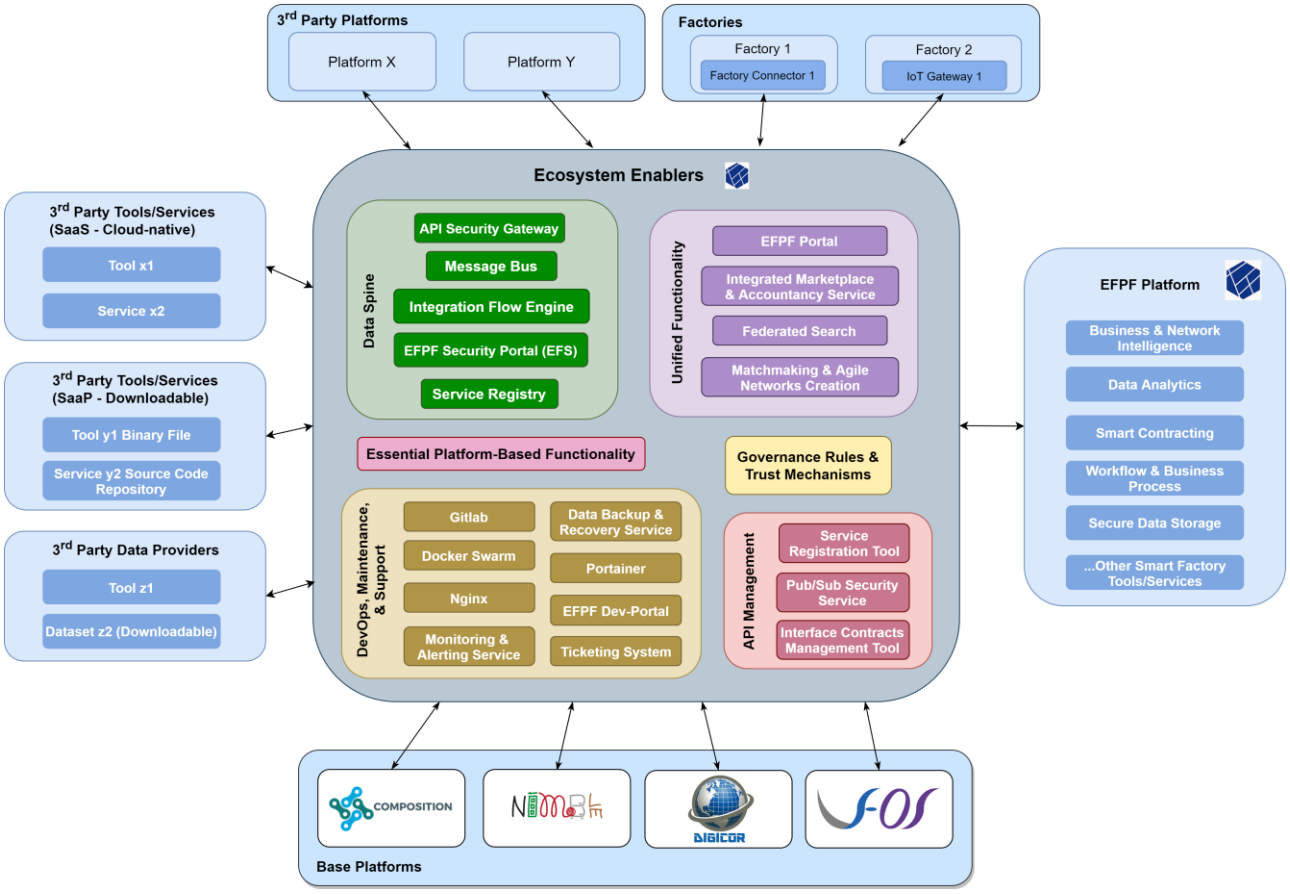

Figure 1. High-level Architecture of the EFPF Ecosystem

The EFPF platform follows the microservices architecture approach in which different functional modules implement individual functionalities that can be composed based on specific user needs. In order to implement this approach, all components in the EFPF ecosystem are prescribed to implement and publish open interfaces, preferably REST interfaces, allowing the exchange of data.

The EFPF ecosystem is designed considering the federation approach in mind where the distributed heterogeneous digital manufacturing platforms developed, provided and managed by different independent entities permit the creation of added value within the

ecosystem. To enable communication among them, an integration and communication layer, i.e., the Data Spine that acts as a translator/adapter between them is used. In addition, the rest of the Ecosystem Enablers provide the common core functionality and the digital infrastructure that is needed for the efficient operation of the ecosystem. Thus, the EFPF ecosystem as a whole follows Service-oriented architecture (SOA) style. The main elements in the EFPF federation are:

- **Ecosystem Enablers:** In the previous version of the EFPF ecosystem architecture, the Data Spine, that provides the interoperability infrastructure that interlinks and establishes interoperability between heterogeneous tools, services, and platforms and enables the creation of composite applications was illustrated as the only central core entity. In the latest version of architecture presented in this final report, the architectural vision was extended beyond interoperability and service composition to also include DevOps for easy deployment, clustering for high availability, automation for better usability, and components for an efficient infrastructure monitoring, management, operations, etc. The Ecosystem Enablers are categorized into 6 types based on the functionality they offer:
    1. Identity Federation, Cross-Platform Interoperability & Service Composition (Data Spine)
    2. DevOps, Maintenance & Support
    3. API Management
    4. Unified Functionality
    5. Essential Platform-Based Functionality
    6. Governance Rules & Trust Mechanisms
- **Data Spine:** This Ecosystem Enabler is the central entity or gluing mechanism in the EFPF federation. The Data Spine provides the interoperability infrastructure that initially interlinks and establishes interoperability between the four base platforms: COMPOSITION, DIGICOR, NIMBLE and vf-OS (see D3.1 for more details). It adheres to common industry standards and follows a modular approach to enable the creation of a modular, flexible, and extensible ecosystem. Therefore, it can be easily extended beyond interconnecting the base platforms to “plug in” new 3rd party platforms and interlink them with the already connected platforms. Figure 1 also highlights the platform agnostic nature of the Data Spine, i.e., it is evident from the high-level architecture that as far as interactions with the Data Spine are concerned, there is no distinction between the EFPF platform and the base platforms or any other 3rd party platforms. Thus, the Data Spine would be independent from the rest of the EFPF platform. This hypothetically means that even if the EFPF platform were “switched-off” in the future, the Data Spine would not be affected and therefore would continue to support an interconnected ecosystem.
- **EFPF Platform:** This is a digital platform that provides unified access to dispersed (IoT, digital manufacturing, data analytics, blockchain, distributed workflow, business intelligence, matchmaking, etc.) tools and services through the Ecosystem Enabler called ‘EFPF Portal’ that acts as the single point of entry for the ecosystem. The tools and services brought together in the EFPF platform are the market ready or reference implementations of the Smart Factory and Industry 4.0 tools from the EFPF project partners. The collection of enhanced versions of such tools and services from the base or 3rd party platforms deployed together as microservices would constitute the EFPF platform. These micro-services are made accessible through the EFPF Portal using the Single Sign-On (SSO) functionality offered by the Data Spine.

- **Base Platforms:** The EFPF ecosystem is created by initially interlinking the four digital manufacturing platforms from the European Factories-of-Future (FoF-11-2016) cluster focused on supply chains and logistics [DMC22]—namely NIMBLE [NIM22], COMPOSITION [COM22], DIGICOR [DIG22], and vf-OS [VFO22]. These are termed as the 'Base Platforms'. The base platforms provide functionality that is complementary to each other with minimum overlap and hence by interlinking them, the EFPF ecosystem is able to offer a comprehensive set of business functions.
- **3rd Party Platforms:** In addition to the four base platforms, the EFPF ecosystem enables interlinking of other 3rd party platforms that address the specific needs of connected smart factories. The examples of 3rd party platforms that joined the EFPF ecosystem include ValueChain's Network Portal platform [VLC22], Nextworks' Symphony platform [NXT22] and SMECluster's Industreweb platform [C2K22].
- **3rd Party Tools, Services, and Data:** The EFPF ecosystem can also be extended by connecting individual tools, services, and data APIs, etc., that do not belong to an existing platform.

1.2 EFPF Integration and Deployment

The objectives of the EFPF Integration and Deployment work package have been to provide a development and deployment architecture for the integrated EFPF platform (i.e., Ecosystem Enablers), so that EFPF, and later EFF, can perform continuous integration of the Ecosystem Enablers, manage its deployment on partner infrastructure and release incremental versions of the Ecosystem Enablers for functional testing. The owners of third party tools and services, base platforms and EFPF Platform tools and services will manage the respective development and deployment of their resources.

The work has included the selection and development of tools, services and platforms for integration, deployment, maintenance and quality of service, including a code repository and tools for continuous integration and deployment.

The development and deployment architecture to support the software development process for the Data Spine and Ecosystem enablers and ensure that the EFPF platform runtime environment could support the pilot activities and large-scale Open Call experimentation activities during the project.

The Ecosystem Enablers and the infrastructure for development and deployment will be transferred to the EFF after the end of the EFPF Project. Consequently, the design decisions for development and deployment also must support the organic growth of the platform ecosystem and be ready for – or easily extensible to – full scale commercial operation by the EFF.

The major architectural concerns have been to enable a modular and extensible infrastructure for the Ecosystem Enablers where new modules and components can be added, development can be distributed, deployment can be made on premise, cloud or distributed, and quality attributes like scalability, maintainability and availability are ensured. The heterogeneous nature of the development organization with many organizations in different locations, using different tools, had to be considered when designing the architecture. Constraints on common processing, software and standardization are kept to the level necessary to ensure the architectural design goals while not hindering development.

1.3 Viewpoints

This deliverable follows the same process and methodology for architectural description (AD) [Hil00] as in deliverable D6.1, which is based on ISO/IEC/IEEE 42010:2011 “Systems and software engineering - Architecture description” [IEEE 42010, 2011].

The development and delivery of the key components in the EFPF architecture fall within the Development and Deployment viewpoints of the EFPF architecture, as outlined in Rozanski and Woods [RW12].

The data comprising the development and deployment views in D6.1 was gathered by an architecture description template using a selection of views from the viewpoints described by Rozanski and Woods [RW12]. The information gathered was chosen to get a picture of the current state of the components and provide information to develop a suitable target development and deployment architecture for the pilots and open calls. The process of documenting ownership and control over source code was also a concern. Information about development, deployment, testing and release schedules for all components was of interest to design and set the scope for the deployment pipeline and runtime environments.

During the project, selected information on development and deployment for relevant components has been kept in Gitlab. For D6.2, an updated version of this template was combined with configuration information and sent out to partners to gather the most recent updates to the selection of views. With the CI/CD process and its scope defined, detailed information on the development and testing practices of components outside of Ecosystem Enablers were no longer a concern for the EFPF architecture. These are now the responsibility of the component owners, interaction with EFPF is managed by Governance Rules and up-to-date information on deployment and versions may be found in the Service Registry. The collected information is presented in overviews in the document and spreadsheets in the Annexes. The supplied information has only been minimally edited to avoid distorting the data. Contact information has been removed. Some answers are not consistent and need further updates even after several survey iterations. Enforcing the use of the suite of API management tools will likely help with keeping this data updated but gathering and continually updating the information needed for the ecosystem architects is a challenge that EFF will need to consider.

The Operational Viewpoint [RW12] documents how the system, running in its production environment, will be administered, operated, and supported. This view will be reported in “D7.2 Planning, Operational Management and Technical Support for EFPF Platform - Final Report” in M48. However, some concerns overlap with this report and will be mentioned here. Examples of concerns covered by the Operational Viewpoint are installation and upgrade, functional migration, data migration, operational monitoring and control, alerting, configuration management, performance monitoring, support, backup and restore.

1.3.1 Development View

The development viewpoint supports the software development process and describes the aspects of the architecture of interest to stakeholders involved in building, testing, and maintaining the system. Models commonly used in this view are module structure models, common design models, and codeline models. The concerns addressed are module organization, codeline organization, testing and design standards and the tools and processes used in development. The development view design decisions have not been

normative for tool and services in the EFPF Platform or for Base Platforms, only Ecosystem Enablers. Descriptive information has been collected for all components developed by the project, however, to provide an overview of compatibility and dependencies.

This deliverable documents module organization describing the logical grouping of code and the codeline organization with repository, build and test system. This includes the following issues:

- Ownership, and availability of the source code
- Instrumentation, the deployment pipeline
- Standards for design and testing, policies and guideline
- Validation and testing
- Policies and guidelines
- Delivery schedules: Continuous, fixed schedule, or when needed
- Releases: the release process for the Ecosystem Enablers, primarily ensuring that new releases of the Data Spine can be deployed without interrupting operations.
- Versioning: the externally visible items that should be subject to versioning: data schemas and formats, protocols and standards and interfaces. Dependent services can be notified of the deprecation of an interface by the Interface Contracts Management Tool.

1.3.2 Deployment View

The deployment viewpoint describes the mapping of software artefacts to the system's runtime environment and the dependencies the system has on the runtime environment. Models used are e.g. runtime platform models, network models and dependency models. These address concerns such as required hardware or cloud hosting specifications, software dependencies, runtime platforms, technology compatibility and network requirements, expressed by stakeholders e.g. system administrators, developers and testers.

- Dependencies
    - The dependencies to other components, primarily for use when designing integration testing and to provide an integrated dependency view
- Runtime Platform
    - Description of runtime platform for development, testing and production. Current status and planned runtime platform for pilots and open call (base platforms and services)
    - Container technology, orchestration and management are provided by a layer in the runtime platform.

1.4 Concerns

The development and deployment architecture was developed to fulfil the objectives stated in the Description of Action and the and quality attributes relevant for Open Call and EFF operation. The concerns and quality attributes that influenced the design was:

Performance efficiency: This quality attribute represents performance relative to the amount of resources used under a given set of conditions [IEEE 42010, 2011]. The runtime environment and container technology has been designed to support horizontal and vertical scaling. The deployment architecture must not limit the ability of the system to meet performance requirements. It has been designed to enable deployment on different runtime environments and configurations, to meet changing requirements and enable fast re-deployment on new nodes.

The EFPF ecosystem makes it easy to integrate new tools/services and promotes reusability. To ensure high performance, high throughput and high availability, the performance critical Ecosystem Enablers such as the Data Spine have CD/CD pipelines configured for automated deployment and are deployed within a cluster using the Docker Swarm container orchestration technology.

Co-existence: This quality attribute, a sub-characteristic of compatibility, represents the degree to which a component can perform its required functions while sharing is run-time environment with other components. The chosen container technology and deployment process should support that the Ecosystem Enablers and subcomponents will share runtime environment without interfering with each other, be individually upgraded and that the runtime environment may host possible multiple versions of components.

Reliability: The runtime platform must be able to perform its functions under expected conditions for a specified period of time (e.g for Open Call Experiments). The reliability quality characteristic has four sub-characteristics which must be fulfilled:

- **Maturity:** the reliability of the system under the conditions of normal operation.
- **Availability:** when required for use, the system must be operational and accessible.
- **Fault tolerance:** The degree to which the system continues to operate as specified when hardware or software malfunctions.
- **Recoverability:** The degree to which the system can restore the desired state after a failure or interruption.

The deployment architecture design must enable a stable environment where components can easily be managed in a uniform manner and restarted when malfunctioning. It must also be easy to re-deploy the system and restore the previous state as a disaster recovery measure. Operational staff must also be able to monitor the health of the runtime platform to rapidly address failures and malfunctions. The backup routines, deployment pipeline and container orchestration enable system operators to rapidly restore system operation and state.

Maintainability: how effective a system can be modified changes in requirements and its environment. Maintainability has the following sub-characteristics relevant to the development and deployment architecture:

- **Modularity:** a change to one component should have minimal impact on other components. The components should have separate units of deployment and run isolated from each other.
- **Modifiability:** the system can be modified without introducing defects or affecting quality. Flexible configuration of deployment should be possible to modify, reuse or switch out a subcomponent in the run-time environment without impact on other components.
- **Testability:** The Ecosystem Enablers must be possible to deploy specifically for test, and testing procedures and tools will be developed.

Important to the maintainability of EFPF Ecosystem Enablers is that anyone in the support team can deploy, monitor, and manage any component. To ensure this, all Ecosystem Enablers should be managed and monitored in a unified way. The loosely coupled and modular nature of the EFPF ecosystem helps significantly towards its maintainability. A high-quality user documentation of the Ecosystem Enablers and the smart factory services and tools in the EFPF ecosystem has been published on the EFPF Dev-Portal.

Portability: how effective a system or component can be transferred from one run-time environment, e.g., changing hardware or switching between on-premises or cloud deployment. This characteristic is composed of the following relevant sub-characteristics:

- **Adaptability:** the degree to which the system can effectively be adapted to different deployment and runtime environments.
- **Installability:** the degree to which the system can be successfully installed and uninstalled in the specified environment.

This quality attribute is important to the future application of EFPF in commercial operation by EFF. Installation must be configurable, automated, and adaptable to work in a commercial environment. During the EFPF project, runtime environments were not fixed from the beginning but have changed so adaptability has been a key requirement. The design allows for different or evolving hardware, software or other operational or usage environments, allowing EFF to choose between on-premises or cloud deployment.

2 Development Viewpoint

2.1 Overview

This section provides an overview of the module organization, the codeline organization of the EFPF Ecosystem and describes the deployment pipeline for the Data Spine and business critical Ecosystem and validation and testing procedures and tools. The Data Spine has been the testbed and architectural prototype when developing both the deployment pipeline and testing procedures.

2.2 Code Organization

2.2.1 Module Organization

The EFPF ecosystem is built for the creation of innovative cross-platform applications and to easily integrate existing, established solutions and cannot grow if a single set of coding tools, standards and development stacks are enforced. The module organization separates core components that comprise the integration platform and must share release cycles and infrastructure, the Ecosystem Enablers, from the tools and services of the EFPF Platform and 3rd party integrations.

The purpose of this section is to provide an overview of which components that have code managed by the EFPF and will be the responsibility of EFF to support and develop after the end of the project. The audience is ecosystem administrators, who can also use the information to find information about EFPF Platform tools and services. It is also intended to help identify any gaps or risks in the management of code critical to the operation of EFF.

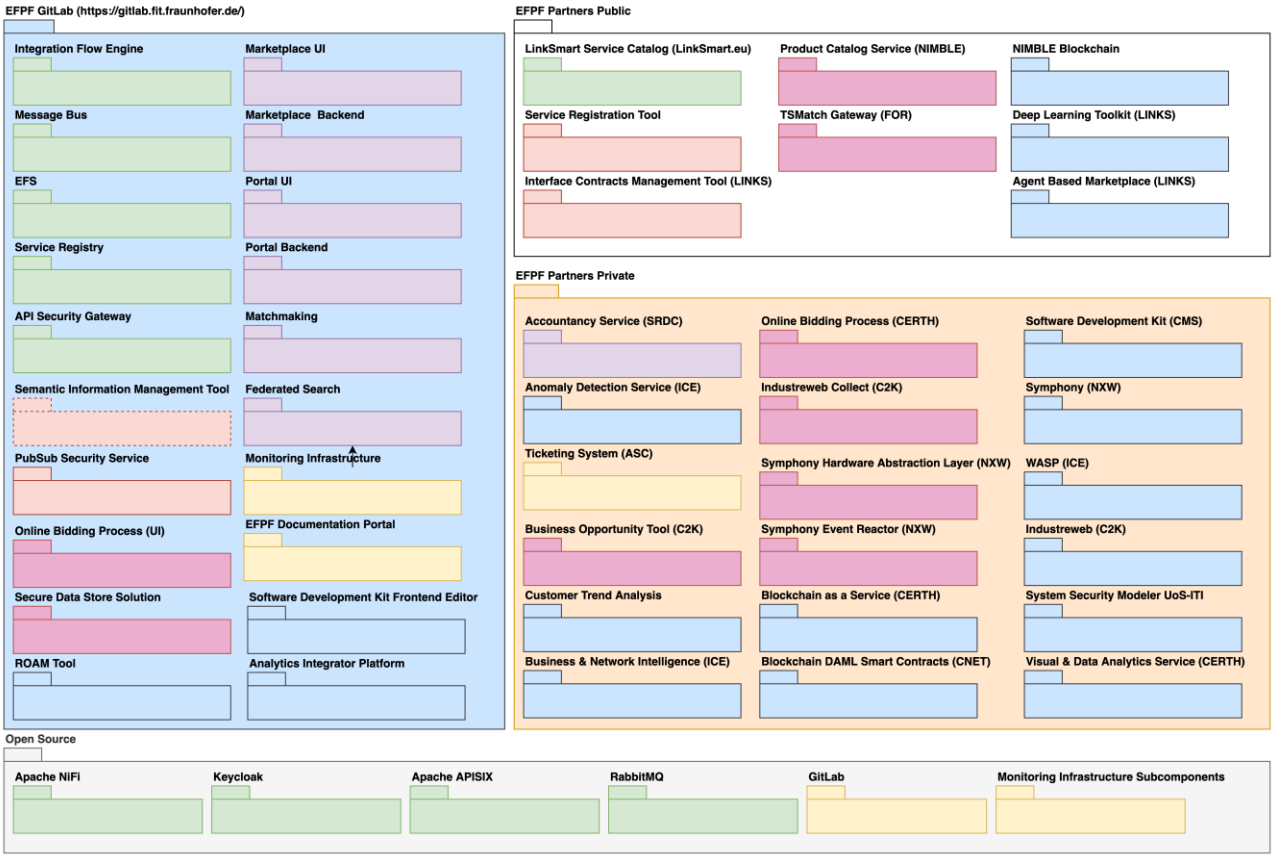

Figure 2. Code Organization

The diagrams are simplified. The contents of the EFPF GitLab package in the diagram represents modules for which code resides the GitLab instance set up for the project hosted by FIT (https.//gitlab.fit.fraunhofer.de/). The “EFPF Partners Open Source” represents modules that have the code publicly available but in a private repository. “EFPF Partners Private” contents are proprietary code modules that are not publicly available. Finally, “Open Source” represents external dependencies hosted publicly as open source on GitHub or similar repositories. External dependencies were included if it was estimated that they are business critical or changing the implementation had a significant impact on the EFPF Ecosystem Enablers. E.g., replacing NiFi or RabbitMQ would require a significant effort (weeks or months), while replacing PostgreSQL as the Keycloak database is a matter of changing the Docker configuration and the work of a few days. The Monitoring infrastructure subcomponents are included, even though they are not critical to the operation of the platform, as the complete monitoring infrastructure would take some time to replace. External dependencies of EFPF Platform components are not included, e.g., Hyperledger Sawtooth or DAML which are dependencies of Smart Contracting. The color-coding for ecosystem elements of the Context View is used.

Responsibility and ownership can mostly be inferred from the package names and is not specifically included in the diagram. The full input from the EFPF Technical Partners is available as Annex C.

2.2.2 Code Repository

A centralised version of GitLab is used for project planning, source code management as well as continuous integration (CI), continuous delivery (CD), continuous deployment (CD)

and monitoring. A dedicated group is used for the development and deployment management of tools and services in the EFPF project. New sub-groups and projects are created and assigned to the project partners as needed. A role-based access management is employed to ensure privacy and a fine-grained access control. The continuous integration and testing environment are available to all EFPF base platform and tool/service providers. It is extensively used for the core EFPF infrastructure, for example, the deployment of the Data Spine components on remote servers is completely automated using GitLab CI/CD pipelines, Ansible playbooks, and GitLab Runners. The group is populated and adapted for the integration of additional platforms and/or tools.

The GitLab infrastructure primarily consists of:

- **GitLab SCM:** The GitLab SCM comes with Git-based repositories for source code management that enables collaboration across the software development teams. It also has a provision for grouping and sub-grouping of projects, a wiki for documentation and an issue tracking functionality for management of technical issues. In addition, it provides the Milestone management functionality that can be used for project planning and release management at group-level.
    - https://gitlab.fit.fraunhofer.de/

- **GitLab CI/CD:** The GitLab CI/CD pipelines enable the development teams to build, test, deliver and monitor their code as a part of a single, integrated workflow in order to collaborate easily and efficiently. A CI/CD pipeline can be configured to be triggered on each commit or code push to a particular branch. The GitLab CI/CD infrastructure comes with a Docker registry for CD. A sample project with a simple workflow that serves as a usage example for the GitLab CI/CD process has also been provided for reference.
    - https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-integration-and-deployment/container_registry

The GitLab infrastructure is hosted on Fraunhofer FIT's servers in Sankt Augustin, Germany. The user registration data is kept and removed at user's request. The source code, wiki, issues, Docker images, build and test logs kept in GitLab are visible to the user and can be removed by the user or the repository/project owner at any time. The GitLab project resources can be configured as private (only specific users), internal (every user), or public (internet). Figure 3 illustrates a snapshot of the 'EFPF Portal' project on GitLab.

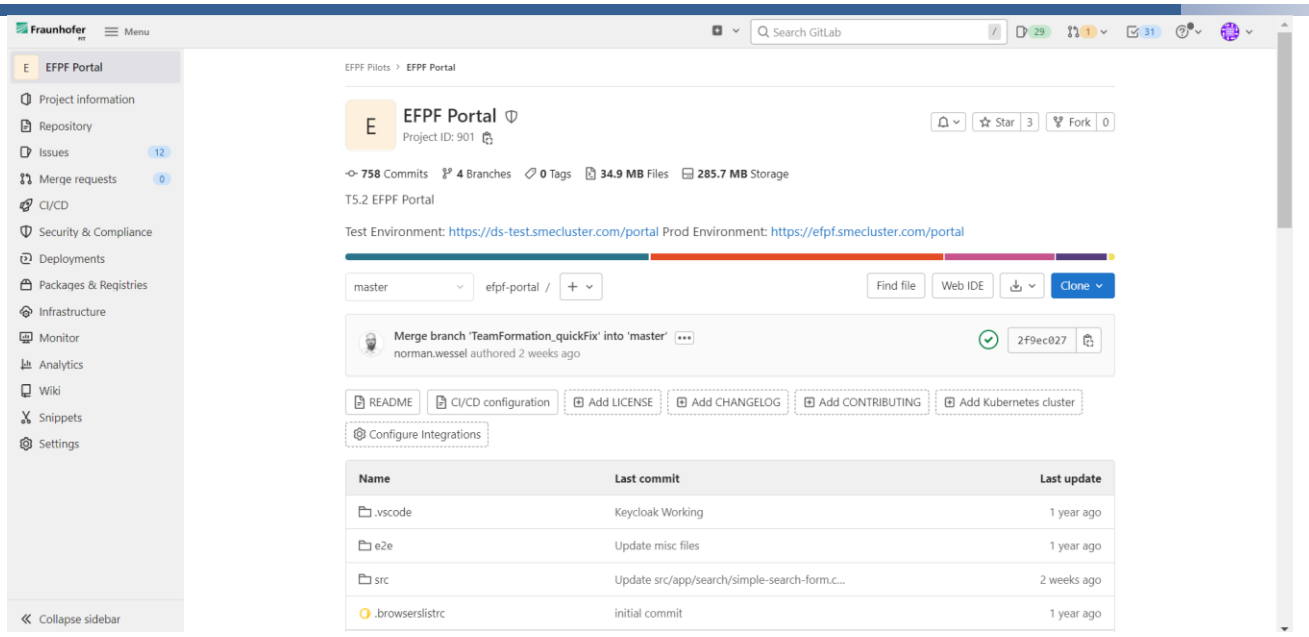

Figure 3. The EFPF Portal GitLab project

2.3 Deployment Pipeline

The deployment is the process of making an application work on a specific target machine. Nowadays, the increasing number of virtualization technologies and virtual application management system (like container orchestrator) makes the deployment operation more challenging, and it suggests to create generic patterns that can deploy application to different virtual environments.

Usually, this operation is automated using **CI/CD Pipelines,** a system to automate DevOps processes through a description of the operations that must be sequentially executed.

The pipelines can also be split in two main categories:

- **Continuous Integration (CI) Pipelines:** A set of operation in which all the developers merge code changes in a central repository. Each code change triggers an automated build and test system to provide feedbacks to the developer who made the change.
- **Continuous Delivery and Deployment (CD) Pipelines:** A set of operation to move the built code to the target environment, which can be the staging environment (to perform the final tests and make the software ready for the production environment) or directly the production environment, if the acceptance tests have already been executed.

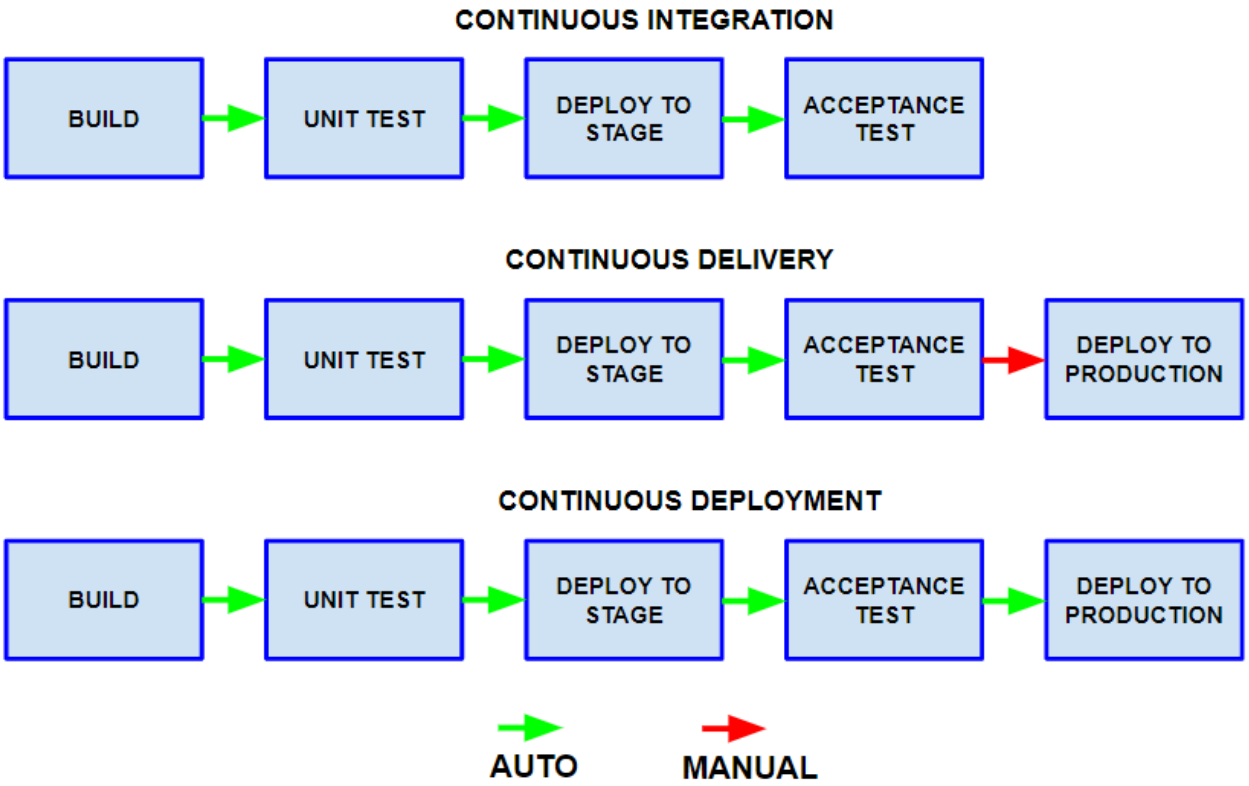

Figure 4. CI/CD Pipelines

As introduced in D3.3, the EFPF ecosystem is composed by a set of software components that need to be deployed and kept updated over the time. A centralized Gitlab repository contains all the tools and configuration necessary to deploy or update a component to the corresponding environment (test and production). A high level overview of the EFPF CI/CD Pipeline system is shown in Figure 5:

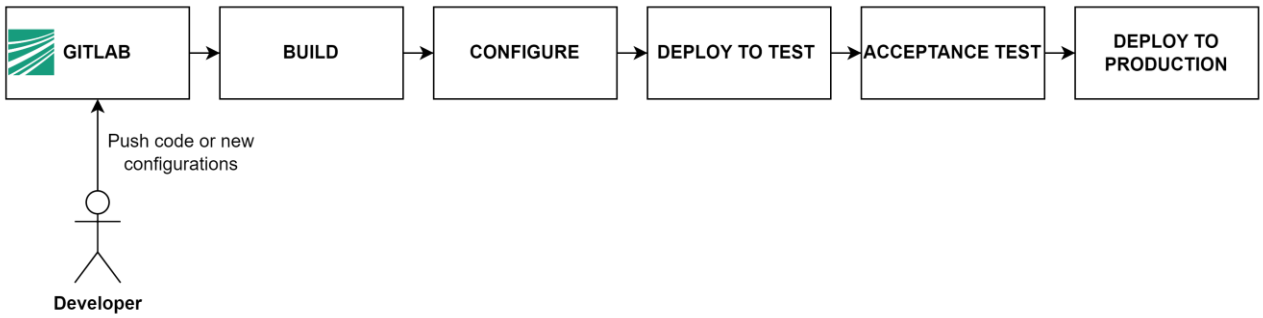

Figure 5. EFPF CI/CD Pipeline

Where all the operation are described in the following table

CI/CD Operation	Tool	Description
BUILD	Docker	The EFPF Components have been delivered as Docker Containers, therefore, a Dockerfile is responsible to produce the artifacts
CONFIGURE	Ansible + Docker Compose	The software is pre-configured using a set of ansible scripts, in combination of other specific Container's configurations described by the docker-compose file (produced by the ansible script itself)
DEPLOY TO TEST	Gitlab CI/CD Pipelines	The tools provided by GitLab (Gitlab CI/CD System) are responsible to transfer all the artifacts and their configuration files to the target machine and execute the boot up of the service
ACCEPTANCE TEST	Human validation of the system + automatic tests	Once the service has been deployed, a first human evaluation is performed to ensure the correctness of the deployment in terms of functionalities. Moreover, a periodic execution of some tests ensure that the system is constantly up and running as expected
DEPLOY TO PRODUCTION	Gitlab CI/CD Pipelines	The tools provided by GitLab (Gitlab CI/CD System) are responsible to transfer all the artifacts and their configuration files to the target machine and execute the boot up of the service

Most of the EFPF tools and services are configured and deployed through Ansible Playbooks, a tool that offers a repeatable and re-usable way to execute a set of instructions through an optimized approach (the current statement is executed only if needed). For instance: a "create directory" instruction is executed only if the directory itself has not already been created.

The Ansible Playbooks are stored in the EFPF Gitlab instance (repository *efpf-integration-and-deployment*) and two main branches has been created:

- **Master**: it contains the ansible playbooks related to the test environment
- **Production**: it contains the ansible playbooks related to the production environment

Playbooks are responsible to address all the tasks related to the pre-configurations and (some of) post-configurations of the services, including:

- Creating directories and docker volumes if needed

- Creating docker networks
- Copying all the services' configuration files from the Gitlab machine to the target machine

More in details, here is the procedure to write an EFPF CI/CD Pipeline to deploy and integrate a component within the EFPF ecosystem:

Identification of the dependencies with other components in the EFPF ecosystem

The first step is to identify whether the component to be deployed is strictly dependent to one or more EFPF component in terms of day-0 configuration variables (an example could be whether a component should have an EFPF Security Portal pre-configured user at boot time, like the EFPF Integration Flow Engine).

In this case, the EFPF CI/CD System offers an environment where a set of shared environmental variables are hosted. This environment is hosted in the *efpf-integration-and-deployment* Gitlab repository (https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-integration-and-deployment) in the *VARIABLE* file located in the root of the repository.

Identification of the service deployment requirements

The second step is focused on the definition of the requirements related to the deployment. The EFPF Deployment system has been designed to automatically deploy and manage Docker Containers. This means that, on top of this process there are a set of Docker images that need to be cached into the EFPF Gitlab Container Registry (https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-integration-and-deployment/container_registry). A multi-container docker services is described by a docker-compose, a standard YAML descriptor that is used by the docker-compose orchestrator to understand the basic information of the services to be deployed, including:

- The number of microservices that composes the overall deployment
- Which Docker image should be downloaded to create each Docker container
- How the microservices are interconnected in terms of virtual networks
- Which docker volumes should be created and, if needed, which of those are shared among different microservices
- Environmental variables, port to be exposed etc.

The output of this step consists in two information:

1. How many microservices composes the overall deployment?
2. Which docker images should be pulled and (consequently) cached in the EFPF Container registry?

Now, it is possible to fill the second critical deployment configuration file, the *Local Variable File*, placed under the directory "vars", in the root of the EFPF Integration and Deployment Gitlab Repository. Here is the template of this file

```
##########################################################################################################
##########################################################################################################
########################################     <SERVICE NAME>     ###################################
########################################   LOCAL VARIABLES   #####################################
##########################################################################################################
##########################################################################################################
#
# THIS VARIABLE IS REALLY IMPORTANT (NUMBER OF SERVICES IN THE DOCKER-COMPOSE FILE)
NUM_SERVICES=<how many microservices compose the overall service deployment>
#
################################### PLAYBOOK LOCAL VARIABLES #####################################
export <SERVICE_NAME>_PLAYBOOK_PATH=<path of the Ansible Playbook starting from the root of the repo>
export <SERVICE_NAME>_STACK_NAME=<name of the service>
export <SERVICE_i_NAME>_IMAGE=<docker image name of the microservice #i>
export <SERVICE_i_NAME>_TAG=<docker image tag of the microservice #i>
export <SERVICE_i_NAME>_NAME=<docker container name of the microservice #i>
export <SERVICE_i+1_NAME>_IMAGE=<docker image name of the microservice #i+1>
export <SERVICE_i+1_NAME>_TAG=<docker image tag of the microservice #i+1>
export <SERVICE_i+1_NAME>_NAME=<docker container name of the microservice #i+1>
#… for each microservice …
#
#
################################### CI/CD LOCAL VARIABLES ########################################
# DOCKER HUB IMGS
export DOCKER_HUB_REGISTRY#i=<url of the external registry where the docker images is stored (if unset, docker hub is taken)>
export DOCKER_HUB_USERNAME#i =<username to authenticate to the external registry (if unset, auth is not required)>
export DOCKER_HUB_PASSWORD#i =<password to authenticate to the external registry (if unset, auth is not required)>
export DOCKER_HUB_IMG#i =<full name of the docker images to be pulled from the external registry (in the form registry/img)>
export DOCKER_HUB_TAG#i =<tag of the docker images to be pulled from the external registry (assuming the form registry/img:tag)>
#... for each microservice…


#
# REGISTRY IMGS (The following variables are used from the CI/CD system to deploy the services after the pull-and-cache phase, so usually should be taken from the env variables above)
export IMG#i =$<SERVICE_i_NAME>_IMAGE
export TAG#i =$<SERVICE_i_NAME>_TAG
export IMG#i+1=$<SERVICE_i+1_NAME>_IMAGE
export TAG#i+1=$<SERVICE_i+1_NAME>_TAG
export STACK_NAME=$<SERVICE_NAME>_STACK_NAME
export PLAYBOOK_PATH=$<SERVICE_NAME>_PLAYBOOK_PATH
##########################################################################################################
```

Add a new entry to the EFPF CI/CD Pipeline

The EFPF CI/CD Pipeline system is based on the one offered by Gitlab. In the root of the *Integration and Deployment* repository, there is a file named *.gitlab-ci.yml* that describe all the operations that must be followed to automatically execute the CI/CD tasks.

This system is based on two jobs. Each of those jobs has been already defined through job templates. Here are the two jobs

- **Remove the current docker stack**

```
.remove_stack: &remove_stack
  <<: *runner_tag
  stage: remove_stack
  when: manual
  script:
    - source $LOCAL_VARIABLES
    - docker stack rm $STACK_NAME || true
```

This job is composed by two operations: first, the local variables (explained above) are loaded to basically set the STACK_NAME environmental variables, then the current docker stack is removed.

- **Deploy or update a docker stack**

```
.deploy: &deploy
  <<: *runner_tag
  stage: deploy
  when: manual
  script: |
    set -ex
    source $LOCAL_VARIABLES
    docker login -u token -p $REGISTRY_TOKEN_PROD $CI_REGISTRY
    for c in `seq 1 $NUM_SERVICES`
    do
      echo "SERVICE $c"
      IMGNAME=IMG$c
      TAGNAME=TAG$c
      DOCKER_HUB_IMG=DOCKER_HUB_IMG$c
      DOCKER_HUB_TAG=DOCKER_HUB_TAG$c
      DOCKER_HUB_REGISTRY=DOCKER_HUB_REGISTRY$c
      DOCKER_HUB_USERNAME=DOCKER_HUB_USERNAME$c
      DOCKER_HUB_PASSWORD=DOCKER_HUB_PASSWORD$c
      REG_IMG=$(eval echo "\$${IMGNAME}")
      REG_TAG=$(eval echo "\$${TAGNAME}")
      HUB_IMG=$(eval echo "\$${DOCKER_HUB_IMG}")
      HUB_TAG=$(eval echo "\$${DOCKER_HUB_TAG}")
      HUB_REGISTRY=$(eval echo "\$${DOCKER_HUB_REGISTRY}")
      HUB_USERNAME=$(eval echo "\$${DOCKER_HUB_USERNAME}")
      HUB_PASSWORD=$(eval echo "\$${DOCKER_HUB_PASSWORD}")
      RET=0
      docker manifest inspect $CI_REGISTRY_IMAGE/${REG_IMG}:${REG_TAG} > /dev/null || RET=1
      if [ $HUB_TAG == "latest" ];
      then
        RET=1
      fi
      if [ $RET -eq 1 ]
      then
        if [ -z ${HUB_PASSWORD} ];
        then
          echo "PULLING WITHOUT CREDENTIALS"
        else
          if [ -z ${HUB_USERNAME} ];
          then
```

```
            echo "PULLING IMAGE WITH TOKEN"
            HUB_USERNAME=token
        fi
        if [ -z ${HUB_REGISTRY} ];
        then
            echo "LOGIN TO DOCKER HUB"
        fi
        echo "LOGIN TO ${HUB_REGISTRY}"
        docker login -u ${HUB_USERNAME} -p ${HUB_PASSWORD} ${HUB_REGISTRY}
    fi
    echo "REMOTE PULLING IMG ${HUB_IMG}:${HUB_TAG}"
    docker pull ${HUB_IMG}:${HUB_TAG}
    docker tag ${HUB_IMG}:${HUB_TAG} $CI_REGISTRY_IMAGE/${REG_IMG}:${REG_TAG}
    docker login -u token -p $REGISTRY_TOKEN_PROD $CI_REGISTRY
    docker push $CI_REGISTRY_IMAGE/${REG_IMG}:${REG_TAG}
  fi
  docker pull $CI_REGISTRY_IMAGE/${REG_IMG}:${REG_TAG}
done
mkdir secret
echo "$ANSIBLE_SSHKEY" > secret/ansible.key
chmod 400 secret/ansible.key
ansible-playbook $PLAYBOOK_PATH --private-key=~/.ssh/id_rsa
```

This job, based on the local variables explained above, implements the following operations:

- Pull and cache all the docker images specified in the Local Variable file (it logins to the remote docker registry if needed)
- Execute the Ansible Playbook defined in the PLAYBOOK_PATH environmental variable

To define a new entry in the CI/CD pipeline system based on the templated defined above, a new section at the end of the *.gitlab-ci.yml* must be defined like the following one:

```
.local-<my_service>: &local-<my_service>
  variables:
    LOCAL_VARIABLES: vars/<my_service>.env

clean-<my_service>:
  <<: *remove_stack
  <<: *local-<my_service>

deploy-<my_service>:
  <<: *deploy
  <<: *local-<my_service>
```

This will add two jobs above the others already defined

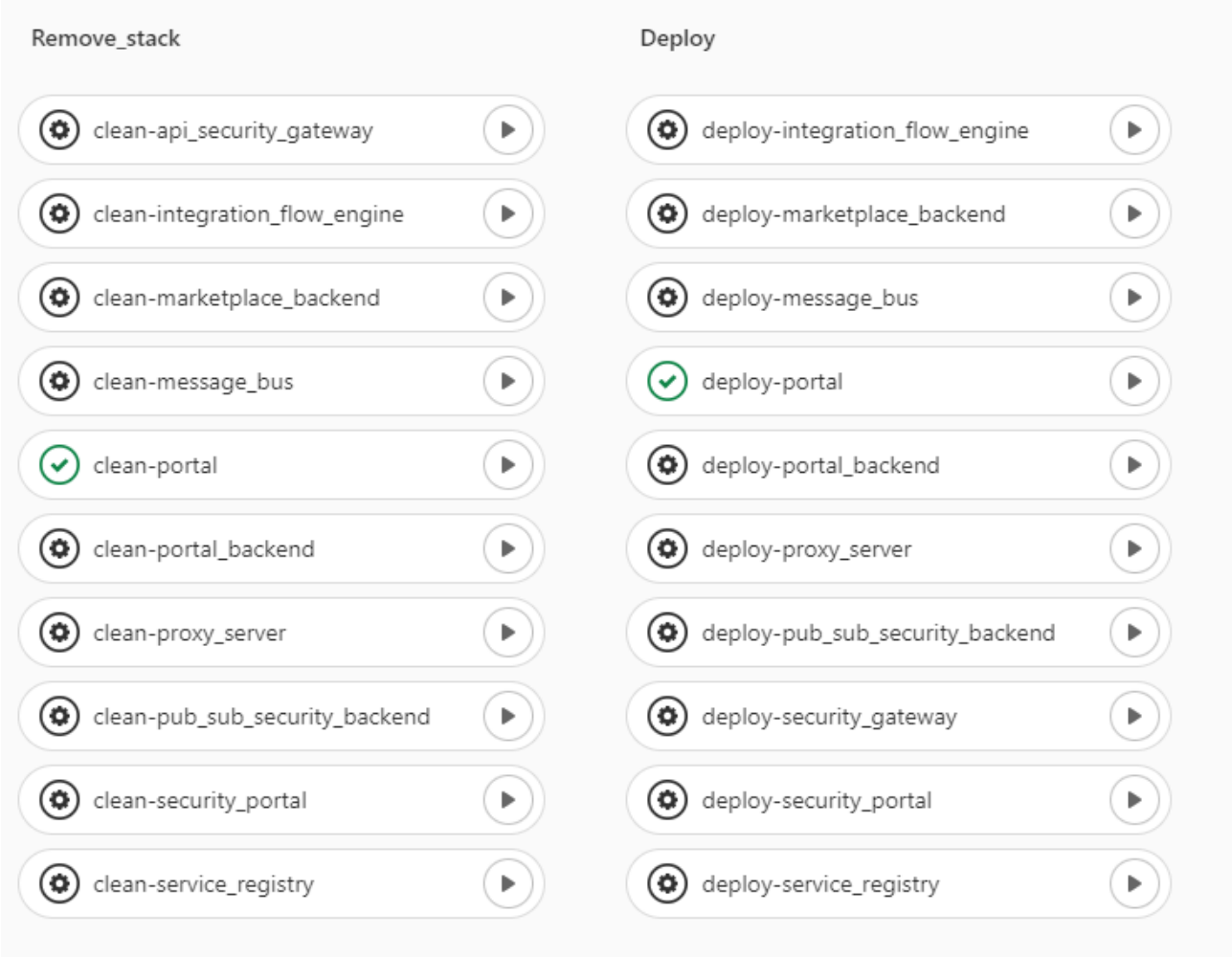

Figure 6. Pipeline Jobs

Write the Ansible Playbook

Last step is to write the core of the pre and post-configuration of the service. As introduced above, Ansible has been chosen to execute all the operation need to prepare the deployment environment of a services and, eventually, perform some of the day-0 configuration once the service is up and running.

Ansible playbook can be highly parametrized using the environmental variables defined above, so that if a value should be changed it's not needed to modify the playbook but, instead, changing the environmental variable and re-running the pipeline is sufficient to update the running services (zero-downtime is also ensured by the Docker Swarm orchestrator, as written in the next sections). Here is an example of how to use variables with Ansible:

```
vars:
  ci_registry_image: "{{ lookup('env', 'CI_REGISTRY_IMAGE') }}"
  service_registry_image: "{{ lookup('env', 'SERVICE_REGISTRY_IMAGE') }}"
  service_registry_tag: "{{ lookup('env', 'SERVICE_REGISTRY_TAG') }}"
  service_registry_name: "{{ lookup('env', 'SERVICE_REGISTRY_NAME') }}"
  rabbitmq_default_user: "{{ lookup('env', 'RABBITMQ_DEFAULT_USER') }}"
```

```
rabbitmq_default_pass: "{{ lookup('env', 'RABBITMQ_DEFAULT_PASS') }}"
registry_token: "{{ lookup('env', 'REGISTRY_TOKEN') }}"
ci_registry: "{{ lookup('env', 'CI_REGISTRY') }}"
registry_token_prod: "{{ lookup('env', 'REGISTRY_TOKEN_PROD') }}"
```

The macro *lookup* tells to Ansible that the variable should be fetched from the environmental variables.

Here are some of the most common operations that have been used from the already deployed component:

- Create a folder if it doesn't exist

```
- name: create tmp directory
  file:
    path: /tmp/serviceregistry
    state: directory
```

- Create a docker network if it doesn't exist

```
- name: Create the network if not exists
  docker_network:
    name: security-network
    driver: overlay
    scope: swarm
```

- Generate the docker-compose file from a *jinga* template

```
- name: generate serviceregistry docker-compose
  template:
    src: ./docker-compose.yml.j2
    dest: /tmp/serviceregistry/docker-compose.yml
```

- Deploy the service (the docker-compose file should be already generated

```
- name: deploy
  shell: docker login -u token -p {{ registry_token_prod }} {{ ci_registry }} && docker stack deploy -c
/tmp/serviceregistry/docker-compose.yml --with-registry-auth serviceregistry
```

2.4 Validation and Testing

2.4.1 Integration Testing

Integration testing is the phase in testing for the EFPF platform in which individual components of the Data Spine are combined and tested as a group. Integration testing is conducted every time a new deployment is made to evaluate the compliance of the system or component with the specified functional requirements collected during the development of the platform.

2.4.1.1 Integration Testing Scenarios

The integration testing focus on the core of the EFPF infrastructure, the Data Spine, and its components. The purpose of these tests is to check that each component is up, running and communicating with the other components of the platform. Each component is tested differently, and each feature of each component is tested separately. A more detailed description of these scenarios is provided in D3.12.

2.4.1.2 Integration Testing Execution

Since the implementation of the selected Integration Testing scenarios, has been carried out using the GitLab CI/CD infrastructure, a dedicated GitLab project, Testing EFPF, has been created for this purpose.

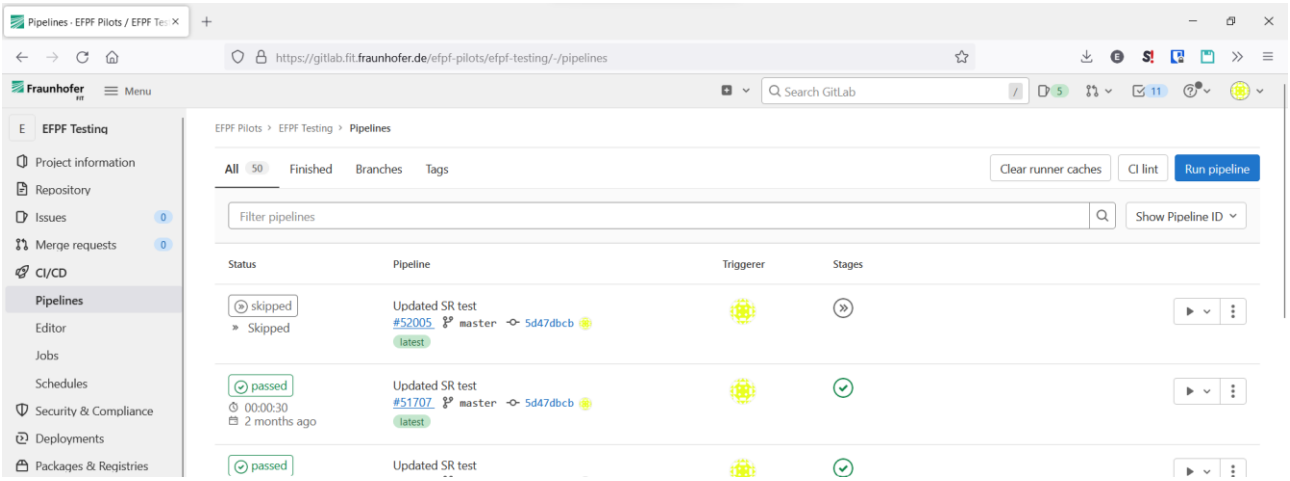

Figure 7. The testing pipeline console

The Gitlab project EFPF Testing contains the source code for all the integration test scenarios. To run the test the users have just to open the Pipelines tab in the CI/CD section of the project. In this screen it is available the information about the status of the test (Passed, Skipped or Failed), the user who has run the test and the different stages of the test.

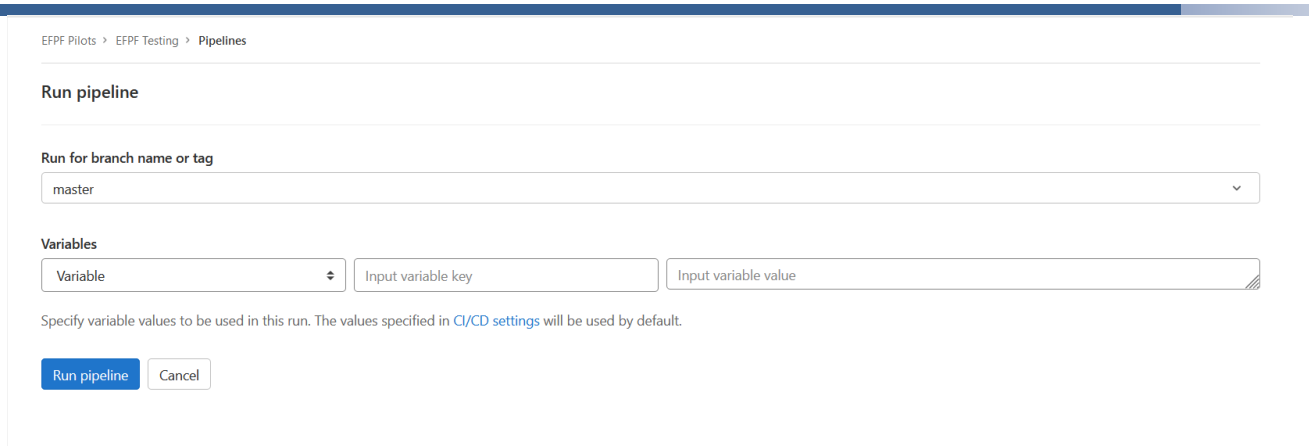

Figure 8. Environment variable definition.

When running the selected pipeline, it is possible to override the pre-defined environment variables. This is useful since with the same pipeline it is possible to test different environments, or it is possible as well to test the integration of a component with a different environment.

The full list of the environment variables is available in the settings of the CI/CD configuration for the project. From this section it is possible to change the default values for the defined variables and to see their value. This section contains sensitive information such as the credentials for the test accounts and it is hence only visible by the administrators of the project.

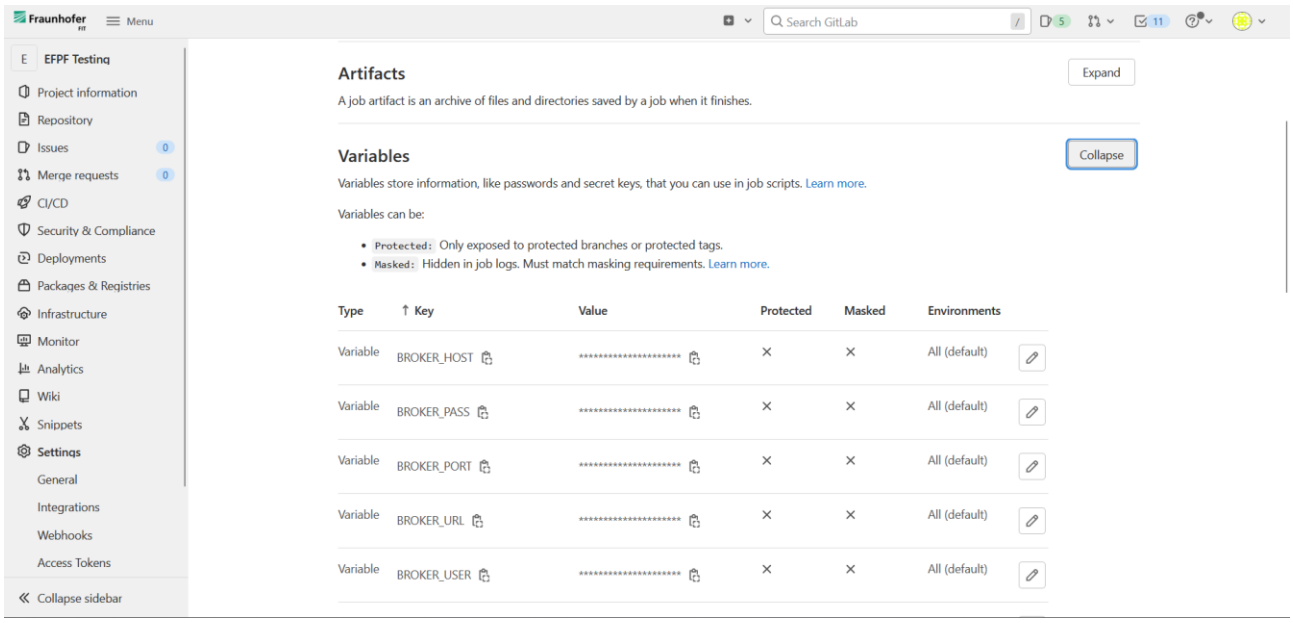

Figure 9. List of environmental variables.

Due to how the testing pipeline (shown in Table 1) has been defined, each component of the Data Spine corresponds to a different stage of the pipeline. Each stage can be run independently and for multiple times and with different environmental variables.

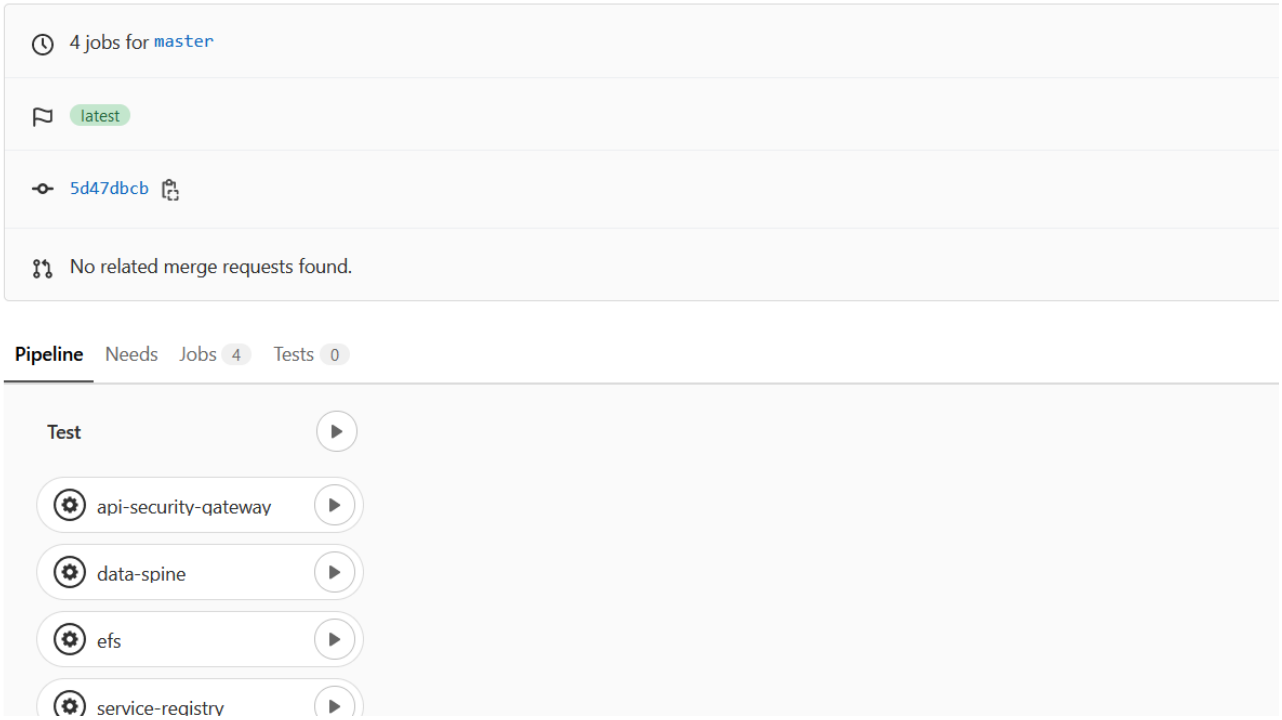

Figure 10: Pipeline composition

Since the scripts for the Integration Testing scenarios have been implemented using the Python language, the results for each of the functionalities tested for each of the stages of the pipeline (representing different Data Spine components) are displayed in the stage execution results.

The example shown in Figure 11 shows the results for the service registry stage of the pipeline where two functionalities are tested and all the operations performed to test the functionalities are printed.

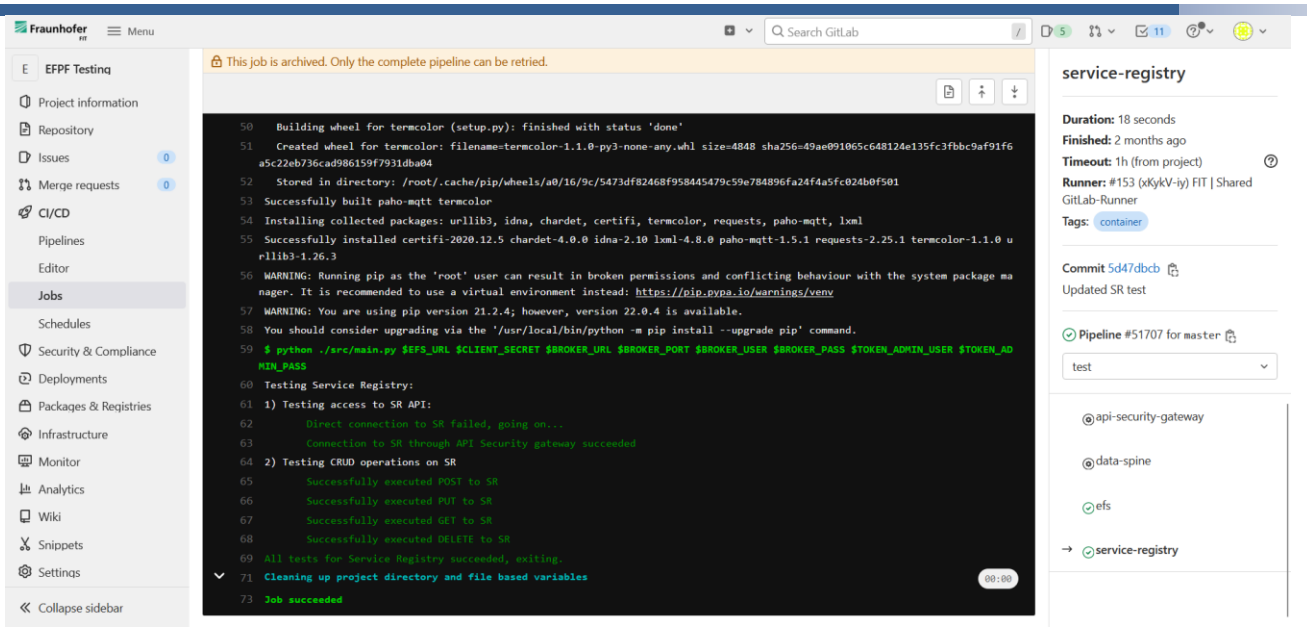

Figure 11. Detailed results of the test

Table 1: GitLab CI/CD Pipeline Config

GitLab CI/CD Pipeline config
service-registry: stage: test when: manual image: name: python:3.8 script: - cd service-registry - pip install -r requirements.txt - python ./src/main.py $EFS_URL $CLIENT_SECRET $BROKER_URL $BROKER_PORT $BROKER_USER $BROKER_PASS $TOKEN_ADMIN_USER $TOKEN_ADMIN_PASS tags: - container efs: stage: test when: manual image:

```
    name: python:3.8
  script:
    - cd efs
    - pip install -r requirements.txt
    - python ./src/main.py $EFS_URL $CLIENT_SECRET $TOKEN_ADMIN_USER $TOKEN_ADMIN_PASS
  tags:
    - container


api-security-gateway:
  stage: test
  when: manual
  image:
    name: python:3.8
  script:
    - cd api-security-gateway
    - pip install -r requirements.txt
    - python ./src/main.py $EFS_URL $CLIENT_SECRET $TOKEN_BASIC_USER $TOKEN_BASIC_PASS
  tags:
    - container


message-bus:
  stage: test
  when: manual
  image:
    name: python:3.8
  script:
    - cd message-bus
    - pip install -r requirements.txt
    -
 python ./src/main.py $EFS_URL $CLIENT_SECRET $TOKEN_ADMIN_USER $TOKEN_ADMIN_PASS $T
OKEN_BASIC_USER $TOKEN_BASIC_PASS
  tags:
    - container


data-spine:
```

```
 stage: test
 when: manual
 image:
   name: python:3.8
 script:
   - cd data-spine
   - pip install -r requirements.txt
   -
 python ./src/main.py $EFS_URL $CLIENT_SECRET $TOKEN_ADMIN_USER $TOKEN_ADMIN_PASS $T
OKEN_BASIC_USER $TOKEN_BASIC_PASS
 tags:
   - container
```

2.4.2 Performance Testing

The EFPF performance testing have been defined and written to be versatile and cover different possible load testing categories. Load tests can in fact cover different categories according to how and with which parameters they are run. At the current stage of the EFPF Project these tests have been used to check the correct functioning of the platform under minimal load, without any problems. Another use of these test has been to simulate the use of the platform by many different users before opening it up for open-call projects.

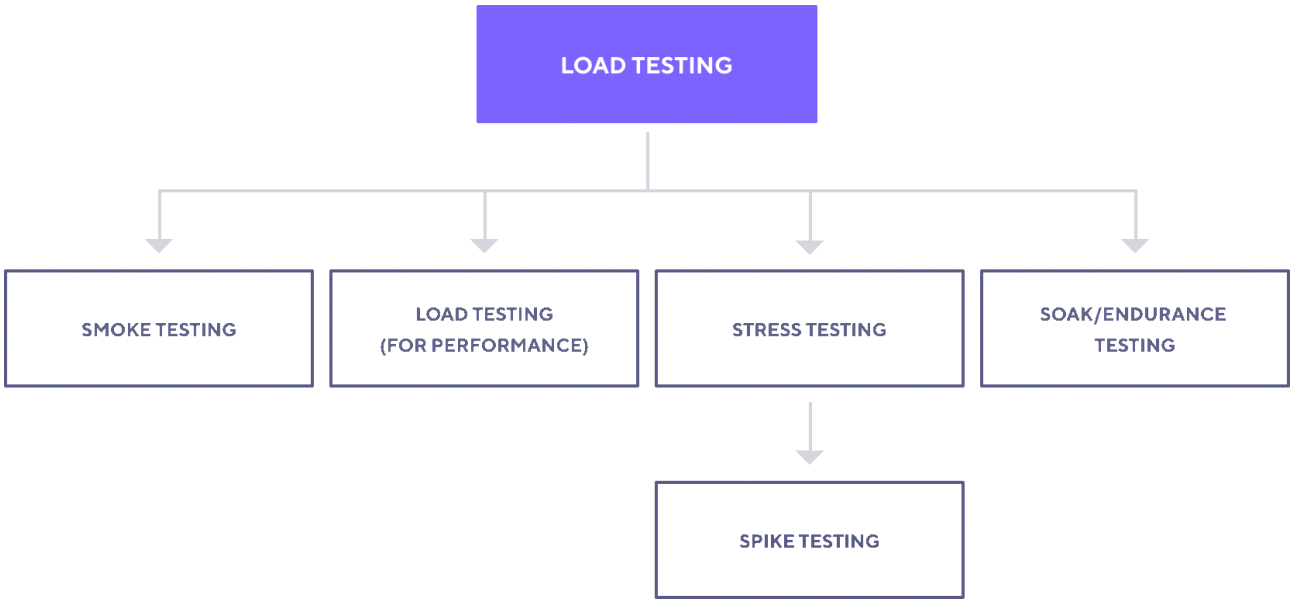

Figure 12. Subcategories of performance (load) testing

2.4.2.1 Performance Testing Scenarios

The performance testing focus on the core of the EFPF infrastructure, the Data Spine, and its components. Three scenarios have been identified. The first two scenarios cover the two communication ways in which the Data Spine can be involved. These scenarios cover the communication between different tools and services. The third scenario covers instead

the situation in which users interact directly with the platform's user interface endpoints. The scenarios and their architecture are described in more detail in D3.12

2.4.2.2 Performance Testing Execution

All the three Performance Test scenarios' scripts are distributed as Docker images. This allows for an easy distribution of the scripts. This is necessary since executing these scripts requires a powerful machine which has to be connected with a connection capable of sustaining all the load that the users want to simulate.

Each Docker container can be run independently from the other. To run the tests the user has to enter the following commands on a machine running docker.

```
docker run --rm -it –env-file ./.env efpf-performance-testing-synchronous:latest

docker run --rm -it –env-file ./.env efpf-performance-testing-asynchronous:latest

docker run –rm -it –env-file ./.env efpf-performance-testing-ui:latest
```

These commands will run the containers of the tests and will fetch the required values for the predefined environment variables from a file named .env, placed in the same folder where the test command is run, containing the values for the required environment variables for each test.

In Table 2-4 are reported the names and the descriptions for each of the environmental variables required for each performance test scenario.

Table 2. Environment variables for Synchronous Scenario

Variable Name	Description
BASE_URL	Direct URL for the Test service providing test data
EFPF_URL	URL for the Test service providing test data routed through the Data Spine
TARGET_USERS	Number of target users to generate
EFS_URL	URL for EFS
TEST_USERNAME	Username of test account
TEST_PASSWORD	Password of test account
INFLUX_USER	InfluxDB username
INFLUX_PASS	InfluxDB password

Table 3. Environment variables for Asynchronous Scenario

Variable Name	Description
MQTT_TOPIC_IN	MQTT topic on which publish the test data
MQTT_TOPIC_IN	MQTT topic on which subscribe to the test data

N_CLIENTS	Number of clients to generate which will exchange data on the Data Spine
EFS_URL	URL for EFS
TEST_USERNAME	Username of test account
TEST_PASSWORD	Password of test account
INFLUX_USER	InfluxDB username
INFLUX_PASS	InfluxDB password

Table 4. Environment variables for UI Scenario

Variable Name	Description
EFPF_URL	URL for the UI of the components to be tested
TARGET_USERS	Number of target users to generate
EFS_URL	URL for EFS
TEST_USERNAME	Username of test account
TEST_PASSWORD	Password of test account
INFLUX_USER	InfluxDB username
INFLUX_PASS	InfluxDB password

The k6 framework used as the core for these performance tests provides insights for the outcomes of the tests straight from the command line from which these tests are executed.

However it is possible to have these tests displayed in a more user friendly way, and to store the results of the tests in case different deployment configurations are being tested. In this case a custom configured InfluxDB instance has to be deployed as well. If this is provided and the correspondent environmental variables are set in the .env file, then the test scripts will upload the results of the InfluxDB instance.

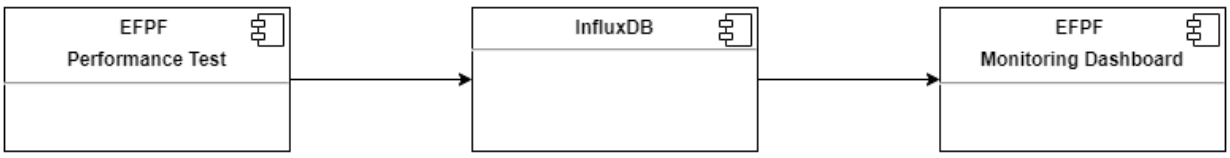

Figure 13. Distribution of test results architecture

Since the EFPF monitoring infrastructure has Grafana at its core and can be easily integrated with the InfluxDB database, three custom dashboard have been developed to visualize in real time the results of the tests. This choice provides a single entry point for visualizing the results of the tests and to see the performance of the infrastructure using the other dashboards of the EFPF monitoring infrastructure.

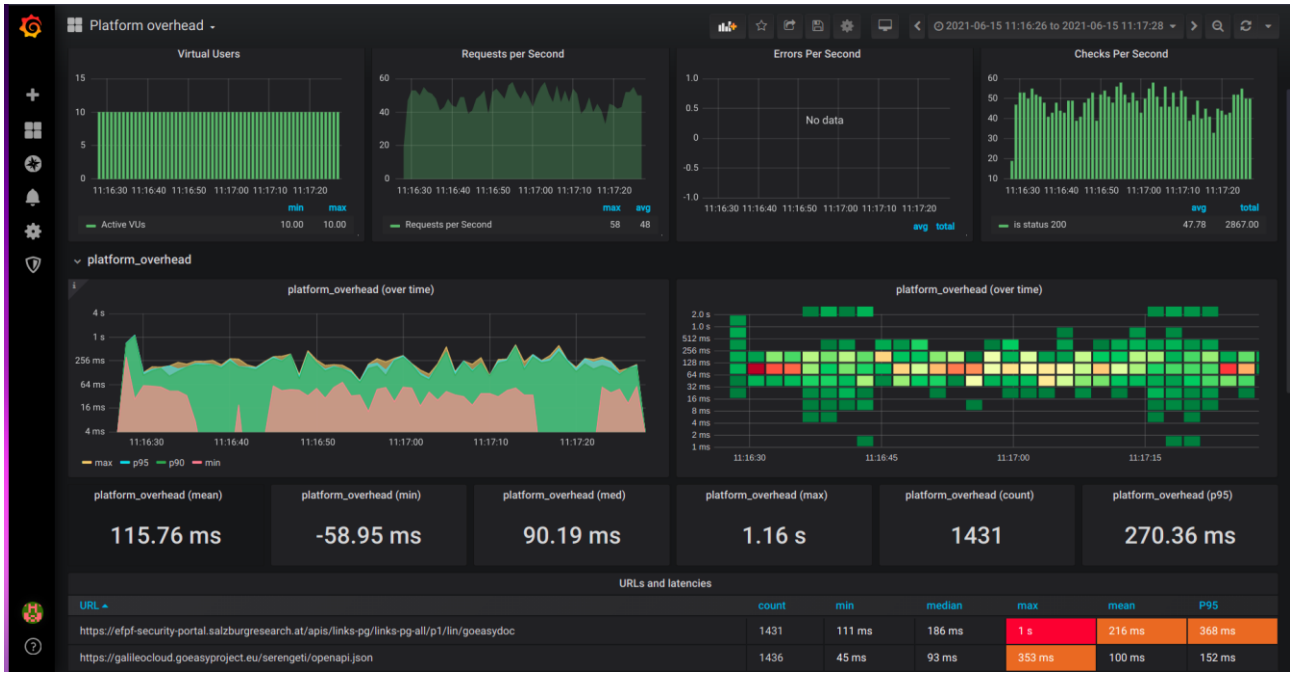

Figure 14. Grafana Dashboard containing the test results

2.5 Policies and Guidelines

During the project, the components have been released as needed. For the Open Calls, two coordinated releases of dependent services were made (“Phase 1” and “Phase 2”) to ensure a stable system and a coherent set of services. No further fixed release schedule or release train has been set, but due to the multiple teams involved and runtime dependencies in the platform, this will be a necessity for the Ecosystem Enablers when managed by the EFF.

3 Deployment Viewpoint

3.1 Overview

The production environment was deployed in two major rollouts, after which the Ecosystem Enablers were in place.

3.2 Runtime platform

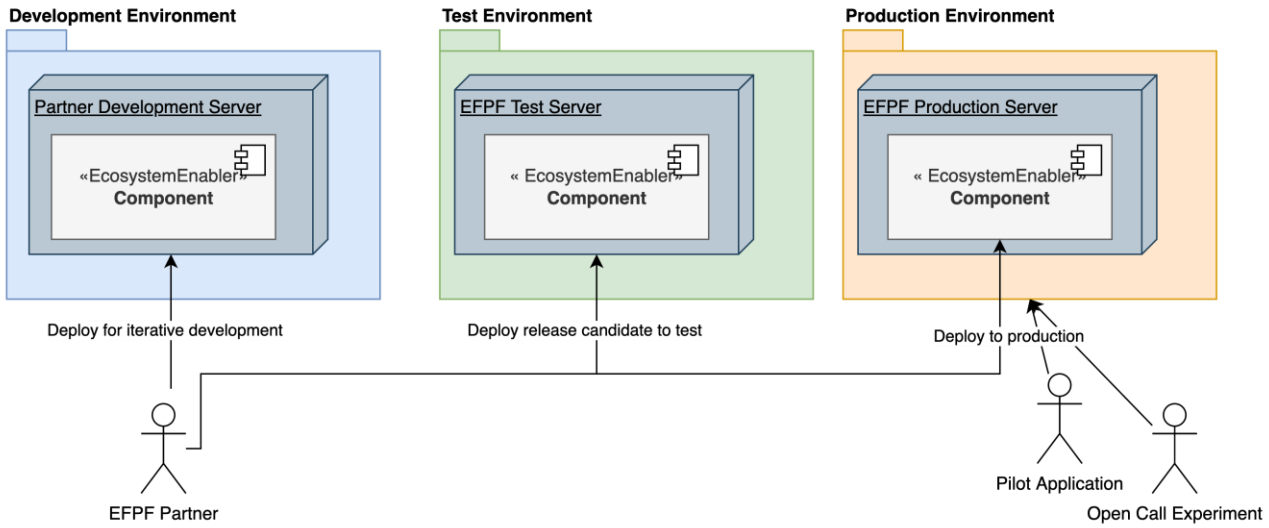

Figure 15. Runtime Platform Overview

3.2.1 Environments

Development of the EFPF Ecosystem and integration of base platforms started at the same time as the specification of deployment pipeline and runtime environment. The initial versions of the Data Spine, Portal and other central components were distributed over the hosting resources of the technical partners. In D6.1 – "EFPF Integration and Deployment I", this runtime environment was described. It was hosted on several distributed nodes. No common runtime platform, container technology or choice of hosting had been decided. This configuration now comprises the Development Environment described below.

A staging environment was necessary where tests could be run, and release candidates approved for production. This is called the Test Environment. It was decided that one stable environment could be used for both Pilots and Open Call experiments, the Production Environment. A limited number of runtime environments would require less effort to manage and support. It was a highly prioritized task to move all Data Spine components as well as the Portal and Marketplace components to the CI/CD Pipeline and deploy these together on the Test and Production Environments.

The decision was made to use on-premises hosting at C2K rather than relying on cloud resources, e.g., AWS or Azure Cloud. This was motivated by having predictable costs and a guaranteed continuous environment when transitioning to EFF.

It was decided to limit Test and Production hosting to the Ecosystem Enablers, specifically the Data Spine and Portal. This was to ensure performance efficiency and co-existence by

having sufficient resources for the Data Spine and Portal, that provide identity management, integration, interoperability, service composition and the main entry point to the system. The common runtime platform and container technology specified in this document are mandatory only for components deployed in the Test and Production Environments. EFPF Platform Tools and Services are integrated with the Portal and Data Spine but may choose other development and deployment solutions that have already been invested in and fit the rest of the company product portfolio, as stated in the DOA. This means having a coherent development and deployment architecture for the central components of the ecosystem while lowering the barrier to entry for tool and services that wish to be part of the ecosystem.

The choice of a common container technology provides maintainability and portability for the Ecosystem Enablers. The project selected the widely used Docker container technology and after evaluating Kubernetes, decided on Docker Swarm for container management. The automated and configurable deployment pipeline together with the container technology enables EFPF and EFF to migrate to cloud hosting or deploy multiple instances of the platform if this is desired. E.g., the on-premises hosting can be complemented with single-tenant instances of the Data Spine hosted in the cloud for resource demanding customers or applications.

The diagrams have been simplified and only Ecosystem Enablers are shown. The infrastructure and container management tools – Docker Swarm, Portainer, Nginx - located on the test and production virtual machines with Data Spine have been omitted from the diagrams. For detailed information, see Annex D.

3.2.1.1 Development environment

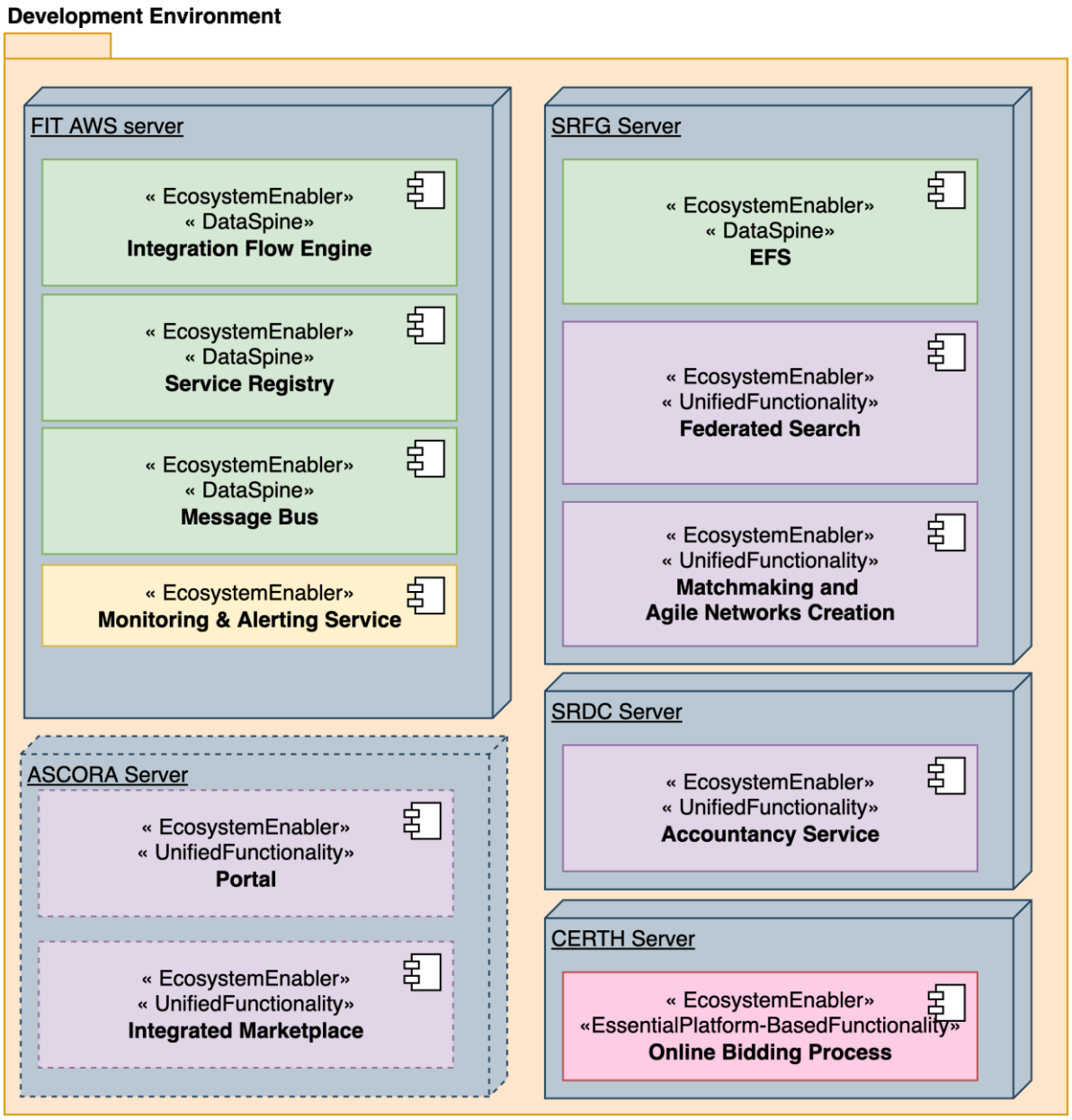

Figure 16. The Development Environment

The development environment was originally set up as a testbed by integrating the deployments of the initial versions (major version 0) of the components in the early stages of development. The main concern was to have a platform available for evaluating integration and technology choices as early as possible in the project, without waiting for design decisions on deployment pipeline, container technology, hosting, or common standards. Iterative development of components, interfaces and user interaction is done in this environment. Hosting is distributed even on sub-component level and components are generally hosted at the main responsible development partner. There are no policies guaranteeing that the deployed version of any component is stable, ad hoc deployment is allowed, and the configuration of services, network, or security may change without notice. However, in practice, no significant changes to the development environment architecture

have been made since deliverable D6.1 and the core components (Ecosystem Enablers) deployed there have been stable. (A high level of intra-project co-operation and collaboration have also helped to quickly resolve any incompatibilities.) Due to the schedule of the Test and Production hosting procurement and set-up of the Test and Production environments, earlier stages of the Pilots were integrated with the Development environment.

In the diagram above, only Ecosystem Enablers are shown. The development versions of Portal and Integrated Marketplace have been taken offline and now only deploy using Test and Production Environments. The Development Environment has not seen much use in the later stages of the project and can be taken offline.

3.2.1.2 Test environment

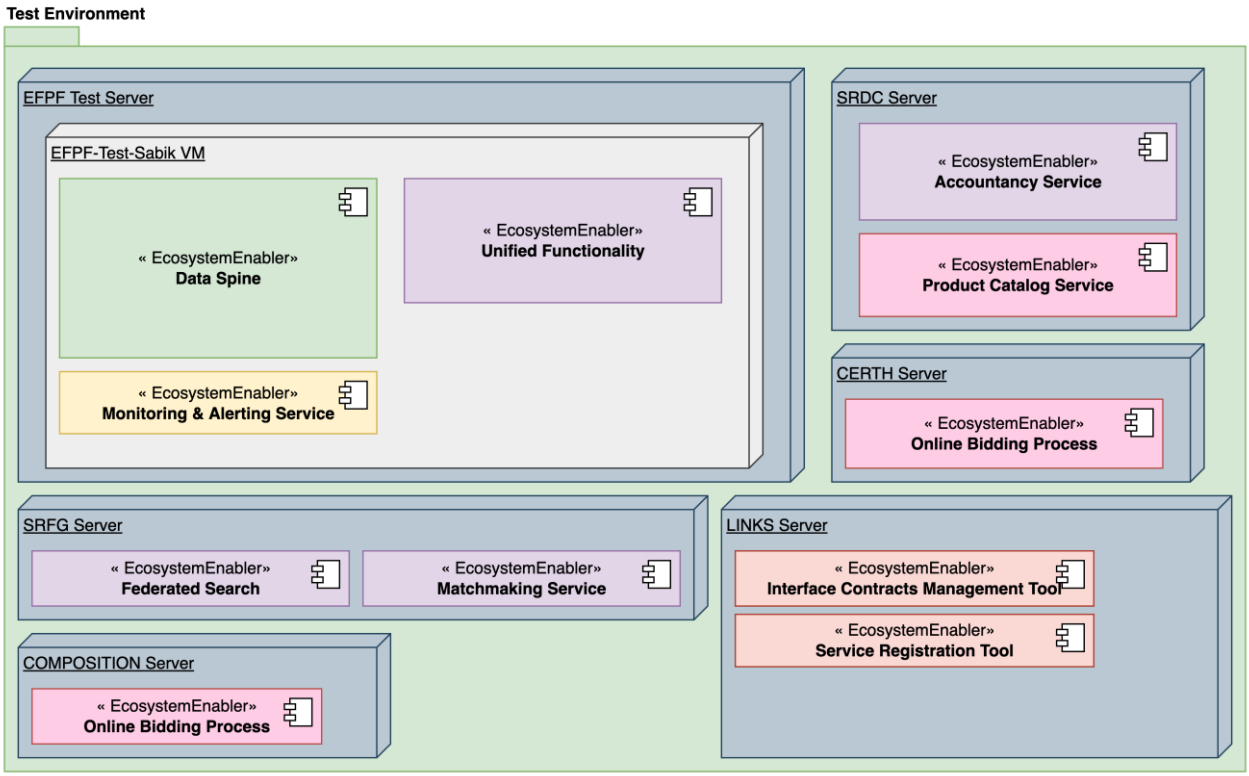

Figure 17. The Test Environment

A staging environment for testing the release candidates of the platform components was set up and hosted by C2K on-premises. To ensure reasonable resource requirements and guarantee stable operation, the Test environment was reserved for deployment of the Ecosystem Enablers. The specified CI/CD pipeline must be used to deploy in the Test Environment. Release candidates are deployed to the Test environment according to the respective release schedules. Test procedures are run and integration tests for dependent components can be made against the Test server versions. The EFPF components to be tested for integration – tools and services - will be hosted on other servers, however. Release candidates that pass the tests can be deployed in the Production Environment. The Test environment is currently used as a staging environment only by EFPF components but may in the future also be used for externally developed tools and services.

The resources assigned to the Test environment have been upgraded to accommodate for test procedures and increased use by dependent components.

At the time of writing, the Test Environment server is located at C2K premises, and its configuration is:

- OS: Windows Server 2019
- RAM: 64 GB
- Storage: 556 GB

The EFPF components are deployed in virtual machines on Windows Server 2019:

VM Name	Domain	Public IP	RAM	CPU	Disk Usage
EFPF-Test-Sabik	ds-test.smecluster.com	62.232.213.5	24 GB	6	125 GB

3.2.1.3 Production Environment

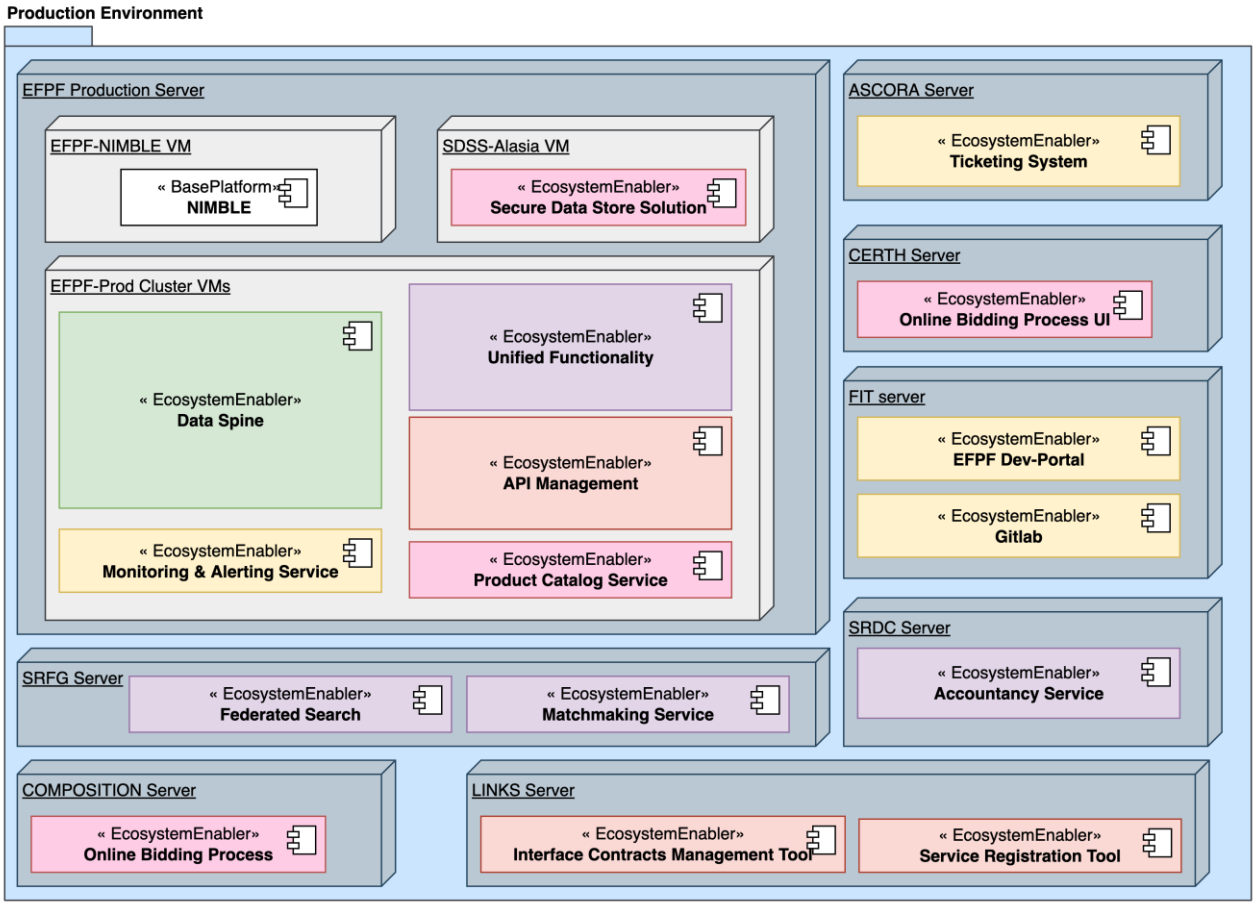

Figure 18. The Production Environment

The Production Environment hosts the stable versions of the Ecosystem Enablers. This primarily means the Data Spine and Portal. However, other Ecosystem Enablers can be

hosted there if they implement the CI/CD pipeline and container technology. If they have high cohesion with the Data Spine and Portal and resource consumption is estimated to be low and stable, they may be hosted together with the Portal and Data Spine on the same nodes. Otherwise, a separate virtual machine is set up on the server. For the EFF, codeline organization may be a factor as well – deploying proprietary code together with EFF managed code may introduce a support problem. Performance efficiency and reliability for the Data Spine and Portal is the deciding factor when allowing other components to be deployed in the Production Environment.

The resources assigned to the Production environment was initially based on an estimate of the requirements of the most resource consuming sub-components of the Data Spine, the Message Bus and the Integration Flow Engine, given data throughput in number of events and megabytes of data per second. A reliable estimate was found to be hard to provide as it depends on the usage patterns of these components and the optimizations that can be made. The resources available in the Production Environment have been upgraded to match actual resource requirements during the Open Call experiments.

Migration of the NIMBLE service used in federated search - currently hosted by SRFG – to the NIMBLE instance on the production server is in progress. Only the production server instance is shown in the diagram.

At the time of writing, the Production Environment server is located at C2K premises, and its configuration is:

- OS: Windows Server 2019
- RAM: 176 GB
- Storage: 2 TB

The EFPF components are deployed in virtual machines on Windows Server 2019:

VM Name	Domain	Public IP	RAM	CPU	Disk Usage
EFPF-NIMBLE	nimble.smecluster.com	62.232.213.5	16 GB	1	100 GB
EFPF-Prod-Master	efpf.smecluster.com	62.232.213.8	24 GB	8	100 GB
EFPF-Prod-W1	efpf.smecluster.com	62.232.213.8	24 GB	8	100 GB
EFPF-Prod-W2	efpf.smecluster.com	62.232.213.8	49 GB	8	100 GB
SDSS- Alasia	sdss.tools.smecluster.com	62.232.213.6	8 GB	1	50 GB

The Data Spine and Portal are deployed on one master and two worker nodes. The NIMBLE instance and the Secure Data Store Solution (SDSS) are deployed separately.

The inter-factory marketplace services of the COMPOSITION base platform have been integrated into the EFPF Unified Functionality and the instance that was connected to EFPF during the first phases of the project has been shut down. Other instances of the COMPOSITION platform – which was designed as single tenant, with multiple per-factory instances connected by the inter-factory services – may connect to EFPF is they choose to do so.

The Accountancy Service, Federated Search and Matchmaking Service are located on project partner servers. The possibility of migrating these to the Production Environment server should be investigated.

3.2.2 Container Technology, Orchestration and Management

The EFPF ecosystem is composed by a set of microservices developed and deployed following the EFPF CI/CD System explained above. Most of the source code is hosted on the EFPF Gitlab instance, which offers the possibility to automatically build and release the software through its embedded CI/CD system.

Once the build system has produced the artifacts (mostly docker images) to be deployed in the target environment, the choice of the orchestration and management technology is crucial to ensure the stability and the sustainability of the services over the time.

An orchestration tool suitable to host the EFPF services must ensure:

- **Ease to use:** The EFPF integration and deployment routine will be used to expert and non-expert end users. The container technology must be a good tradeoff between easy to use and overall maturity/stability
- **Ease to maintain:** The EFPF platform will be composed by a huge number of different services, with different virtualization entities (virtual networks, virtual disks like docker volumes and so on). An easy to maintain platform reduces the risks of services downtime.
- **Maturity:** Since the EFPF is a ready-for-production platform, it should ensure an high level of stability and maturity
- **Zero downtime update:** Most of the EFPF services must be up and running 24/7. The orchestrator must ensure zero-downtime when a service is updated, scaled or some configuration change.
- **Ease to scale:** The orchestration technology must offer an easy to scale clustering solution, to provide a smart way to increase the number of “workers” VMs

The next table shows the overview of the two EFFP environments:

Environment	# VMs	vCPU	RAM	Disk Size	Orchestrator
Production	3	8	24GB	100GB	Docker Swarm
Test	1	8	24GB	100GB	Docker Compose

The Production environment offers **Docker Swarm** as container orchestrator technology. Swarm is a container orchestrator based on a “multiple masters multiple workers” clustering system. Workers can be dynamically plugged into the cluster; one worker can dynamically become a master. The maintenance of the distributed system is ensured by the master nodes, which can be interrogated to

- Add a new node to the cluster
- Change the role of a cluster node (master to worker and vice versa)
- Assign a label to a cluster node (labels can be used to tag a node and distinguish it from the rest of the cluster)
- Perform all the deploy and maintenance operations on the services running in the cluster, including
    - Deploy a service or a stack

- Scale a service
- Update a service or a stack
- Remove a service or a stack
- Inspect a service or a stack, to view the actual configuration
- Migrate a service or a stack from a node to another

The operations explained above can be performed using the Docker Swarm CLI. It's common to implement, along with the CLI, an Orchestration Management System: on the EFPF Platform there is an instance of Portainer 2.0, which offers the full compatibility with Swarm cluster distributed system. This service can be used to execute almost every task to manage the swarm cluster and its services, including the real time view of the logs related to a Docker Service. The implementation of a Cluster Management System is also a good opportunity to ensure a higher level of security of the overall platform: it's possible to define users that can directly interact with the host VMs (through a SSH access) and, in the meanwhile, users that can interact with (some of) the EFPF services from a web browser. This possibility suggested also the implementation of container isolation mechanisms: authentication along with authorization are the base mechanisms that uniquely identify a certain user and are used to understand the set of operations and permissions that this user can perform to the services on top of the swarm cluster.

The EPFF Test Environment instead, is based on a single node architecture. It is used to perform the deployment and integration tests needed to ensure the correct stability of the service deployed in the Production Environment. The compatibility between test and production environment is ensured by the implicit compatibility between the Docker Compose and Docker Swarm systems: Swarm is an improvement of Compose, it adds all the clustering features, as well as some minor features. The description of a Docker Compose and a Docker Swarm multi-container service are the same: the standard YAML docker-compose descriptor.

With this assumption, deploying services on the EFPF platform require the effort to correctly describe the features and the configuration of the application through a single descriptor that can be tested and integrated on the Test Environment and, if the acceptance tests pass, finally deployed on the production environment

3.3 Dependencies

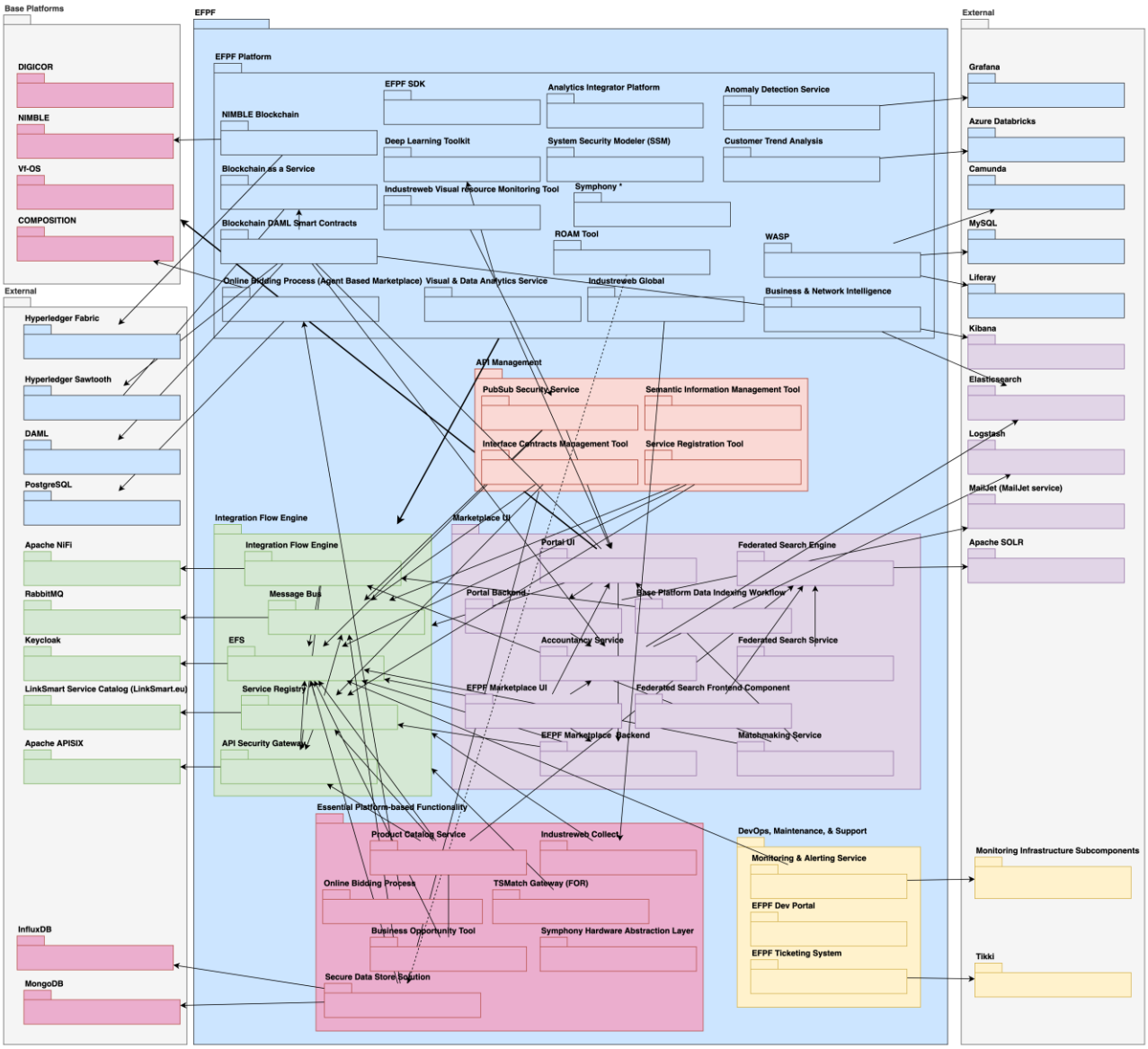

Figure 19. Dependency Diagram

The purpose of the dependency diagram is to give stakeholders an overview of internal dependencies in the EFPF Ecosystem, and dependencies to external components. The dependencies in Ecosystem Enablers are the most important for ecosystem administrators and architects and should be updated frequently. Dependency information for the EFPF Platform is of interest also to system integrators, however, it is the responsibility of tools and service owners to provide and update this information.

A dependency indicates that a change to the target element (indicated by the direction of the line) in the dependency may require a change to the source element in the dependency. This can be used to trace the impact of changes or necessary updates to dependencies, e.g., due to security issues.

The information is not represented on the API level; the purpose here is to trace dependencies. The color-coding for ecosystem elements of the Context View is used. For readability, dependencies valid for all components in a group (e.g., to Data Spine from

EFPF Platform) are shown as a dependency on group level with a slightly thicker line. The diagram necessarily has a lot of dependencies and can be hard to read - it should be used as a start when analysing dependencies; dependency tables with detailed information from the technical partners are available in Annex E.

4 After EFPF: Recommendations for EFF and Further Work

The development and deployment architecture has been continuously revised and upgraded after releases and during operation. The Pilots and Open Call experiments have been very useful to test and validate the architecture. Encountered issues have been evaluated and have resulted in system upgrades and improvement issues being implemented and added to the backlog. This section will list lessons learned and recommended actions for EFF, some of which can be implemented before the end of the project.

4.1 Recommendations and Lessons Learned

In the EFPF Ecosystem, user creation is provided via the EFPF portal, the API Management Tools has provided Message Bus topic creation and access control and service registration. However, significant effort has been spent setting up access control, SSO clients and additional user accounts and access control to tools for the Open Call experiments. The setup of additional users, access rights, SSO clients and distribution of client secrets, additional Message Bus vhosts, and other steps that have been performed by the support staff during should be automated and customer self-help tools should be provided, developed to match the EFF business model. Automating such tasks will require less effort per customer for EFF operations and support staff.

The code of the Data Spine Enablers is Open Source, as is the case for most of the Ecosystem Enablers. For those components that are not open source, the EFF needs to assess if this should be requested or if the current state is satisfactory.

The Gitlab instance used as code and image repository and to run the CI/CD pipeline is currently hosted by FIT and needs to be migrated (copied) to EFF servers at the end of the project. Replacing Gitlab requires a lot of work since it is integral to the deployment pipeline, the assessment of the operations team is that this is not feasible.

The Tikki ticket management system hosted by ASC needs to be migrated to EFF servers or replaced at the end of the project. The current support organization and process could be used with other ticketing systems.

The development environment has been largely used as a sandbox environment where prototypes and experimental versions of components can safely call other components and try out services and do function tests during development without disturbing operations. The test environment could be used this way, but there is a risk that this may interfere with tests. It is up to EFF to decide if a sandbox environment is useful and desirable. It should only need to run on minimal specifications, with restrictions set on Message Bus and Integration Flow Engine use.

Current server resources have been adequate for the Pilots and Open Call experiments, although the system has successfully been vertically scaled during operation. However, in commercial operation the resource requirements may become even higher, and some customers may require reserved capacity for their applications. Additional scaling strategies and deployment of multiple instances for single-tenant use or multi-tenancy

performance isolation and quotas should be explored by EFF. The architecture supports this, it is a matter of providing the administration and configuration necessary.

The architecture ensures that zero downtime releases are enabled for Data Spine, to avoid interruption in data traffic and processing. However, for upgrades of hardware resources or assignment of new hardware resources to the virtual machine, downtime is necessary. In a commercial deployment, redundancy in hardware is required to enable Data Spine upgrades without downtime.

The experience from the project has been that it is hard to maintain up-to-date information on components in a distributed system such as the EFPF ecosystem. Management of component configuration on deployment pipeline level is done in Gitlab, using variables. This environment provides good control and overview for the detailed configuration for Test and Production environments. The use of Ansible scripts would make Ansible a natural alternative for configuration management tool for the Data Spine, to be used by the maintenance and operations team. The Service registry and associated tools also provide information on run-time configuration of published services. However, high-level overview of development and deployment view (e.g., ownership, versioning, release schedules) of all components in the ecosystem has been done via spreadsheets, which is hard to maintain and query. It is recommended that EFF acquires or develops a high-level configuration management database that can keep track of this information and produce reports for the EFF ecosystem architects.

Before handover, testing of all backup and restore routines for the core containers and review of their documentation should be performed together with EFF staff.

4.2 Technical Meeting Outcomes

At the end of June 2022, a technical meeting was held to review the status of development and deployment of Data Spine and EFPF Ecosystem and the future hosting, management and sustainability of the Data Spine, other Ecosystem Enablers and EFPF Platform tools and services. Decisions made during this meeting impact the development and deployment architecture of EFPF and will be reported here.

Major decisions included the close-down of the development environment, which is no longer being used and will be shut down end of August 2022. The Ticketing System (Tikki) is using EFS in the development environment – the realm database is mirrored there and this setup was found more resilient when tracking issues related to production EFS itself - so it will need to be reconfigured.

A few improvements in the deployment and runtime are in progress, these were discussed and found desirable but not critical, and will be continued for the remainder of the project. The migration of tools and services to before the end of the project and EFF handover was of highest priority.

Essential tools in for development, deployment and support that need to be migrated to the production server (see section 3.2.1.3) are:

- GitLab SCM
- GitLab Container Registry
- GitLab CI/CD
- EFPF Development Portal

- Tikki
- Monitoring Services

It was also found desirable to move the Accountancy Service (currently at SRDC) to the EFPF Production Environment so that it comes under the ownership and management of EFF. SRDC will specify the resource requirements of Accountancy Service to assess whether it can be moved to the Production environment and discuss this with the maintenance and operations team.

Services from a COMPOSITION instance running on FIT AWS is used in the production environment. E.g. the Circular Economy Pilot uses this instance. Migration of these services to production server will be investigated.

Migration of the NIMBLE service used in federated search, currently hosted by SRFG, to the NIMBLE instance on the production server is in progress.

Tikki is hosted and owned by ASC. The project will investigate other options for realization of the ticketing system for EFF, e.g., Gitlab.

Knowledge transfer to EFF and additional documentation of operational procedures for the Data Spine should take place during M43-48 of the project. Roles, responsibilities and required skills in deployment, operation and maintenance are being documented.

5 Conclusion and Outlook

This deliverable has reported the work performed and status of the development and deployment architecture of the EFPF ecosystem. It will be used to assess the steps that need to be taken to transfer the operations of the ecosystem to the EFF. The information provides an overview of code organization, dependencies, runtime environments where desired changes and migrations can be easily identified by EFF. The delivery pipeline provides automated, configurable deployment of the Ecosystem Enablers that enables instances to be deployed swiftly and on different hosting infrastructure. The operational infrastructure and component management enables the system to be horizontally scalable and resilient and is ready for commercial operation. The infrastructure may need to be further vertically scaled and mechanisms for single-tenant hosting explored for EFF to meet the demands of large-scale business operation. EFF prioritized tasks for migration, additional features and improvements and knowledge transfer before handover from EFPF to EFF will be the focus of the remaining work in WP6.

Annex A: Document History

Document History	
Versions	V0.1: Document set-up and draft Table of Contents V0.2: Additional content added V0.3: FIT contributions integrated V0.4: LINKS contributions integrated V0.5: NXW contributions integrated V0.6: Additional editing, diagrams, spreadsheet information V0.7: Additional editing V0.8 Updates from Technical Meeting, tables, diagrams V1.0 Ready for internal review
Contributions	CNET: • Mathias Axling • Matts Ahlsen FIT: • Rohit Deshmukh • Alexander Schneider NXW: • Gabriele Scivoletto • Gianluca Insolvibile LINKS: • Edoardo Pristeri

Annex B: References

[Hil00] Hilliard, Rich. "Ieee-std-1471-2000 recommended practice for architectural description of software-intensive systems." IEEE, http://standards. ieee. org 12.16-20 (2000): 2000.

[IEEE 42010, 2011] May, I. S. O. Systems and software engineering–architecture description. Technical Report. ISO/IEC/IEEE 42010, 2011.

[RW12] Rozanski, Nick, and Eoin Woods. "Software Systems Architecture: Viewpoint Oriented System Development." (2012).

[KRU04] Kruchten, P. (2004). The Rational Unified Process: An Introduction. Addison-Wesley Professional.

[HF10] Humble, J., Farley, D (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Professional.

[ISO11] ISO (2011). ISO/IEC 25010:2011. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. Accessed Sept 2019.

[DOC19] Docker Enterprise Container Platform. https://www.docker.com/ Accessed Sept 2019.

[POR19] Portainer for Docker Management. https://www.portainer.io/ Accessed Sept 2019.

[KEY19] Keycloak Open-source identity and access management solution https://www.keycloak.org/ Accessed Sept 2019.

[NIM22] Nimble (Collaboration Network for Industry, Manufacturing, Business and Logistics in Europe) Project. Available online: https://www.nimble-project.org/ (accessed on 1 June 2022).

[COM22] COMPOSITION (Ecosystem for Collaborative Manufacturing Processes) Project. Available online: https://www.composition-project.eu/ (accessed on 1 June 2022).

[DIG22] DIGICOR (Decentralised Agile Coordination Across Supply Chains) Project. Available online: https://www.digicor-project.eu (accessed on 1 June 2022).

[VFO22] vf-OS (Virtual Factory Operating System) Project. Available online: https://www.vf-os.eu (accessed on 1 June 2022).

[VLC22] ValueChain's Network Portal platform. Available online: https://valuechain.com/products/network-portal/ (accessed 1 June 2022).

[NXW22] Nextworks' Symphony platform. Available online: https://www.nextworks.it/en/products/symphony (accessed 1 June 2022).

[C2K22] SMECluster's IndustreWeb platform. Available online: https://www.industreweb.co.uk/ (accessed 1 June 2022).

Annex C: Development Viewpoint Table

Context View Element	Name	Subcomponent	Development Team	Owner-ship	Code repository	Container registry	Versioning	Code Quality	Delivery pipeline	Delivery schedule
Data Spine	Data Spine		FIT, SRFG, NXW, ASC	FIT	EFPF (https://gitlab.fit.fraunhofer.de/)	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-integration-and-deployment/container_registry/)		The contributions to the source codes of the Data Spine components are reviewed and quality checked by the respective developers, reviewers and maintainers. The development and quality assurance process differs for different Data Spine components: Apache NiFi follows a Review-Then-Commit (RTC)[1] process; RabbitMQ follows a multi-step development process[2] that involves pull requests to discuss, review, collaborate on and accept code contributions; LinkSmart® Service Catalog follows a similar development and review process[3] that involves contribution through pull requests as well. The contributions are also checked against the defined acceptance criteria during these processes. [1] https://cwiki.apache.org/confluence/display/NIFI/Contributor+Guide [2] https://github.com/rabbitmq/rabbitmq-server/blob/master/CONTRIBUTING.md [3] https://github.com/linksmart/service-catalog#contributing	EFPF	Phase 1-2, no fixed schedule after that.
Data Spine	Integration Flow Engine (IFE)	Integration Flow Engine	Apache nifi Team[1] and community [1] https://nifi.apache.org/people.html	Apache Software Foundation	Public				EFPF	When needed.
Data Spine	Service Registry	Service Registry	Team at Fraunhofer FIT and community	Fraunhofer FIT	Public				EFPF	When needed.
Data Spine	Message Bus	Message Bus	Team at Pivotal Software and community	Pivotal Software, Inc.	Public				EFPF	When needed.
Data Spine	EFPF Security Portal (EFS)	EFPF Security Portal (EFS)	Team at SRFG and JBoss	SRFG	EFPF (https://gitlab.fit.fraunhofer.de/)	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-integration-and-deployment/container_registry/efpf_keycloak)			EFPF	When needed.
Data Spine	API Security Gateway	API Security Gateway	Team at SRFG and Apache developers	SRFG	Public				EFPF	When needed.
Unified Functionality	EFPF Portal	EFPF Portal UI	ASC (task lead), VLC	ASC	EFPF (https://gitlab.fit.fraunhofer.de/)	EFPF (https://gitlab.fit.fraunhofer.de/)	No versioning strategy defined.	No coding standards defined.	EFPF on GitLab	When needed.
Unified Functionality	EFPF Portal	EFPF Portal Backend	ASC (task lead), VLC	ASC	EFPF (https://gitlab.fit.fraunhofer.de/)	EFPF (https://gitlab.fit.fraunhofer.de/)	No versioning strategy defined.	No coding standards defined.	EFPF on GitLab	When needed.
Unified Functionality	Accountancy Service		SRDC	SRDC	SRDC Private Repository	DockerHub Elasticsearch: https://hub.docker.com/_/elast	A new release is created only when an update is implemented on the Accountancy Service, since there are no	N/A	All necessary dashboards developed on Kibana, based on	When needed

						icsearch Kibana: https://hub.docker.com/_/kibana	planned scheduled updates.		a data model and no updates are foreseen. The data modal and dashboards are updated and deployed when there are unexpected problems happened during integration. Delivery is realized after an update is implemented when needed.	
Unified Functionality	EFPF Marketplace	EFPF Marketplace UI	ASC (task lead), SRDC	ASC, SRFG, CERTH, ISMB, SRDC (Accountancy Service)	EFPF (https://gitlab.fit.fraunhofer.de/)	N/A	No versioning strategy defined.	No coding standards defined.	Manual deployment	When needed.
Unified Functionality	EFPF Marketplace	EFPF Marketplace Backend	ASC (task lead), SRDC	ASC, SRFG, CERTH, ISMB, SRDC (Accountancy Service)	EFPF (https://gitlab.fit.fraunhofer.de/)	EFPF (https://gitlab.fit.fraunhofer.de/)	No versioning strategy defined.	No coding standards defined.	EFPF on GitLab	When needed.
Unified Functionality	Federated Search	Federated Search Engine	SRFG (task lead), CERTH	SRFG	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/matchmaking-services)	DockerHub: https://hub.docker.com/r/nimbleplatform/matchmaking-service				When needed.
Unified Functionality	Federated Search	Base Platform Data Indexing Workflow	SRFG (task lead), CERTH	SRFG	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/matchmaking-services)	N/A				When needed.
Unified Functionality	Federated Search	Federated Search Service	SRFG (task lead), CERTH	SRFG	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/matchmaking-services)					When needed.
Unified Functionality	Federated Search	Federated Search Frontend Component	SRFG (task lead), CERTH	SRFG	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-portal)					When needed.
Unified Functionality	Matchmaking Service		SRFG (task lead), CERTH	SRFG	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/matchmaking-services)			No coding standards defined.	Releases/ Updates Development branch including separate deployment as Docker container. When no main issues arise, merge dev branch into production branch and update production Docker container on DockerHub. Versioning No versioning strategy defined. Delivery	When needed.

									Schedule When needed.	
API Management	Pub/Sub Security Service	Front End	ICE, UoS, AID, VLC, ASC	ICE	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-portal)	EFPF https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-portal/container_registry Not available as standalone component		Initially deployed and tested on ICE Servers	EFPF on Gitlab	When needed
API Management	Pub/Sub Security Service	Backend	ICE, ASC	ICE	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-security-components/efpf-pub-sub-security/pub-sub-backend)	EFPF https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-security-components/efpf-pub-sub-security/pub-sub-backend/container_registry https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-integration-and-deployment/container_registry/560 Also available in ICE Docker Hub	latest	Initially deployed and tested on ICE Servers	EFPF on Gitlab	When needed
API Management	Pub/Sub Security Service	Database	ICE, ASC	ICE	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-security-components/efpf-pub-sub-security/pub-sub-backend)	EFPF/Docker Hub https://hub.docker.com/_/mongo https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-integration-and-deployment/container_registry/691 mongo:latest	latest	Initially deployed and tested on ICE Servers	EFPF on Gitlab	When needed.
API Management	Interface Contracts Management Tool		LINKS	LINKS	LINKS Public repository: https://bitbucket.org/links-foundation/icmt/src/master/	N/A	0.1	Tests available in repository	Other, manual	As needed
API Management	Service Registration Tool		LINKS	LINKS	LINKS Public repository: https://bitbucket.org/links-foundation/srt/src/master/	N/A	0.1-rc1	Tests available in repository	Other, manual	As needed
API Management	Semantic Information Management (SIM) tool		FIT	FIT	EFPF (https://gitlab.fit.fraunhofer.de/)					When needed.
DevOps, Maintenance, & Support	Monitoring & Alerting Service		FIT, (Deployment: NXW)	FIT	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-integration-and-deployment/-/tree/MI-test/Playbooks/Monitoring_Infrastructure, https://gitlab.fit.fraun	Public (DockerHub, quay.io - for auth2 proxy). Endpoints: https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-integration-and-deployment/-/wikis/Dep				When needed.

					hofer.de/efpf-pilots/monitoring-toolkit/-/tree/master/)	loyment-Environments				
DevOps, Maintenance, & Support	Monitoring & Alerting Service	Monitoring Server	Teams at Prometheus, Grafana and oauth2-proxy and community		Public					
DevOps, Maintenance, & Support	Monitoring & Alerting Service	Monitoring Data Collectors at Target Hosts	Teams at cAdvisor (Google), vector (Datadog, Inc. - vector.dev) and community		Public					
DevOps, Maintenance, & Support	Gitlab		Gitlab	Gitlab	Gitlab.com	Docker Hub	Docker image			
DevOps, Maintenance, & Support	EFPF Documentation Portal		EFPF	EFPF	EFPF (https://gitlab.fit.fraunhofer.de/)	EFPF https://gitlab.fit.fraunhofer.de/ https://gitlab.fit.fraunhofer.de/efpf-pilots/efpf-integration-and-deployment/container_registry/560 Also avail	Docker image			
DevOps, Maintenance, & Support	EFPF Ticketing System	Tikki	ASC	ASC	ASC Private Repository	ASC Private	UI			
Essential Platform-Based Functionality	Product Catalogue Service		SRDC	SRDC	NIMBLE Github Repository (https://github.com/nimble-platform/catalog-service)	Docker Hub: https://hub.docker.com/r/nimbleplatform/catalogue-service-micro	Versions are set at each deployment by updating Maven POM file.	No coding standards defined.	Jenkins continuous integration	When needed.
Essential Platform-Based Functionality	Secure Data Store Solution		ASC, ISMB, UOS-ITI	ASC	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/secure-data-store-solution/)	EFPF (https://gitlab.fit.fraunhofer.de/)	SDSS API No versioning strategy defined.	No coding standards defined.	Gitlab CI/CD	When needed
Essential Platform-Based Functionality	Industreweb Collect		C2K	C2K	Private	N/A	User definable data model including OPC UA Communication protocol - Format is JSON by default but can be defined by user Production data model - ISA-95 inspired model–used in DIGICOR New versions are managed in the development environment and stable versions deployed automatically on new installed nodes, and upgraded manually where licensed upgrades are included	The component initially is deployed on a C2K test branch for functionality testing by an in-house test team, and stakeholders. Once approved changes are merged with Live branch and deployed to the live environment for final test. Simple functional testing against acceptance criteria defined for each Jira issue.	Jenkins continuous integration. Individual deployment manually on local nodes.	Continuous
Essential Platform-Based Functionality	TSMatch Gateway		FORTISS	Owned by fortiss, code available under	FORTISS https://git.fortiss.org/iot_external/tsmatch	FORTISS Git		The Dynamic Factory Connector component is currently deployed at fortiss premises. When a new functionality is introduced, tests are first done locally; upon approval, integration with EFPF	The Dynamic Factory Connector is currently	When needed

				an MIT license				services is performed and tested (usually specific to the service and use case). Unit tests are used to examine the new functionality.	deployed at fortiss premises. Executable version is to be used for distribution with manual adjustment to accommodate the factory premises.	
Essential Platform-Based Functionality	Symphony Hardware Abstraction Layer (HAL)		NXW	All of the source code is IPR of Nextworks.	The source code is owned and managed by Nextworks, and all IPR is the property of Nextworks.		The versioning is based on Symphony release plan and schedule.	The components continuously test by internal test team.	Manual deployment	The releases and updates are based on symphony release plan and schedule.
Essential Platform-Based Functionality	Online Bidding Process	Front-end Interfaces	CERTH	CERTH	EFPF (https://gitlab.fit.fraunhofer.de/)		2	Integration testing with LINKS' agents + testing by end-users of CE pilot	EFPF	When needed
Essential Platform-Based Functionality	Online Bidding Process		CERTH	CERTH	CERTH Private Repo - Gitlab	DockerHub	API, REST with HTTPS V1.0 Data format for information exchange JSON V1.0 - Similar to Agents Exchange Language format from COMPOSITION Security protocol Basic Auth using OpenID Connect V1.0 The component provides a REST API and communication protocol is HTTPS. The security protocol is Basic Auth by using OpenID Connect. Data format for information exchange is JSON. The format is similar to Agents Exchange Language format from COMPOSITION. Git is used for control versioning. In particular, the EGit plugin from Eclipse IDE is used. It fits well as the project is developed using Eclipse IDE and Maven as build tool for dependency management and building of source code Every component is internally tested and after that it is tested in integration with LINKS agents.	Automated tests in source code package were created by Maven, extended JUnit , Jetty server. Software quality assurance by both static and dynamic analysis techniques, e.g. PMD tool.		initially Fixed schedule based on project time plan. After that "When needed. "
Essential Platform-Based Functionality	Online Bidding Process	Matchmaking engine for EFPF Online Bidding Process	CERTH	CERTH	CERTH Private Repo - Gitlab	DockerHub	API, REST with HTTPS V1.0 Data format for information exchange JSON V1.0 - Similar to Agents Exchange Language format from COMPOSITION Security protocol - Basic Auth using OpenID Connect V1.0	Automated tests in source code package were created by Maven, extended JUnit , Jetty server. Software quality assurance by both static and dynamic analysis techniques, e.g. PMD tool.		initial fixed schedule according o project plan, then "When needed"
Essential Platform-Based	Business Opportunity Tool	Team Formation Tool	C2K	C2K	C2K Private	N/A				Ad hoc release process.

Functionality										
EFPF Platform	Online Bidding Process	Agent-based Marketplace	LINKS	LINKS	Public (Github)			N/A	Other, manual	When needed
EFPF Platform	Anomaly Detection Service		ICE	ICE	ICE Gitlab Repository: https://git.icelab.cloud	DockerHub: https://docker.hub.icelab.cloud/repository/analytics	No versioning strategy defined.	The component is first test locally, then deployed and tested in ICE Servers	ICE Gitlab	When needed
EFPF Platform	Customer Trend Analysis	Elanyo EFPF Behavioral Predictive Framework (BPF)	ELN	ELN			-			Ad hoc release process.
EFPF Platform	Blockchain & Smart Contracting	Blockchain as a Service (BaaS)	CERTH	CERTH	CERTH Private Repo - Gitlab		BaaS APIs V2.0 (Actually Supply-chain as a service API dedicated to EFPF)	Smart contracts determinism and finality can be proven with tests.	Based on project's schedule	Agile, as required.
EFPF Platform	Blockchain & Smart Contracting	Blockchain DAML Smart Contracts	CNET	CNET	CNET Bitbucket	N/A	DAML JSON API v1.17.0 DAML DAR files v1.17.0	* Not using EFPF CI/CD * DAML scenario testing	Other	When needed
EFPF Platform	Blockchain & Smart Contracting	NIMBLE Blockchain	SRFG	SRFG	Github (https://github.com/nimble-platform/logistic-contract)	N/A	Ledger API		Other	When needed
EFPF Platform	Business & Network Intelligence	BNIDashboard	ICE	ICE	ICE Gitlab Repository: https://git.icelab.cloud/bni/bnidashboard	DockerHub: https://hub.docker.com/repository/docker/informationcatalyst/bni-dashboard	No versioning strategy defined.	The component is first test locally, then deployed and tested in ICE Servers	ICE Gitlab	When needed
EFPF Platform	Business & Network Intelligence	ELK stack (Elasticsearch, Kibana) In development and testing phase	ICE	ICE	ICE Gitlab Repository: https://git.icelab.cloud/bni/elk-stack	DockerHub Elasticsearch: https://hub.docker.com/_/elasticsearch Kibana: https://hub.docker.com/_/kibana	7.16.0	The component is first tested locally, then deployed and tested on ICE Servers	ICE Gitlab	When needed
EFPF Platform	Visual & Data Analytics Service	CERTH/COMPOSITION Data and Visual Analytics	CERTH	Algorithms/Python scripts by CERTH - No provision of source code. Pseudo codes and algorithms in COMPOSITION webpage. Methodologies published in scientific publications.	CERTH Private Repo - Gitlab	DockerHub	UI services: fill level sensors' monitoring and trend analysis of fill level, tonnage forecasting, price forecasting based on Deep Learning, vibration sensors monitoring and vibration profile real time analysis.	Unit, regression, acceptance testing. Integration test with LINKS DLT. Code quality - analysis for errors detections based on PyFlakes library (https://github.com/PyCQA/pyflakes)	Test cycle using staging server. Pilot users are informed ahead (1-2 weeks) of version upgrade. Complete support for errors and bugs fixing is provided by CERTH continuously.	When needed.
EFPF Platform	Deep Learning Toolkit		LINKS	LINKS	LINKS Public repository: https://bitbucket.org/links-foundation/deep-learning-toolkit/src/master/	N/A	0.2-rc2	Tests available in repository	Other, manual	On demand
EFPF Platform	Industreweb Visual Resource Monitoring Tool		C2K	C2K	C2K Private	N/A				When needed.
EFPF Platform	Industreweb Global		C2K	C2K	C2K Private	N/A				Ad hoc release process.
EFPF Platform	Industreweb Visual Resource Monitoring Tool		C2K	C2K	C2K Private		New versions are managed in the development environment and stable versions	In-house C2K test team, and stakeholders. Approved changes are merged with Live branch and deployed to TEST for final test. Simple functional testing against	Jenkins continuous integration.	Continuous delivery is used.

							deployed automatically on new installed nodes, and upgraded manually where licensed upgrades are included .	acceptance criteria defined for each Jira issue.	Individual deployment manually on local nodes.	
EFPF Platform	Risk, Opportunity, Analysis and Monitoring (ROAM) Tool		ALM	ALM	EFPF (https://gitlab.fit.fraunhofer.de/efpf-pilots/roam)	There are old containers hosted on DockerHub, but they can be updated if necessary. The current containers are automatically built on our server.	API Version 1.0. API endpoints need to include a /v1/. We do not plan to change this for the remainder of the project.	The tool is first deployed locally for testing of frontend features and integration with other tools/services. Backend functionality is tested with unit tests.	Gitlab CI/CD	When needed.
EFPF Platform	Software Development Kit (SDK)	Software Development Kit (SDK) Frontend Editor	SIE	SIE	EFPF (https://gitlab.fit.fraunhofer.de/)	DockerHub	Gitlab	The features are tested by the internal team.	SDK	Ad hoc release process.
EFPF Platform	Software Development Kit (SDK)		CMS	CMS	CMS Private	CMS Private				When needed.
EFPF Platform	System Security Modeler (SSM)		UoS-ITI	Ownership: UoS-ITI Licensing: Free use for e-Factory project partners and commercial agreement for others (cf DoA and Grant Agreement)	UoS-ITI Gitlab		SSM Software (UI)	Senior developer code checking before release	Continuous development (GitLab environment)	Ad hoc release process.
EFPF Platform	Symphony Data Storage		NXW	The source code is owned and managed by Nextworks, and all IPR is the property of Nextworks.	The source code is owned and managed by Nextworks, and all IPR is the property of Nextworks.		The versioning is based on Symphony release plan and schedule.	The components are tested continuously by the internal test team.	Manual deployment	The delivery is based on Symphony release plan and schedule.
EFPF Platform	Symphony Resource Catalogue		NXW	All of the source code is IPR of Nextworks.	The source code is owned and managed by Nextworks, and all IPR is the property of Nextworks.		The versioning is based on Symphony release plan and schedule.	The components continuously test by internal test team.	Manual deployment	The releases and updates are based on symphony release plan and schedule.
EFPF Platform	Symphony Event Reactor		NXW	The source code is owned and managed by Nextworks, and all IPR is the property of Nextworks.	The source code is owned and managed by Nextworks, and all IPR is the property of Nextworks.		The versioning is based on Symphony release plan and schedule.	The components are tested continuously by the internal test team.	Manual deployment	The delivery is based on Symphony release plan and schedule.
EFPF Platform	Workflow and Service Automation Platform (WASP)		ICE	ICE	ICE Gitlab Repository: https://git.icelab.cloud		Gitlab is used for versioning and keeping copies of older versions.	The testing of the WASP platform is carried out using Gradle. Gradle tests are used to test each Jar file which is bundled into the Docker file and then used in the Liferay parent image.	The testing of WASP is carried out on two separate servers, a test server and a productio	There is a fixed schedule for delivery with a new stable version release at the end of

									n server. The changes are first deployed on to the test server and if everything is working as expected than the stable version is deployed on the productio n server.	each continuou s developm ent cycle.
EFPF Platform	**Analytics Integrator Platform (AIP)**		**SIE**	**SIE**	**EFPF (https://gitl ab.fit.fraun hofer.de/)**	**AWS ECR**	**Gitlab**		**EFPF on Gitlab**	**Ad hoc release process.**

Annex D: Deployment Viewpoint Table

Context View Element	Name	Subcomponent	Execution environment	Hosted at node (DEV)	Hosted at node (TEST)	Hosted at node (PROD)
Data Spine	Data Spine			FIT AWS server SRFG Server	C2K TEST	C2K PROD
Data Spine	Integration Flow Engine (IFE)	Integration Flow Engine	Docker container	FIT AWS server	C2K TEST	C2K PROD
Data Spine	Service Registry	Service Registry	Docker container	FIT AWS server	C2K TEST	C2K PROD
Data Spine	Message Bus	Message Bus	Docker container	FIT AWS server	C2K TEST	C2K PROD
Data Spine	EFPF Security Portal (EFS)	EFPF Security Portal (EFS)	Docker container	SRFG Server	C2K TEST	C2K PROD
Data Spine	API Security Gateway	API Security Gateway	Docker container	SRFG Server	C2K TEST	C2K PROD
Unified Functionality	EFPF Portal	EFPF Portal UI	Docker container	N/A	C2K TEST	C2K PROD
Unified Functionality	EFPF Portal	EFPF Portal Backend	Docker container	N/A	C2K TEST	C2K PROD
Unified Functionality	Accountancy Service		Docker container	matrix.srdc.com.tr	matrix.srdc.com.tr	matrix.srdc.com.tr - Investigating move to C2K PROD
Unified Functionality	EFPF Marketplace	EFPF Marketplace UI	EFPF Portal (Embedded)	N/A	C2K TEST	C2K PROD
Unified Functionality	EFPF Marketplace	EFPF Marketplace Backend	Docker container	N/A	C2K TEST	C2K PROD
Unified Functionality	Federated Search	Federated Search Engine	Docker	Apache Solr instance accessible on: https://efactory-security-portal.salzburgresearch.at/solr	Apache Solr instance accessible on: https://efactory-security-portal.salzburgresearch.at/solr	Apache Solr instance accessible on: https://efactory-security-portal.salzburgresearch.at/solr
Unified Functionality	Federated Search	Base Platform Data Indexing Workflow	Workflow configuration in XML	Currently deployed in Nifi instance maintained by SRFG. This will be integrated into DataSpine (main Nifi instance in SMECluster server) Nifi instance	https://ds-test.smecluster.com/nifi/	To be integrated into DataSpine (main Nifi instance in SMECluster server) Nifi instance
Unified Functionality	Federated Search	Federated Search Service	Docker container	REST service deployed on the SRFG server, accessible on: https://efactory-security-portal.salzburgresearch.at/api/index/swagger-ui.html	REST service deployed on the SRFG server, accessible on: https://efactory-security-portal.salzburgresearch.at/api/index/swagger-ui.html	REST service deployed on the SRFG server, accessible on: https://efactory-security-portal.salzburgresearch.at/api/index/swagger-ui.html
Unified Functionality	Federated Search	Federated Search Frontend Component	Angular application	Web component integrated in the eFactory portal web application deployed on the ASC server, accessible on: https://efactory-portal.ascora.eu/simple-search		
Unified Functionality	Matchmaking Service		Workflow configuration in XML	Currently deployed in Nifi instance maintained by SRFG. This will be integrated into DataSpine (main Nifi instance in SMECluster server) Nifi instance	https://ds-test.smecluster.com/nifi/	To be integrated into DataSpine (main Nifi instance in SMECluster server) Nifi instance
API Management	Pub/Sub Security Service	Front End	EFPF Production	NA	NA	C2K PROD
API Management	Pub/Sub Security Service	Backend	EFPF Production	NA	NA	C2K PROD
API Management	Pub/Sub Security Service	Database	EFPF Production	NA	NA	C2K PROD
API Management	Interface Contracts Management Tool		Docker Container	LINKS Server	LINKS Server	LINKS Server
API Management	Service Registration Tool		Docker Container	LINKS Server	LINKS Server	LINKS Server
API Management	Semantic Information Management (SIM) tool			N/A (In development)	N/A (In development)	N/A (In development)
DevOps, Maintenance, & Support	Monitoring & Alerting Service		Docker Container	FIT AWS server	C2K TEST	C2K PROD
DevOps, Maintenance, & Support	Monitoring & Alerting Service	Monitoring Server	Docker Container	FIT AWS server	C2K TEST	C2K PROD
DevOps, Maintenance, & Support	Monitoring & Alerting Service	Monitoring Data Collectors at Target Hosts	Docker Container	FIT AWS server	C2K TEST	C2K PROD
DevOps, Maintenance, & Support	Gitlab		Docker Container	FIT AWS	FIT AWS	FIT AWS - to be migrated to C2K PROD
DevOps, Maintenance, & Support	EFPF Documentation Portal		Docker Container	FIT AWS	FIT AWS	FIT AWS - to be migrated to C2K PROD
DevOps, Maintenance, & Support	EFPF Ticketing System	Tikki		N/A	N/A	ASC
Essential Platform-Based Functionality	Product Catalogue Service		Docker container	N/A	SRFG Staging Server	C2K PROD
Essential Platform-Based Functionality	Secure Data Store Solution		Docker container	-	-	C2K PROD
Essential Platform-Based Functionality	Industreweb Collect		Windows, .Net	Industreweb Server within production facility	Industreweb Server within production facility	Industreweb Server within production facility
Essential	TSMatch		Windows,	Factory Connector component	Factory Connector component that	Factory Connector component that

Platform-Based Functionality	Gateway		Linux	that can be used in a production facility	can be used in a production facility	can be used in a production facility
Essential Platform-Based Functionality	Symphony Hardware Abstraction Layer (HAL)		Cloud	Cloud Node NXW	Cloud Node NXW	Cloud Node NXW
Essential Platform-Based Functionality	Online Bidding Process	Front-end Interfaces	Angular application	CERTH Server	CERTH Server	CERTH Server and accessible by EFPF PROD Environment
Essential Platform-Based Functionality	Online Bidding Process		Docker Container	COMPOSITION Production Server – Instance dedicated to eFactory	COMPOSITION Production Server – Instance dedicated to eFactory Accessible in TEST Envir	COMPOSITION Production Server – Instance dedicated to eFactory Accessible in PROD Envir
Essential Platform-Based Functionality	Online Bidding Process	Matchmaking engine for EFPF Online Bidding Process	Docker Container	COMPOSITION Production Server – Instance dedicated to eFactory	COMPOSITION Production Server – Instance dedicated to eFactory Accessible in TEST Envir	COMPOSITION Production Server – Instance dedicated to eFactory Accessible in PROD Envir
Essential Platform-Based Functionality	Business Opportunity Tool	Team Formation Tool				
EFPF Platform	Online Bidding Process	Agent-based Marketplace	Docker containers	LINKS Server	LINKS Server	LINKS Server
EFPF Platform	Anomaly Detection Service		Docker Container	https://efpf-analytics.icelab.cloud/	https://efpf-analytics.icelab.cloud/	https://efpf-analytics.icelab.cloud/
EFPF Platform	Customer Trend Analysis	Elanyo EFPF Behavioral Predictive Framework (BPF)				
EFPF Platform	Blockchain & Smart Contracting	Blockchain as a Service (BaaS)	Virtual Machine Nodes	N/A	CERTH Server	CERTH Server - Accessible through EFPF Portal in PROD Environment
EFPF Platform	Blockchain & Smart Contracting	Blockchain DAML Smart Contracts	Docker	N/A	N/A	Azure VM efpf-daml-main.northeurope.cloudapp.azure.com (only one instance dev/test/prod)
EFPF Platform	Blockchain & Smart Contracting	NIMBLE Blockchain		N/A	N/A	C2K PROD
EFPF Platform	Business & Network Intelligence	BNIDashboard	Docker Container	Host on ICE Servers - Accessible in DEV Environment	Hosted on ICE Servers - Accessible in TEST Environment	Hosted on ICE Servers - Accessible in PROD Environment
EFPF Platform	Business & Network Intelligence	ELK stack (Elasticsearch, Kibana) In development and testing phase	Docker Containers - Hosted on ICE Servers	Hosted on ICE Servers - Accessible in Dev Environment	Hosted in ICE Servers - Accessible in TEST Environment	Hosted at ICE Servers - Accessible in PROD Environment
EFPF Platform	Visual & Data Analytics Service	CERTH/COMPOSITION Data and Visual Analytics	Flask – python micro web framework		CERTH Server & COMPOSITION Server - instance for EFPF Accessible in TEST Envir	CERTH Server & COMPOSITION Server - instance for EFPF Accessible in PROD Envir
EFPF Platform	Deep Learning Toolkit		Docker Container	LINKS Server	LINKS Server	LINKS Server
EFPF Platform	Industreweb Visual Resource Monitoring Tool					
EFPF Platform	Industreweb Global					
EFPF Platform	Industreweb Visual Resource Monitoring Tool		Windows, .Net	Industreweb Server within production facility	Industreweb Server within production facility	Industreweb Server within production facility
EFPF Platform	Risk, Opportunity, Analysis and Monitoring (ROAM) Tool		Docker container	Locally	Locally	https://efpf.almende.com/
EFPF Platform	Software Development Kit (SDK)	Software Development Kit (SDK) Frontend Editor	Docker container in SDK Platform (Eclipse Che)	-	-	https://efpf.caixamagica.pt (Migration to EFF environment after projects ends is preferred)
EFPF Platform	Software Development Kit (SDK)			-	-	https://efpf.caixamagica.pt (Migration to EFF environment after projects ends is preferred)
EFPF Platform	System Security Modeler (SSM)			UoS-ITI infrastructure	UoS-ITI infrastructure	UoS-ITI infrastructure
EFPF Platform	Symphony Data Storage		Cloud	NXW Cloud Node	NXW Cloud Node	NXW Cloud Node
EFPF Platform	Symphony Resource Catalogue		Cloud	Cloud Node NXW	Cloud Node NXW	Cloud Node NXW
EFPF Platform	Symphony Event Reactor		Cloud	NXW Cloud Node	NXW Cloud Node	NXW Cloud Node
EFPF Platform	Workflow and Service Automation Platform (WASP)		Docker	WASP implementation is composed of a test instance and also a production instance. They both use separate MySQL databases. Both are deployed as Docker containers, using the Liferay parent image. ICE_Main	WASP implementation is composed of a test instance and also a production instance. They both use separate MySQL databases. Both are deployed as Docker containers, using the Liferay parent image. ICE_Main	WASP implementation is composed of a test instance and also a production instance. They both use separate MySQL databases. Both are deployed as Docker containers, using the Liferay parent image. ICE_Main
EFPF Platform	Analytics Integrator Platform (AIP)		Cloud	AWS Cloud	AWS Cloud	AWS Cloud

Annex E: Dependency Table

Context View Element	Name	Subcomponent	EFPF Dependencies	External Dependencies
Data Spine	Data Spine			
Data Spine	Integration Flow Engine (IFE)	Integration Flow Engine	EFS, API Security Gateway	Apache NiFi
Data Spine	Service Registry	Service Registry	EFS, API Security Gateway, Message Bus	LinkSmart Service Catalog
Data Spine	Message Bus	Message Bus	Pub/Sub Security Service	RabbitMQ Server
Data Spine	EFPF Security Portal (EFS)	EFPF Security Portal (EFS)		Keycloak, REST Services
Data Spine	API Security Gateway	API Security Gateway	EFS	Apache APISIX, Keycloak, REST Services, MQTT
Unified Functionality	EFPF Portal	EFPF Portal UI	- Data Spine - Smart Contracting - Governance & Trust - Portal Backend - Marketplace Backend - PubSubSecurity Backend Integrated tools: - Marketplace UI - Smart Factory Tools and Services - PubSubSecurity UI	N/A
Unified Functionality	EFPF Portal	EFPF Portal Backend	- Data Spine - Smart Contracting - Governance & Trust - Accountancy Service - NimbleIndexingService	MailJet (MailJet service)
Unified Functionality	Accountancy Service		· eFactory Portal: Login, registration, platform & tool visit events and related information are sent to the Accountancy Service when users perform these activities. · Base Marketplaces: Payment events realized on marketplaces of DIGICOR NIMBLE, and VF-OS projects and related information are sent to the Accountancy Service when users are redirected to the base marketplaces from eFactory Portal through clicking products and perform a successful purchase.	Elasticsearch, Logstash and Kibana
Unified Functionality	EFPF Marketplace	EFPF Marketplace UI	- Marketplace Backend - Accountancy Service - EFS - EFPF Portal	N/A
Unified Functionality	EFPF Marketplace	EFPF Marketplace Backend	- Data Spine - Accountancy Service - Governance & Trust - Service Registry - EFS	N/A
Unified Functionality	Federated Search	Federated Search Engine		Apache SOLR (Docker Container)
Unified Functionality	Federated Search	Base Platform Data Indexing Workflow	Data Spine / Nifi Workflow	
Unified Functionality	Federated Search	Federated Search Service	EFPF Federated Search Engine	
Unified Functionality	Federated Search	Federated Search Frontend Component	EFPF Federated Search Engine	
Unified Functionality	Matchmaking Service		· Data Spine / Nifi workflow for periodic data indexing from base-platforms · EFPF portal · User authorization framework to grant access to matchmaking functionalities	The federated search implementation has 4 main components. Backend components are deployed and maintained by SRFG in a dedicated server for eFactory project from SRFG (server). Frontend components are integrated into the eFactory portal maintained by ASC. · Federated search engine: Apache Solr instance providing search features · Base platform data indexing workflow: Nifi workflow to index base-platform data periodically · Federated search microservice: Search REST microservice deployed in the SRFG server · Federated search frontend component: Angular 8 component integrated into the eFactory portal
API Management	Pub/Sub Security Service	Front End	EFPF Portal EFS DS RabbitMQ Pub Sub Backend Pub Sub Database	
API Management	Pub/Sub Security Service	Backend	EFPF Portal EFS DS RabbitMQ	

			Pub Sub Frontend Pub Sub Database	
API Management	Pub/Sub Security Service	Database	EFPF Portal EFS DS RabbitMQ Pub Sub Backend Pub Sub Frontend	
API Management	Interface Contracts Management Tool		Message Bus, EFS, Service Registry	
API Management	Service Registration Tool		Message Bus, EFS, Service Registry	
API Management	Semantic Information Management (SIM) tool			
DevOps, Maintenance, & Support	Monitoring & Alerting Service			
DevOps, Maintenance, & Support	Monitoring & Alerting Service	Monitoring Server	EFS	Prometheus, Prometheus-Alertmanager, Grafana, Grafana-Loki, oauth2-proxy
DevOps, Maintenance, & Support	Monitoring & Alerting Service	Monitoring Data Collectors at Target Hosts		cAdvisor, vector
DevOps, Maintenance, & Support	Gitlab			
DevOps, Maintenance, & Support	EFPF Documentation Portal			
DevOps, Maintenance, & Support	EFPF Ticketing System	Tikki		
Essential Platform-Based Functionality	Product Catalogue Service		· API Security Gateway · Service Registry · EFS	N/A
Essential Platform-Based Functionality	Secure Data Store Solution		· Pub/Sub Security Service · Message Bus · EFS	MongoDB and InfluxDB. There is a variant docker image integrating these.
Essential Platform-Based Functionality	Industreweb Collect		Data Spine - message bus + PubSub Security component. This connector publishes data to the eFactory broker and will subscribe to the eFactory Service Registry via a REST interface.	
Essential Platform-Based Functionality	TSMatch Gateway		Data Spine and Message Bus - integration completed	
Essential Platform-Based Functionality	Symphony Hardware Abstraction Layer (HAL)			MQTT, HTTP
Essential Platform-Based Functionality	Online Bidding Process	Front-end Interfaces	Semantic Framework-CERTH and Agents-LINKS	It was based on initially designed open source UIs by CNET
Essential Platform-Based Functionality	Online Bidding Process			
Essential Platform-Based Functionality	Online Bidding Process	Matchmaking engine for EFPF Online Bidding Process	· Agents and Agent Marketplace from LINKS Foundation – For online bidding process that is coming from COMPOSITION and an updated version will be available to eFactory · Federated Search Mechanism of EFPF through Data Spine / NiFi EFPF Portal and SSO	
Essential Platform-Based Functionality	Business Opportunity Tool	Team Formation Tool	Federated Search, EFS SSO	
EFPF Platform	Online Bidding Process	Agent-based Marketplace	EFS (SSO)	Keycloak, Postgres
EFPF Platform	Anomaly Detection Service		EFS, Keycloak	Grafana, H2o
EFPF Platform	Customer Trend Analysis	Elanyo EFPF Behavioral Predictive Framework (BPF)	EFS	Azure SQL Server databricks
EFPF Platform	Blockchain & Smart Contracting	Blockchain as a Service (BaaS)	EFS (SSO)	The BaaS depends on the base blockchain network (Hyperledger Sawtooth nodes) and uses the EFPF Platform as a front-end to get users' registration information.
EFPF Platform	Blockchain & Smart Contracting	Blockchain DAML Smart Contracts	EFS (SSO)	DAML v1.17.0 Hyperledger Sawtooth
EFPF Platform	Blockchain & Smart	NIMBLE Blockchain	EFS	Hyperledger Fabric NIMBLE

	Contracting			
EFPF Platform	Business & Network Intelligence	BNIDashboard	Accountancy Service BNI ELK Stack	ELK Stack
EFPF Platform	Business & Network Intelligence	ELK stack (Elasticsearch, Kibana) In development and testing phase	BNIDashboard Accountancy Service	ELK Stack
EFPF Platform	Visual & Data Analytics Service	CERTH/COMPOSITION Data and Visual Analytics	· Vibration and Fill Level sensors in KLEEMANN and ELDIA premises. – these sensors are maintained by CERTH · Deep Learning Toolkit predictions from LINKS for Price Forecasting service · EFPF portal – the component should be available through the portal interface · EFPF Security Framework – the analytic services should be available to specific users with specific roles	
EFPF Platform	Deep Learning Toolkit		Message Bus, EFS	
EFPF Platform	Industreweb Visual Resource Monitoring Tool		Data Spine - message bus + PubSub Security Service	
EFPF Platform	Industreweb Global		Industreweb Collect	
EFPF Platform	Industreweb Visual Resource Monitoring Tool			
EFPF Platform	Risk, Opportunity, Analysis and Monitoring (ROAM) Tool		· Pub/sub Security Service · Data Spine / Message Bus & Keycloak · In the future: SDSS	
EFPF Platform	Software Development Kit (SDK)	Software Development Kit (SDK) Frontend Editor	SDK Workspace (https://sdk.efpf.caixamagica.pt/)	-
EFPF Platform	Software Development Kit (SDK)			
EFPF Platform	System Security Modeler (SSM)		EFS SSO	
EFPF Platform	Symphony Data Storage			
EFPF Platform	Symphony Resource Catalogue			
EFPF Platform	Symphony Event Reactor		The Event Reactor consumes and produces data using MQTT and HTTP protocols.	
EFPF Platform	Workflow and Service Automation Platform (WASP)		WASP uses the eFactory Security Framework (based on open-source Keycloak technology) to support Single-Sign-On functionality to allow eFactory users to access the platform and make use of all features. Separate (WASP only) user Registration and user authentication functionality is also provided through the native Liferay framework.	Liferay framework. The WASP platform is composed of a Liferay component, workflow execution engine (Camunda) Tomcat webserver and a MySQL database. The WASP platform, with all components, is currently deployed on the ICE server. All these components are deployed as Docker containers.
EFPF Platform	Analytics Integrator Platform (AIP)		-	-

European Factory Platform

www.efpf.org