SOFIE - Secure Open Federation for Internet Everywhere

779984

DELIVERABLE D3.4

Business Platforms, final release

Project title	SOFIE – Secure Open Federation for Internet Everywhere
Contract Number	H2020-IOT-2017-3 – 779984
Duration	1.1.2018 – 31.12.2020
Date of preparation	22.12.2020
Author(s)	Mikael Jaatinen (LMF), Antonio Antonino (LMF), Filippo Vimini (LMF), Santeri Paavolainen (LMF), Ahsan Manzoor (Rovio)
Responsible person	Mikael Jaatinen (LMF), mikael.jaatinen@ericsson.com
Target Dissemination Level	Public
Status of the Document	Completed
Version	1.0
Project web-site	https://www.sofie-iot.eu/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779984.

Table of Contents

1 Introduction 3

2 Release Information 4

2.1 Release identifier 4

2.2 Release scope 4

2.3 Purpose and target users 4

3 Release Contents 5

3.1 Components 5

3.2 CI/CD pipeline 7

3.3 CI and CD Infrastructure 8

3.4 CI architecture 8

3.4.1 Overview 8

3.4.2 Jenkins 10

3.4.3 Build pipelines 10

3.4.4 Build nodes 10

3.4.5 Partner access 10

3.4.6 Logging 10

3.5 CD architecture 10

3.5.1 Overview 10

3.5.2 Deployment nodes 13

3.5.3 Kubernetes 13

3.6 SMAUG 13

3.7 Other CI/CD environments 13

3.7.1 Rovio 13

4 Related Deployments 15

4.1 Testbed and emulation environment 15

4.2 Pilot deployments 15

5 Changes since previous release 16

6 Lessons learned and recommendations for future development 17

7 References 18

8 Appendix I: Onboarding Maturity Levels 20

1 Introduction

SOFIE is a three-year EU Horizon 2020 research and innovation project with the goal to enable diversified applications from various application areas to utilise heterogeneous IoT platforms and autonomous things across technological, organisational and administrative borders in an open and secure manner, making reuse of existing infrastructure and data easy.

IoT business platforms are created in SOFIE based on the federation framework defined in Federation Architecture & Framework work package. For this, an IoT framework repository, consisting of various components, adapters for well-known IoT platforms and security mechanisms is developed. These components can be used to create business platforms, including those for the four SOFIE real-world pilot use cases.

During 2018-2020, SOFIE has delivered three business platform main releases as defined by deliverable “D3.1 - Integration Plan” [Jaa2019]. Within every main release, new functionality was added through Continuous Integration (CI) and Continuous Deployments (CD), using the CI/CD environment developed in this work package.

The first Lab Prototype Release was made available in November 2018 as described by deliverable “D3.2 - Business Platform, Lab Prototype Release” [Jaa2018]. This release demonstrated the use of an initial version of the CI/CD environment, with tools and methodology that supported the integration of a limited interledger demo. At the time, a testbed and Emulation environment, as described by deliverable “D4.2 - Testbed and Emulation Environment Design and Setup” [Lag2019] was also made available to support integration testing. Agile methodology with monthly sprints was introduced to allow continuous functional growth between main releases.

The second Pilot Release was made available in September 2019 as described by deliverable “D3.3 - Business Platform, Pilot Release” [Jaa2019b]. This release builds further on top of “D3.1 - Integration Plan” [Jaa2019] and “D3.2 - Business Platform, Lab Prototype Release” [Jaa2018] by providing a fully developed CI environment, with tools and methodology for integrating WP2 SOFIE Framework components available in public SOFIE repositories and specified by “D2.5 - SOFIE Framework, 2nd version” [Kor2019] as well as the integration of WP5 pilot-specific components in private repositories. The scope of the pilots at the time is explained by deliverables “D5.1 - Baseline System and Measurements” [Oik2018] and “D5.2 - Initial Platform Validation” [Oik2019].

In April 2020 and September 2020, interim code releases have been published by SOFIE. Agile methodology with monthly sprints and quality controls have enabled a smooth integration of the results from both releases in the WP3 CI/CD pipelines for validation and pilot use.

The Final Release, discussed by this deliverable, was made available in December 2020. As the release name suggests, this is the final release in the SOFIE project and it includes all results that were produced during the project.

2 Release Information

2.1 Release identifier

Release name: **SOFIE Business Platform Final Release**

Release date: 17.12.2020

2.2 Release scope

The Business Platform Final Release is based on the final SOFIE Deliverables and is integrated in accordance with deliverable “D3.1 - Integration Plan” [Jaa2019].

The SOFIE final release consists of:

- Integrated WP2 Framework components with CI onboarding level (see Appendix I)
- Integrated WP5 Pilot components with CD onboarding level (see Appendix I)
- CI/CD process, environment and architecture
- Related deployments (SMAUG, Pilots, Testbed and emulation environment)
- Learnings and recommendations for future development

2.3 Purpose and target users

The purpose of the Business Platform Final release is to:

- Integrate WP2 framework components for evaluation in WP4 in accordance with “D2.7 - Federation Framework, final version” [Kor2020] and “D4.5 - Final Architecture, System and Evaluation Report” [Sir2020]
- Integrate WP2 framework components for validation in accordance with “D2.7 - Federation Framework, final version” [Kor2020] and “D5.4 - Final Validation & Replication Guidelines” [Oik2020b]
- Integrate WP2 framework components in public repositories and WP5 pilot-specific components in private repositories for WP5 pilot field trial usage in accordance with “D2.7 - Federation Framework, final version” [Kor2020], “D3.5 - Final Business Platform Integration Report” [Vim2020], “D5.1 – Baseline system and measurements” and “D5.3 - End-to-end Platform Validation” [Oik2018, Oik2020]
- Integrate WP2 framework components in public repositories for the SMAUG reference implementation in accordance with “D2.7 - Federation Framework, final version” [Kor2020] and “D3.5 - Final Business Platform Integration Report” [Vim2020]
- Provide quality control mechanisms for the WP2 Framework components that are available in public software repositories
- Deliver a functioning CI/CD environment in the Amazon Web Services (AWS) public cloud with support for multiple parallel CI/CD pipelines as well as needed instructions for use

3 Release Contents

3.1 Components

One of the reasons for providing CI-as-a-Service to the developers of different pilot software components and framework components is to increase the overall quality of the software components produced. By owning and managing the CI environment, LMF Ericsson is able to take advantage of its expertise in the field and offer a set of additional services that might not be available to the pilots in their testing environments, thus increasing the overall quality of the components code, e.g. by reducing the number of bugs that might affect the functionalities of the component, once made publicly available to its users.

In the same fashion, by hosting the entire CD runtime, LMF offers a Platform-as-a-Service environment by deploying all the frameworks and pilot components on the infrastructure it owns and manages.

WP3 has defined a systematic way of onboarding users on the CI/CD environment with seven onboarding levels, see Appendix I. For every level, a well-defined process has been defined and executed in collaboration between WP3 and the onboarded user repositories that can be either public or private.

All the components that have been released by the project, along with their current status in the CI automation, are described in Table 1 below. Each component is detailed with regard to the following fields:

- **Origin**: the SOFIE partner leading the development of the component.
- **Component Users**: the pilots that have planned to use the component by integrating it into their codebase.
- **Integration level**: the level of maturity of the component with regard to the CI pipeline (levels 0 to 3). Appendix I describes the meaning of each maturity level and the actions that must be taken by the component developers to take the component to each maturity level.
- **Public**: whether the source code of the component is publicly accessible or not.
- **Description**: the URL of the README file of the component source code repository for publicly available components.

The different pilots and projects also use the CD environment to perform overall integration testing across multiple components[1], as described in Table 2 below. Each pilot and project is described with the following information:

- **Origin**: the SOFIE partner leading the pilot or project.
- **Integration level**: the level of maturity of the pilot or project with regards to the CD pipeline (levels 4 to 7). See Appendix I for detailed description of the levels.
- **Reference**: location of the detailed description of the pilot or project.

The tables below reflect the components as of the release date of this deliverable. A link to the README file is provided for public SOFIE project source code repositories. The users of the components are Food Supply Chain (FSC), Decentralized Energy Data Exchange (DEDE), Decentralized Energy Flexibility Marketplace (DEFM) and Context-Aware Mobile Gaming Pilot (CAMG). The highest integration level for individual components is level 3.

[1] Details on which and how the components are integrated is described in deliverable D3.5.

Table 1. The SOFIE project software components.

Component	Origin	Component Users	Integration level	Public	Description
Discovery and Provisioning	ROVIO	CAMG, SMAUG	1[2]	yes	README
DSO dashboard	ENG	DEFM	3	no	-
DSO backend	ENG	DEFM	3	no	-
Identity, Authentication and Authorization	AUEB	DEDE, FSC, SMAUG	3	yes	README
Interledger Asset Transfer	AALTO	DEDE, DEMF, FSC, CAMG, SMAUG	3	yes	README
Interledger demo	AALTO	-	3	no	-
Offer Marketplace	AALTO	DEFM, CAMG, SMAUG	3	yes	README
Privacy and Data Sovereignty	AUEB	DEDE, FSC, SMAUG	3	yes	README
Semantic Representation	AALTO	DEFM, FSC, CAMG, SMAUG	3	yes	README
SOFIE Adapter Application	GT	DEDE	3	no	-
FSC consortium smart contracts	SYN	FSC	3	yes	README
FSC adapter smart contract	SYN	FSC	3	yes	README
FSC public smart contract	SYN	FSC	3	yes	README
FSC transportation federation adapter	SYN	FSC	3	yes	README
FSC supervisor	SYN	FSC	3	no	-

Table 2: CD integration levels of different pilots and prototypes for the CI/CD environment.

Project	Abbreviation	Origin	Integration level	Reference
Food Supply Chain	FSC	SYN	5	D5.3 Section 3
Decentralized Energy Data Exchange	DEDE	GT	5	D5.3 Section 4
Decentralized Energy Flexibility Marketplace	DEFM	ENG	5	D5.3 Section 5
Context-Aware Mobile Gaming	CAMG	ROVIO	3[3]	D5.3 Section 6

[2] Discovery and provisioning has been integrated in Rovio's internal CI/CD environment.
[3] The Context-Aware Mobile Gaming pilot has been integrated in Rovio's internal CI/CD environment.

3.2 CI/CD pipeline

On-boarding a component onto the CI/CD pipeline requires cooperation between LMF and the developer of the component. Clear instructions have been provided by LMF about how to prepare a component to be on-boarded and how to properly write documentation for it in the integration documentation template file.

At any given time, a component can be in one of the seven maturity levels shown in Appendix I. Levels **0-3** are relative to the CI maturity, while levels **4-7** indicate the maturity of the component with regard to the CD environment. The target level of any component depends on the requirements of the creator of the component, and may vary from component to another. In general, Level 3 is considered a desirable goal for all components, and level 5 for any component needing integration testing. Levels 6 and 7 are considered to be feasible, but only if desirable from a component development viewpoint.

Any component must meet the full CI maturity level (level 3) before proceeding to CD integration (level 4 onwards).

Specifically for the CI maturity levels, component developers are responsible for:

1. Giving the CI agent read-only access to the component repository either via SSH-based (preferred) or HTTPS-based authentication.
2. Configuring webhooks to trigger new builds in the CI environment whenever new commits are pushed on a branch specified in the integration documentation. *The completion of steps 1 and 2 promotes the component to maturity level 1.*
3. Defining the component build process in a way that produces Docker images as final artifacts that are parametrized with regard to potential hard-coded IP addresses and port numbers. The tagging details of the Docker images must be specified in the integration documentation.
4. For those artifacts, writing unit tests that are collected in JUnit format in a path that must be specified in the integration documentation. *Fulfilment of steps 3 and 4 promote the component to maturity level 2.* Nevertheless, in case of failure of either the build process or the unit tests, the component cannot be taken to step 3, since the artifacts will not be pushed onto the remote artifact repository.

Taking a component from level 2 to level 3 is entirely the responsibility of LMF, which will create the required ECRs, or Elastic Container Registries, on AWS. The registries have the proper permissions so that only the LMF staff is able to access them in read-write mode and the component developers in read-only mode. Once the required ECRs are created, the CI build process pushes the artifacts there so that the CD pipeline can be triggered and can pull the new artifacts.

Moving from level 3 to level 4 requires co-operation between the component developer and LMF, as knowledge of the proper deployment and configuration has to be encoded into the CD pipeline. Furthermore, moving from level 4 to level 5 requires definition of integration tests that are suitable for the chosen deployment model and validation goals. This process is iterative and requires close collaboration from multiple parties.

Level 4 and 5 deployments are temporary and used only for automated integration testing. A level 6 deployment is a persistent deployment that is automatically created when a level 5 integration test passes all its tests successfully. Furthermore, level 7 deployment is similar to level 6 deployment, but triggered manually (so-called manual promotion deployment). The necessity of level 6 & 7 deployments has to be evaluated on a per-component basis, requiring

consideration of security and safety aspects, visibility of the deployment to external parties etc. It is expected that above level 5 deployments are used only when a sufficient benefit can be demonstrated above the increased complexity and other considerations (such as security, GDPR etc.). During the SOFIE project, no clear need for persistent deployments hosted in WP3 CI/CD have been identified - it has been more practical to host the pilot deployments fully (including physical components and commercial/SP components) in the pilot specific environment. On the other hand, persistent deployments hosted in the WP3 CD environment would come at a substantially increased cost. For these reasons, the onboarding step to CD level 6 or 7 will remain as a potential future item to address.

3.3 CI and CD Infrastructure

The core capabilities the environment provides are:

1. Perform **continuous integration** tasks when triggered by changes in watched repositories of SOFIE WP2 and WP5 material. Continuous integration supports multiple repositories and in the build pipelines.
2. Perform **continuous delivery** tasks, including automated integration tests, and deployment to staging and production environments. Continuous delivery supports multiple pipelines, e.g. for each pilot separately.
3. **Support per-pilot integration** needs by being able to provide servers for pilots to configure any necessary gateways etc. used during integration testing, staging and production deployments.
4. Enough flexibility in CI/CD and environment to make it reasonably feasible to automate various evaluation tasks, and/or manual validation and evaluation tasks.

The underlying infrastructure where the integration and evaluation environment is being deployed as part of WP3 is the **Amazon Web Services (AWS)** cloud environment. A German AWS datacenter is used for production deployment of the CI/CD environment.

Figure 1 shows the interactions between LMF, the pilot components, and the CI/CD pipeline. Changes to the integrated repository trigger a CI task that builds the repository and performs unit tests. On successful completion of the CI task, a CD task is triggered, which will perform integration tests. If integration tests are successful, a promotion to a persistent deployment may be performed if required.

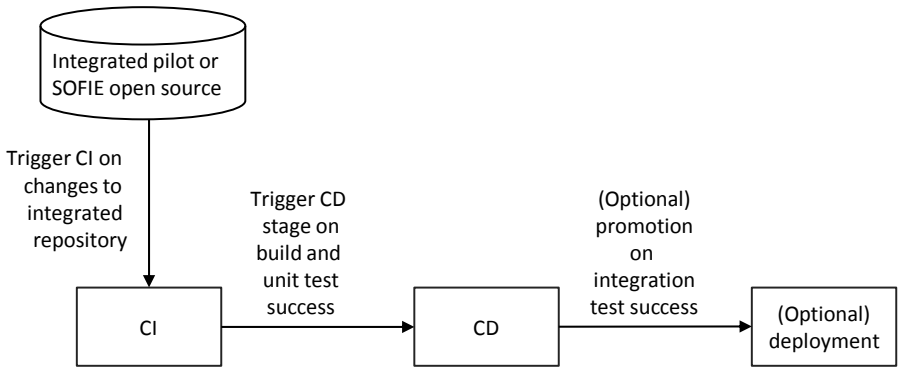

Figure 1. Overview of how WP2 and WP5 components are integrated and deployed using the CI/CD pipeline built by LMF.

3.4 CI architecture

3.4.1 Overview

Figure 2 shows an overview of the infrastructure environment components and their major relationships. The “infrastructure as code” approach for the environment setup allows easy and

repeatable deployments of the complex multi-node integration environment. The Terraform tool is suitable for this purpose as it supports incremental deployments and parameterization of the deployments, including integration of externally managed resources into the deployment templates. In this model, the integration environment itself is described in the Terraform template language and stored in a version-controlled source code repository.

The CI/CD requirements in SOFIE have not justified the additional investment to design and deploy a redundant CI environment, although it has been highly reliable in practice. The underlying assumption is that while CI and CD processes are important, a failure due to the loss of a virtual machine, database etc. is an unlikely event, and, in the worst case, manual recovery through rebuilding of failed components will last only a week at most.

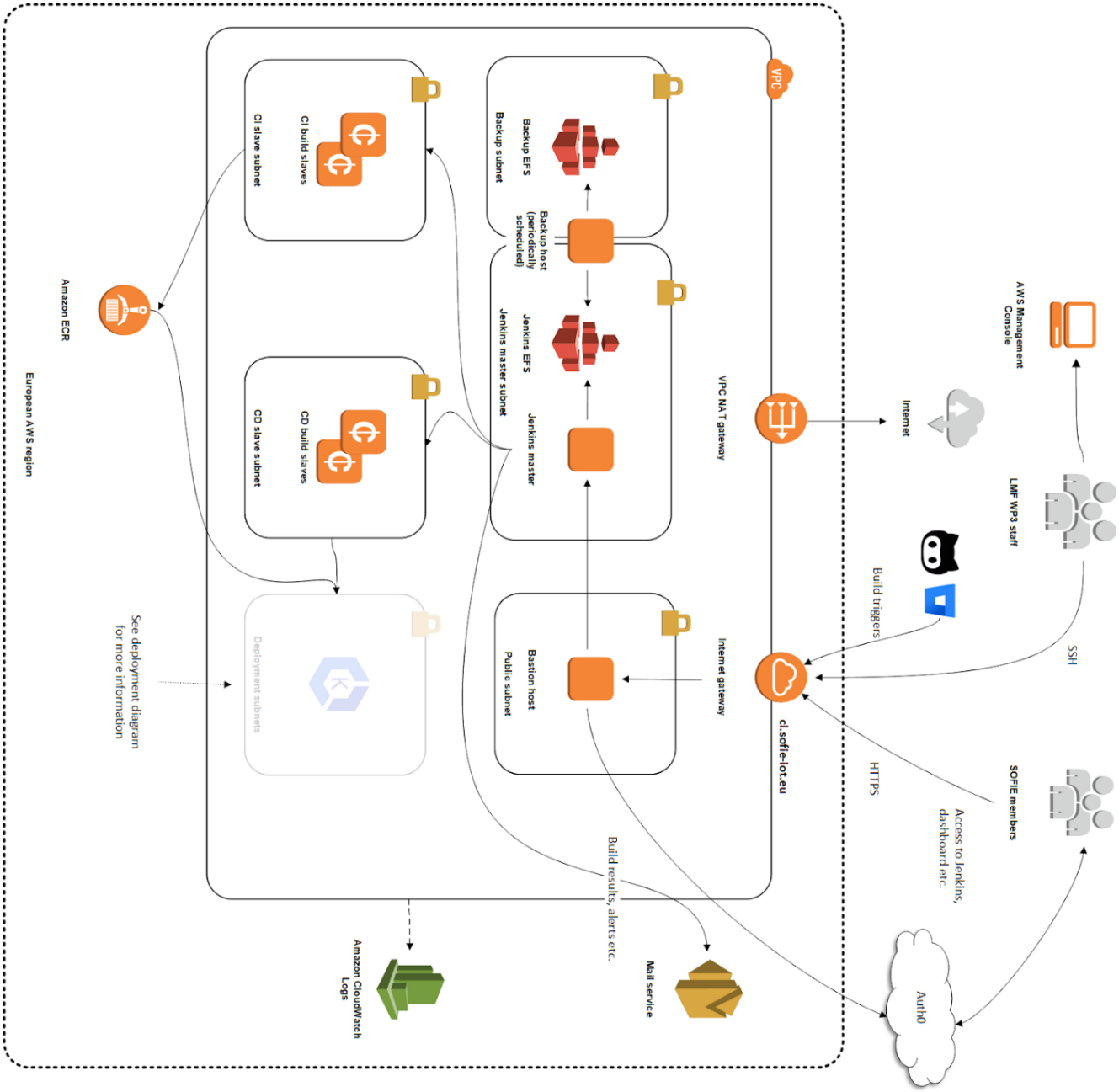

Figure 2. Overview of the CI architecture. The environment is run in an isolated network accessible only via the bastion host for authenticated Jenkins users and administrative staff. The deployment includes automated scheduled backups, centralized logging and monitoring, and dynamically scaled build fleet for CI and CD tasks.

3.4.2 Jenkins

The heart of the CI/CD system is a virtual machine running the Jenkins software.

Jenkins is currently the most popular open source CI/CD tool. The tool is used to build and test software continuously, meaning that developers can continuously integrate changes into a build repository. The same Jenkins system is used for both CI and CD functionality, although the per-component and per-project build pipeline definitions differ. Most of the Jenkins configuration is part of the persistent networked file system to ensure loss of the Jenkins virtual machine has no effect on information on pipeline definitions, job logs, access control configuration etc. The Jenkins instance is also configured to send email and Slack notifications upon successfully or failed CI/CD jobs.

The Jenkins virtual machine is configured to access external network resources via a NAT gateway which is fixed to a persistent external IP address, facilitating, if necessary, firewalling on the fixed external IP address.

3.4.3 Build pipelines

The detailed build instructions for each component and project are managed in a separate version-controlled repository. These per-component and per-project pipeline definitions specify where the build job is to fetch source code (such as SOFIE project public repository, or a partner's private source code repository), what build environment to use, and what commands to issue to build the code and run unit tests. The build environment is specified as a docker container. The build may result in docker images being built, which are then automatically tagged with the build identifier and sent to the container registry.

3.4.4 Build nodes

The individual build nodes are, for cost-efficiency purposes, provisioned from a dynamic pricing pool ("spot fleet"). Although it is possible to lose a build node due to pricing fluctuations, a replacement is automatically launched and configured for use. The build nodes are remotely controlled by the Jenkins system, which will send commands for them to run the per-component and per-project pipelines as either manually or automatically triggered.

3.4.5 Partner access

The Jenkins system is exposed to both WP3 administrators and partners via a web interface. A level of isolation is provided by running a proxy on a separate bastion host, isolating the Jenkins system itself from direct access from the Internet.

Staff and partners are authenticated using an external authentication service provider (auth0), which allows partners to use flexible authentication options, providing username/password and Google platform authentication options. Access control is mediated within the Jenkins system based on profile assignments, allowing, if necessary, different partners only the visibility to their own build jobs.

3.4.6 Logging

The CI/CD environment uses a centralized logging and monitoring system. The integrated AWS monitoring system provides basic monitoring of different resources, including virtual machines, storage space, network usage etc. The different virtual machines are configured to additionally ship logs to the AWS monitoring system's log storage, allowing logs to be stored separately from the (potentially non-persistent) virtual machines as well as viewed and searched efficiently.

3.5 CD architecture

3.5.1 Overview

Figure 3 shows the major components of the deployment environment. The deployment environment is separated from other infrastructure services (different subnets). The deployment process is driven by the Jenkins CD node fleet and uses general Kubernetes deployment

functions provided by AWS Elastic Kubernetes Service (EKS). In the CI/CD environment access to the EKS subnets is limited to the CD nodes only.

Deployed services are run in Docker containers orchestrated by AWS EKS. The EKS consists of a separate master cluster (EKS control plane) and a cluster of worker nodes. The master cluster orchestrates the cluster and provides the Kubernetes API endpoint while the worker nodes run the containers. The EKS control plane is managed by AWS to provide automatic healing creating a fully production ready platform. The worker nodes are an EC2 auto scaling group, which enables automated scaling for the cluster.

In addition to Docker containers, deployments, if necessary, are supplanted by other temporary or persistent services (database, cache, pilot backend gateways, etc.). Ethereum and Indy testbed nodes are deployed in AWS.

Figure 3 illustrates the CD environment for testing, staging and production deployments. The testing and persistent deployments are done on a Kubernetes cluster on separate namespaces for logical and network isolation. The deployment and integration testing process is controlled from the Jenkins master indirectly via the CD builder node. Services maintained outside the Kubernetes environment (such as Ethereum and Indy testbed nodes) are accessible to deployments.

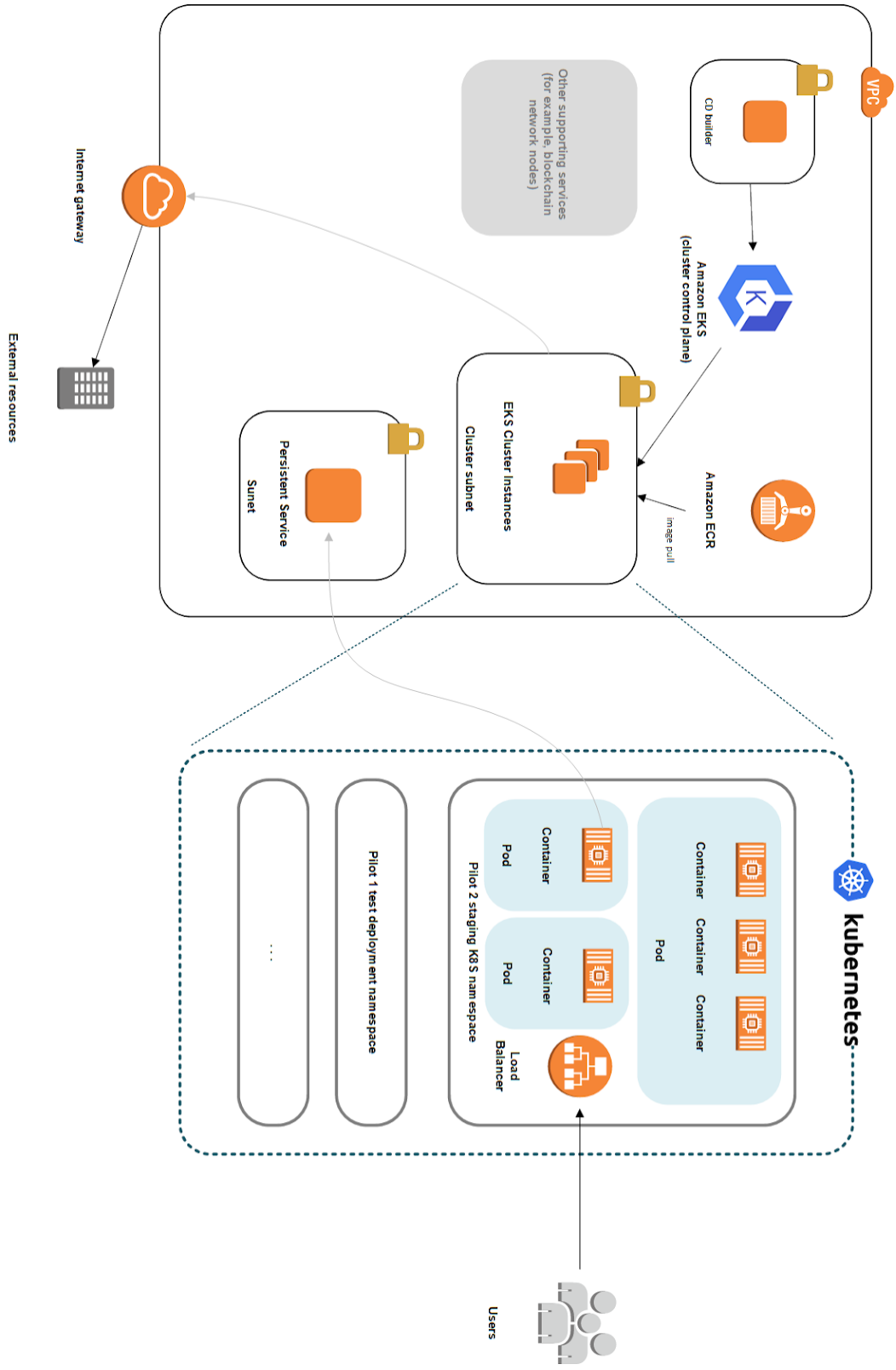

Figure 3. CD environment for testing, staging and production deployments.

Note that the CD environment is controlled from the same Jenkins system that controls also the CI functionality. For this reason, we only describe the main unique characteristics of the CD environment below.

3.5.2 Deployment nodes

The nodes used to run deployment and integration test tasks are separated from build tasks. This is to provide a layer of isolation. The build artifacts are always mediated between CI and CD via the container registry, a good practice to prevent accidental uses of resources not formally defined as build artifacts. Additionally, the build nodes do not have access to the Kubernetes cluster management interface; this is restricted to deployment nodes only.

The Jenkins system runs CD pipeline jobs on deployment nodes specifically. These run further commands to properly deploy the system under test. The details of the deployment vary from project to project, but in general include creation of an unique Kubernetes namespace (deployment isolation), configuring and deploying different containers and services related to the project either in a single step or sequentially in multiple steps, identifying service endpoints and passing them to integration tests, running of the integration tests, and finally collecting test results.

3.5.3 Kubernetes

The deployments from integration level 4 to level 7 occur in a containerized and isolated Kubernetes environment. Access to the deployed environment is normally limited to the deployment nodes alone, but for level 6 to 7 deployments can be configured to allow also public (Internet) access. The CD environment uses the AWS-provided Kubernetes control plane, and runs multiple Kubernetes cluster nodes in a redundant and cost-effective dynamic pricing basis. The deployments pull containers either from the CI/CD container registry, or from public registries if a generic container (such as a Postgres database) is used within the integration deployment.

3.6 SMAUG

The **Secure Marketplace for Access to Ubiquitous Goods**, or SMAUG, is a decentralised and open marketplace where smart locker owners can put their smart lockers for rent, and potential smart locker renters can place bids, for those smart lockers to get the authorisation to use them. Smart locker owners publish smart lockers on the marketplace by creating a request, i.e., a request for offers. The bids that smart locker renters place for those requests are called offers. SMAUG places itself as a reference implementation, to show how all the different SOFIE components can be used together to develop a system that benefits from all the properties that the SOFIE framework provides. Furthermore, SMAUG is being developed by LMF as a WP3 leader, and this means that an important target for SMAUG is to provide high-quality feedback to SOFIE component developers about the set of features the components offer, their level of reusability and extensibility, and their quality relating to how easily they can be integrated into systems other than the four pilots under development. This is achieved by following a “learn by doing” approach and testing the components via direct integration into a system developed from scratch during the last year of the project. For a detailed description of the SMAUG use case, as well as its architecture and how the SOFIE components have been integrated, see deliverable “D3.5 - Final Business Platform Integration Report” [Vim2020].

3.7 Other CI/CD environments

3.7.1 Rovio

Rovio uses its internal pipeline for continuous integration and development for the SOFIE context-aware mobile gaming pilot. The pilot is divided into two parts: the first includes mobile clients developed in Unity game engine and the second includes the backend and ledger developed and deployed on AWS. After the code is committed into a version control system, i.e. GitHub, by the developer, it goes through the build phase where it is compiled. Once the compilation is completed, the code goes through various kinds of tests written by the developer to check the functionality of the code. After testing, it goes to the deployment phase, where the

code is deployed in the staging environment and is published for usage. As for the mobile game clients, after the clients have been remotely built by the pipeline, the installation packages file becomes available for the users through Rovio internal platform. The backend and the ledger are deployed on AWS. As the Fabric ledger is managed by AWS, it cannot be updated automatically. New code is manually deployed every time changes occur. SOFIE mobile gaming pilot stands at Level 3 of the SOFIE corresponding CI/CD onboarding maturity level.

4 Related Deployments

4.1 Testbed and emulation environment

A SOFIE testbed and emulation environment is available as described in deliverable “D4.2 - Testbed and Emulation Environment Design and Setup” [Lag2019].

The testbed spans multiple project partners (AALTO, AUEB, LMF Ericsson) and allows testing various distributed ledger technologies and their interaction with IoT devices on a wider scale. Testbed elements include private Ethereum, Hyperledger Fabric, Hyperledger Indy and access to Guardtime KSI blockchain.

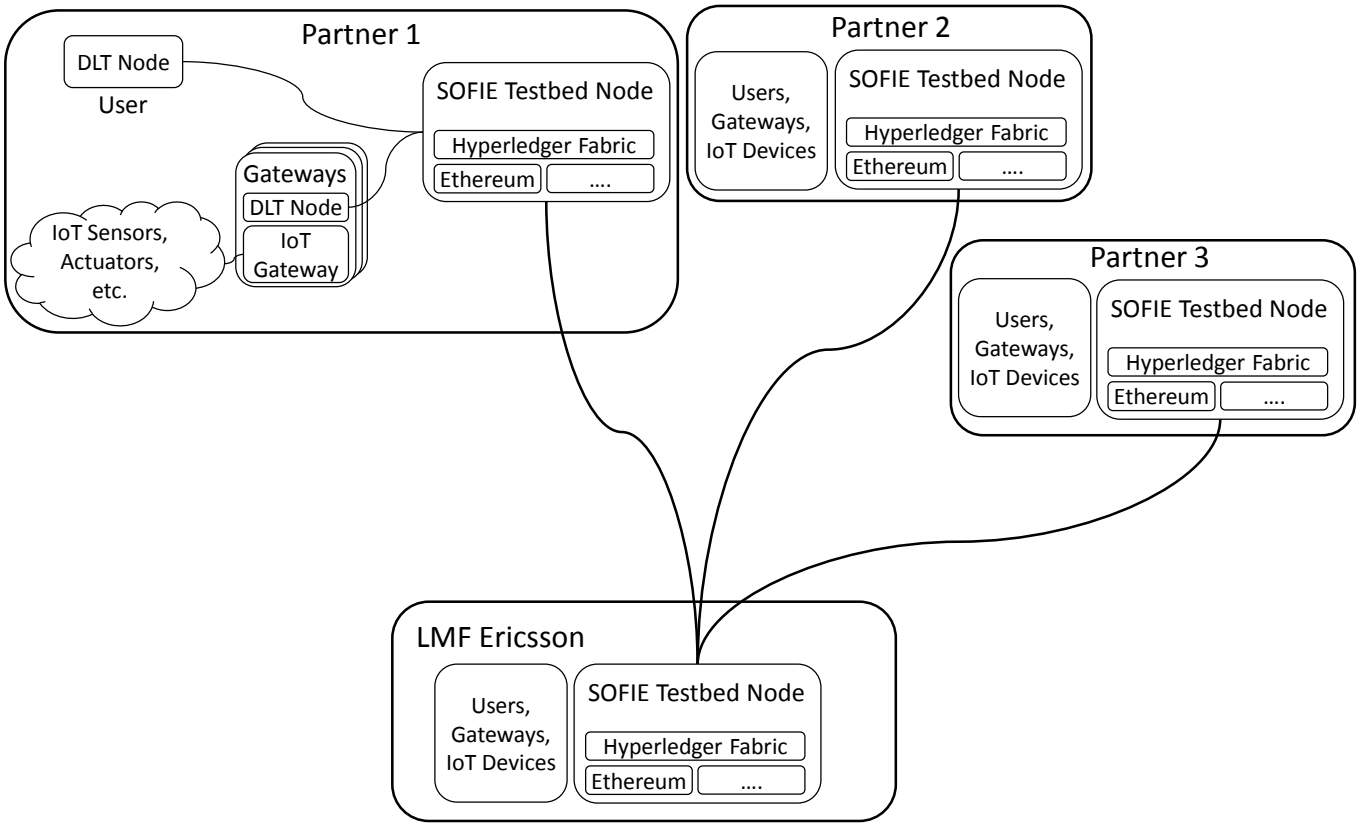

Figure 4. Overview of the SOFIE testbed setup.

The SOFIE emulation environment emulates certain aspects of SOFIE pilots and related, more general, use-cases, thus allowing realistic testing of various solutions without deploying them yet in pilot environments.

The testbed and emulation environment is used as an additional means to validate the correct behavior of integrated business platforms, with a close feedback loop between WP3 and WP4. The local testbed at LMF Ericsson is hosted in the same Amazon Web Services account as the CI/CD environment, making it easy to use testbed components as part of CI and CD activities. The Indy testbed has been also used by other projects than SOFIE.

4.2 Pilot deployments

Before field deployment, a proof of concept prototype has been implemented and demonstrated in the lab environment for every SOFIE pilot, as described by deliverable “D5.2 - Initial Platform Validation” [Oik2019]. The validation of pilots’ use cases occur primarily within the context of the field pilots. The CI/CD environment is used for programmatic validation of individual components and technical functionality of pilots. The extent of the CI/CD validation is decided by pilots as documented in “D5.4 - Final Validation & Replication Guidelines” [Oik2020b].

5 Changes since previous release

The main changes since previous release:

- Feature enrichment and code quality improvements for WP2 framework components
- Enriched CI/CD environment with demonstrated onboarding of components up to CD level 5 as described in Appendix I and readiness to onboard up to CD level 7
- Onboarding of WP2 framework components as described in 3.1 Components, Table 1
- Onboarding of WP5 pilots and SMAUG as described in 3.1 Components, Table 2

6 Lessons learned and recommendations for future development

In general, running a CI/CD environment in a public cloud was in many ways easier than running it in a corporate datacenter. The entire environment is, by default, isolated from other environments, and since it is under the account owner's control, any changes such as network topology modifications are faster to implement. In contrast, it must be noted that the skills and knowledge required for properly setting up a fully configurable infrastructure environment may not be readily available for all projects.

It must be also noted, that for a full open source project, it might be easier to use readily-available CI/CD commercial solutions. For SOFIE, it was expected (and has actually occurred) that some of the integrated software would not be under open source license, and for security concerns a CI/CD environment fully under the project member control was viewed as more appropriate. If this is not, however, the case for other projects, we would recommend to take a thorough look at hosted CI/CD solutions.

The decision to use the infrastructure-as-a-code approach (using the Terraform tool), while causing an initial setup effort cost, turned out to be hugely beneficial in the end. Changes to the infrastructure could be peer-reviewed before being put into production in otherwise identical per-administrator environments. Additionally, there was no need to set up a separate static development or staging environment, as each person on the team could on-demand deploy a copy of the CI/CD environment, fully isolated from the production environment, as needed, and tear it down once finished with developing changes and testing.

The operating costs of the environment varied over time, as we added further functionality and capabilities to the system, as well as due to increased artifact and log storage as more CI/CD jobs were added and run. On the other hand, we did expand over time cost-saving methods such as the increased use of dynamically priced virtual machines (spot instances). The final per-month cost settled to around 500€, substantially less than originally budgeted. The cost includes the CI/CD environment with 6 virtual machines, persistent network disk and storage, persistent external IP address allocation, private DNS, Kubernetes setup, load balancers and a separate testbed setup.

7 References

[Jaa2018] M. Jaatinen et al.,"SOFIE Deliverable D3.2 - Business Platform, Lab Prototype Release", November 2018. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D3.2-Business_Platform_Lab_Prototype_Release.pdf

[Jaa2019] M. Jaatinen et al., "SOFIE Deliverable D3.1 - Integration Plan", April 2019. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D3.1-Integration_Plan.pdf

[Jaa2019b] M. Jaatinen et at., "SOFIE Deliverable D3.3 - Business Platform, Pilot Release", September 2019. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D3.3-Business_Platforms_Pilot_Release-v1.00.pdf

[Lag2019] D. Lagutin et al., "SOFIE Deliverable D4.2 - Testbed and Emulation Environment Design and Setup", February 2019. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D4.2-Testbed_and_Emulation_Environment_Design_and_Setup-v1.00.pdf

[Kor2019] Y. Kortesniemi et al., "SOFIE Deliverable D2.5, Federation Framework, 2nd Version", August 2019. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D2.5-Federation_Framework%2C_2nd_version.pdf

[Kor2020] Y. Kortesniemi et al., "SOFIE Deliverable D2.7, Federation Framework, Final Version", November 2020. Available at: https://www.sofie-iot.eu/results/project-deliverables

[Oik2018] I. Oikonomidis et al., "SOFIE Deliverable D5.1 - Baseline System and Measurements", June 2018. Available at: http://media.voog.com/0000/0042/0957/files/SOFIE_D5.1-Baseline_System_and_Measurements.pdf

[Oik2019] I. Oikonomidis et al., "SOFIE Deliverable D5.2 - Initial Platform Validation", July 2019. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D5.2-Initial_Platform_Validation.pdf

[Oik2020] I. Oikonomidis et al., "SOFIE Deliverable D5.3 - End-to-end Platform Validation", July 2020. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D5.3_End-to-end_Platform_Validation_v1.00-2.pdf

[Oik2020b] I. Oikonomidis et al., "SOFIE Deliverable D5.4 - Final Validation & Replication Guidelines", December 2020. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D5.4-Final_Validation_Replication_Guidelines.pdf

[Sir2018] V.A. Siris et al., "SOFIE Deliverable D4.1 - Validation and Evaluation Plan", October 2018. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D4.1-Validation_and_Evaluation_Plan-v1.00.pdf

[Sir2020] V.A. Siris et al., "SOFIE Deliverable D4.5 - Final Architecture, System, and Pilots Evaluation Report". December 2020. Available at: https://www.sofie-iot.eu/results/project-deliverables

[Vim2020] F. Vimini et al.,“SOFIE Deliverable D3.5 - Final Business Platform Integration Report”, December 2020. Available at: https://www.sofie-iot.eu/results/project-deliverables

8 Appendix I: Onboarding Maturity Levels

#	Level criteria	Requirements for component developers
Level 0	No integration to CI/CD	-
Level 1	CI is triggered on new commits, pull repository (fetch source code)	1. Give *read-only* access to the repository to Jenkins (possibly SSH-based) 2. Configure a webhook for push events to the component-specific URL
Level 2	CI Build and unit tests pass, unit test results are collected correctly	1. Build produces properly parametrized and properly documented Docker images 2. At least one unit test is implemented and must complete with no errors. Test results are collected in JUnit format.
Level 3	CI Build artifacts are pushed to artifact storage (ECR, artifactory, others) and can be used by developers (e.g. pulled locally)	1. Identification of Docker images to be pushed (usually accomplished in Level 2 already)
Level 4	CD test environment, that uses the artifacts, can be deployed, and does not enter a crash-restart loop	1. Definition of integration deployment schema (with WP3) 2. Definition of proper configuration for the deployment to meet integration testing needs (with WP3) 3. Responses and fixed related to problems identified during deployment
Level 5	CD Some integration tests exist, and they pass on the CD test deployment	1. Definition of an *initial* automated integration test (WP3 will implement this test, verifying that the deployment is testable) 2. Development of further automated integration tests (WP3 assists and validates tests)
Level 6 and 7 deployment levels are specified, but expected to be used only if there is a good cost-benefit rationale for using them, as deploying any public-facing service requires in addition to continuously increased maintenance and monitoring needs, also an initial security review to ensure a minimally secure deployment.		
Level 6	CD Staging deployment can be deployed after successful integration test build (e.g. semi-persistent deployment)	1. Providing rationale for the necessity of Level 6 deployment including a cost-benefit analysis for increased hosting costs 2. Identifying any additional deployment resources needed 3. Identifying configuration differences between Level 6 deployment and integration testing deployment
Level 7	CD Promotion process and deployment for "production" deployment	1. Providing rationale for the necessity of Level 7 deployment including a cost-benefit analysis for increased hosting

		costs 2. Identifying any persistence requirements 3. Identifying additional deployment and configuration change necessary 4. Implementing any migration functionality in the application, if needed