Deliverable D2.6

Initial DECIDE DevOps Framework Integration

Editor(s):	Antonio Fernández Llamas Javier Gavilanes Ruano
Responsible Partner:	Experis IT
Status-Version:	Final – v1.0
Date:	28/02/2018
Distribution level (CO, PU):	CO

Project Number:	GA 726755
Project Title:	DECIDE

Title of Deliverable:	D2.6 Initial DECIDE DevOps Framework Integration
Due Date of Delivery to the EC:	28/02/2018

Workpackage responsible for the Deliverable:	WP2 – DECIDE requirements and DECIDE solution integration
Editor(s):	Experis IT
Contributor(s):	Experis IT
Reviewer(s):	Leire Orue-Echevarria (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP4, WP5, WP6

Abstract:	This deliverable will provide the integrated DECIDE DevOps Framework. The initial version will be an initial prototype with the core functionalities implemented.
Keyword List:	DevOps framework, integration, multi-cloud, microservice.
Licensing information:	This component is offered under the MIT license. The document itself is delivered as a description for the European Commission about the released software, so it is not public.
Disclaimer	This deliverable reflects only the author's views and views and the Commission is not responsible for any use that may be made of the information contained therein

Document Description

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
V0.1	12/02/2018	First draft version	Experis IT
V0.2	15/02/2018	Second draft	Experis IT
V0.3	23/02/2018	Third draft	Experis IT
V1.0	27/02/2018	Ready for submission	Leire Orue-Echevarria (TECNALIA)

Table of Contents

List of Figures

List of Tables

Terms and abbreviations

API	Application Programming Interface
EC	European Commission
HTML	Hypertext Mark-up Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KR	Key Result
MCSLA	Multi-cloud Service Level Agreement
MIT	Massachusetts Institute of Technology
MTBF	Mean Time Between Failures
MVC	Model-view-controller
NFP	Non-functional Properties
NFR	Non-functional Requirement
RAM	Random Access Memory
REST	Representational State Transfer
UI	User Interface
URL	Uniform Resource Locator
WP	Work Package

Executive Summary

This document contains the technical description of the DevOps Framework. The main purpose of the DevOps Framework inside DECIDE project is to offer a web-based platform for the users (DevOps team members in DECIDE), along with several other components distributed as independent modules. The DevOps Framework will integrate all these modules, offering multiple workflows linking tools with each other. These workflows enable to carry out tasks which involve the management of the areas related to the development and operation, trying to explore and track the current state of a DECIDE application by sending or receiving information from the rest of the DECIDE independent tool components. The target users of the DECIDE DevOps framework is both developers and operators of multi-cloud native applications.

In this document we detail the initial version of the DevOps framework, explaining its general architecture and the enabling technologies used for the initial implementation. Moreover, we present how the tools communicate with each other, and show the integration between some of these modules. Finally, we will expose the deployment guidelines, so anyone can build the platform system following the instructions provided.

Due to the high number of DECIDE components that must be integrated, in this deliverable we will focus on explaining how we have instantiated each tool using independent microservices, as well as the benefits obtained from the use of this technology based on the deployment of isolated containers, analysing topics such as scalability and performance of the general implementation.

Future versions of this document will show the evolution of the DevOps framework, improving the architecture and integrating new tools into the system that currently still are in a very infant status, and at this stage are not fully ready to be used in our current scenario.

1 Introduction

1.1 About this deliverable

This deliverable explains the architecture of the initial DECIDE DevOps framework prototype, detailing on one hand. how it has been designed and on the other hand, the enabling technologies used for the implementation. The content included in this document corresponds to the first version of the platform, due in month 15.

Furthermore, this document will present an initial approach for the deployment of the DevOps framework, that will be useful for other project team members in further releases of DECIDE project.

1.2 Document structure

The document is structured in four (4) main sections:

- At first, we will introduce the DevOps framework context with a brief introduction to the project, explaining the objectives and structure of the document.
- After explaining the main requirements of the DevOps platform, we will go on explaining the implementation itself, starting with a functional description of the solution, followed by a technical description of the general architecture as well as how each module behaves itself, enumerating several technical constraints of the project.
- Afterwards, we will present the installation guidelines and its deployment, so anyone can use the developed framework and test it.
- To finalize the document, we will expose the conclusions of the deliverable, and some steps that will be accomplished in future versions of the project.

2 Implementation

2.1 Functional description

The DECIDE DevOps Framework is the platform from which the different Key Results will be accessed. Its main purpose is to offer an intuitive interface to the user where they can set up a specific multi-cloud native application and consume any of the other tools integrated in the system. The framework will provide an entry point to DECIDE and will handle the interconnection between all the elements involved, providing a global overview about the state of the application to the end user.

Functionalities:

The main functionalities of the DECIDE DevOps Framework can be summarized as:

1. *Entry point*. The framework must provide centralized access to the different tools and KRs. It will also provide the necessary facilities for user and application management.
2. *KR integration*. The framework must transparently unify the graphical interfaces of the Key Results. Besides, it will allow for the creation and edition of the Application Description, a file that contains the parameters to configure the tools and serves as an integration point for these tools. This file has been defined in deliverable D2.1 [1].
3. *Workflow orchestration*. The DevOps framework will be able to launch the different tools and KRs and make sure that the DECIDE workflow is followed as intended.
4. *Application configuration*. The DevOps framework will allow users to introduce the application information that is needed for the tools to function properly.

DECIDE DevOps framework will follow an incremental strategy, according to which different prototypes of the framework will be released in months 15, 27 and 33. The current M15 prototype has the following coverage of the expected functionalities:

1. *Entry point.* **Partially covered**. This prototype provides a platform with centralized access to some of the DECIDE tools. User and application management will be addressed in a future release.
2. *KR integration*. **Partially covered**. The prototype gives access to some of the DECIDE KRs (ARCHITECT and OPTIMUS). However, the level of integration will be tighter in the next releases, and there will be integration with all tools.
3. *Workflow orchestration*. **Limited coverage**. The DevOps framework provides facilities to communicate with the different components, even though at this stage, the level of maturity of these components is not enough to run a complete workflow.
4. *Application configuration*. **Partially covered**. The prototype will let users create and configure a basic version of the Application Description, the file that contains the application configuration parameters.

Requirements:

The global requirements for the DECIDE DevOps Framework have been analyzed, reviewed and gathered in D2.1 [1]. The requirements planned for the DevOps Framework and those which have been implemented in this prototype are described on Table 1.

Table 1. Requirements covered by the M15 prototype

Req. ID	Req. Description	Requirement coverage by the prototype
KR1-REQ1	The system must provide the user with an entry point to DECIDE.	The prototype will provide access to a platform from which the different tools can be utilized.
KR1-REQ2	The system must unify transparently the UIs from the different KRs.	The prototype will provide access to the tools, whose UI will be embedded in the platform, following a common set of guidelines.
KR1-REQ3	The system must provide a generic DECIDE UI.	The prototype includes a dashboard that unifies information from some of the tools.
KR1-REQ4	The system must receive ARCHITECT's patterns.	The prototype will receive ARCHITECT's master list of patterns.
KR1-REQ5	The developer must have access to a development environment with the received patterns.	Not covered.
KR1-REQ6	The developer must have access to a development environment with preloaded DECIDE configurations.	Not covered.
KR1-REQ7	The system must allow the developer to submit their code.	The prototype will allow the user to start the compilation process.
KR1-REQ8	The system must be able to version the code submitted by the developer.	Not covered.
KR1-REQ9	The system must be able to resolve the dependencies of the submitted code.	Not covered.
KR1-REQ10	The system must compile the code without errors.	The system will give the option to compile the code.
KR1-REQ11	The system must receive the testing activities that have to be performed on the code.	The prototype will be integrated with a tool able to perform quality tests on the code.
KR1-REQ12	The system must be able to perform the received testing activities.	The integrated quality testing tool will perform testing activities on the code.
KR1-REQ13	The system must present the results from the testing activities.	The prototype's dashboard will present the testing results.
KR1-REQ14	The system must guarantee the continuity of the code within DECIDE's workflow.	The code will reside in a Git repository that is accessible by all tools.
KR1-REQ15	The system must make the code available for DECIDE.	The prototype will provide an option to indicate where the code is located, making it available for all tools.
KR1-REQ16	The system must guarantee the fulfilment of DECIDE's patterns by the developer.	Not covered.
KR1-REQ17	DECIDE DevOps framework must provide support for NFR gathering.	The prototype will provide a wizard that will let the user specify the application's NFRs.

Req. ID	Req. Description	Requirement coverage by the prototype
KR1-REQ18	The system must support developers establishing qualitative NFP that the application must comply with (i.e. security, location, financial, low/high technological risk).	The prototype will provide a wizard that will let the user specify application's NFPs related to location and technological risk.
KR1-REQ19	The system must support developers establishing quantitative NFP that the application must comply with (i.e. MTBF, availability, response time, lag, cost, throughout)).	The prototype will provide a wizard that will let the user specify application's NFPs related to availability and cost.
KR1-REQ20	The system must include a (MC)SLA editor.	Not covered
KR1-REQ21	The system must include an Application Controller.	The prototype will utilize an Application Controller to update the Application Description file.
DEVOPS-REQ1	DECIDE framework must facilitate small and frequent updates of the code.	The prototype will provide continuous integration, which facilitates small and frequent updates of the code.
DEVOPS-REQ4	DECIDE framework must use microservices.	The prototype is build following a microservices architecture.
DEVOPS-REQ5	DECIDE framework must support the continuous integration of the developed apps.	The prototype will support the continuous integration of the code.
DEVOPS-REQ10	DECIDE framework must provide a way for team members to communicate with each other.	Not covered.
DEVOPS-REQ11	DECIDE framework must provide a way for team members to plan the development process.	Not covered.
DEVOPS-REQ13	DECIDE framework must support the application of best practices and design principles during the first phases of the development.	Not covered.

For this prototype, we have focused on developing a strong backend architecture, able to accomplish the basic requirements of the DevOps framework, defining a modular architecture which relies on hosting every tool inside an independent container, so that they can be accessed either from the platform controller, or from any other means. In this initial version of the prototype, we have not considered the user authentication and session functionalities, but we have rather focused on the connectivity between the tools and its orchestration since this is actually the core of DECIDE DevOps framework.

Another goal for this version is to agree on how the UI of each tool should be presented in the DevOps frontend, establishing an agreement between all the DECIDE project members in order to choose common style guidelines for the design of the visual interface.

2.1.1 Fitting into overall DECIDE Architecture

Before explaining in-depth the most important technical aspects of the DevOps framework implementation, we introduce how the framework is connected with the rest of DECIDE modules, and represent the interfaces that enable the communication among them.

As described above, the DevOps framework is responsible for providing an intuitive user interface (UI) to developers and operators, so that they are able to orchestrate the communication between the different DECIDE tools and can provide as input all the parameters necessary to execute them.

Most of the information required by the tools is contained inside the Application Description, which is a configuration file hosted remotely in JSON format, that can be edited by the DevOps framework and by any of the DECIDE tools by pushing those changes to the Git repository where it is stored.

The following picture shows how the DevOps framework fits in the general architecture:

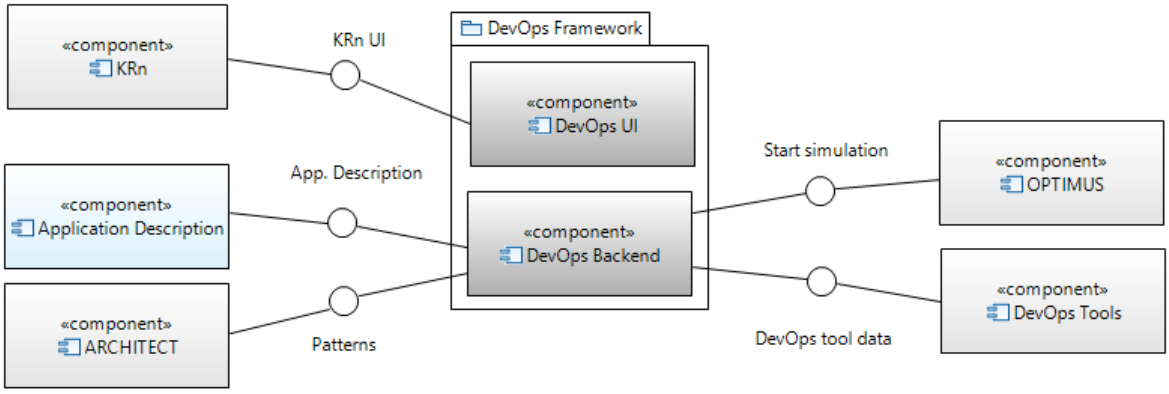

Figure 1. DevOps Framework within DECIDE

2.2 Technical description

In this section we describe the technical specifications of the DevOps framework implementation, explaining the global architecture of the system and the behaviour of the main components.

2.2.1 Prototype architecture

We have designed a microservices architecture based on isolated containers that communicate with each other to obtain the required data. The general architecture of the DevOps framework for this initial version is shown in the diagram below. It is composed of multiple modules that communicate with each other using Cloud Computing techniques, such as service discovery between each module, load balancing to control traffic inside the containers network. The details of each of these components are detailed in the next section 2.2.2

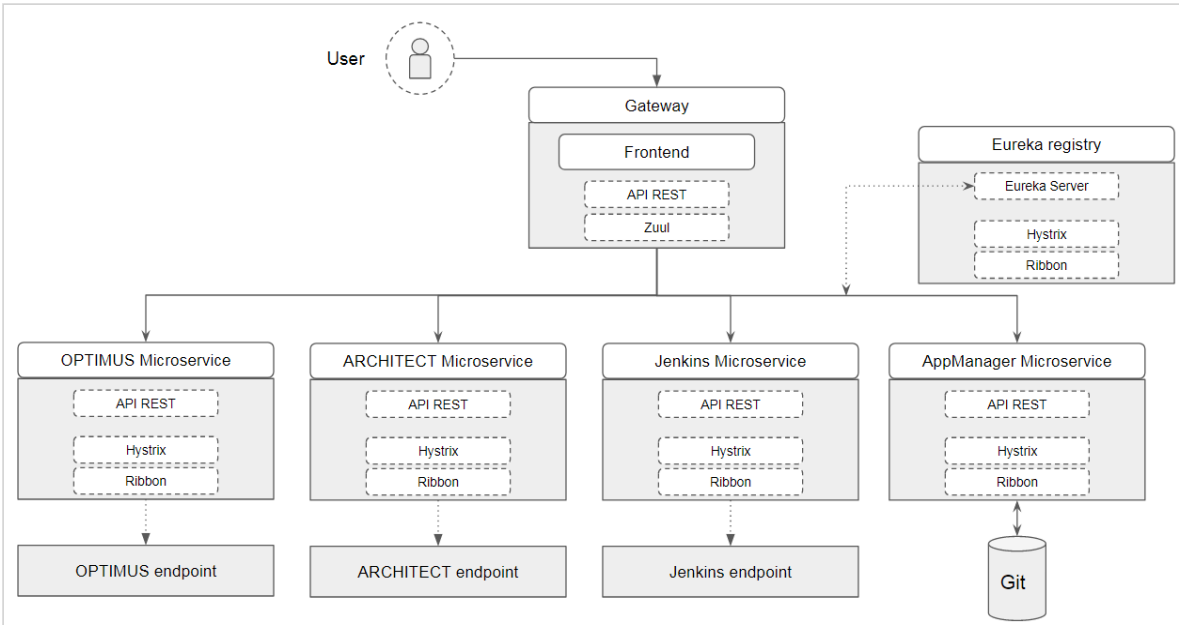

Figure 2. DevOps Framework prototype's architecture diagram

In addition, the DevOps framework will communicate with a local database to store internal information about orchestration between components, and manage the DECIDE applications creation process. This database (not shown in the previous figure) is not yet implemented, since the details of user and application management are not defined at this stage, but it will be included in the next release of the DevOps framework.

Regarding the isolation of each microservice, the DevOps platform has been deployed using Docker technology (more detailed in section 2.2.3), which allows to containerize each application inside a separated component, and redirect the communication with the rest of the network containers, handling network aspects such as service discovery techniques, REST client definition or load balancing between nodes. Finally, this cloud architecture provides a solution ensuring high scalability and fault tolerance, obtaining as a result, a robust approach that allows to implement new tools in the future or adapt the platform easily, in case a tool includes important changes in upcoming versions.

2.2.2 Components description

In this section, we explain each component represented in the above architecture diagram, detailing their main functionalities and specifications in the DevOps framework. We explain firstly several key concepts about the Spring Cloud modules used in the project, so that the internal architecture can be better understood.

2.2.2.1 Spring Cloud

The Spring Cloud framework offers different modules that helps the platform to carry out multiple cloud computing operations automatically, without being concerned on how each node talks to others, or caring about traffic overload between network elements. In the current version of the DevOps framework, we have mainly used three of them:

- ***Zuul:*** It is a very useful component that allows to define an API gateway for the platform, and attach the different microservices path inside the configuration, so in case a request is made to one of these services, *Spring Cloud* will manage the redirection of it. In our case, we have used *Zuul* inside de DECIDE API gateway container, declaring in its configuration the internal paths of each microservices contained in the platform, so in case a service is requested from the outside, the path will be obtained from *Zuul*.

- ***Eureka:*** It is the responsible component for the service discovery procedure. It consists on a centralized server where once a new microservice wakes up, it is registered inside the *Eureka* server. This way, in case other services need information from another service, *Eureka* will provide a suitable domain name to communicate with the destination.
- ***Feign:*** Its main goal is to facilitate the communication from one microservice to another one by consuming an API REST. The *Feign* module allows to create a client with all the API calls of a remote RESTful service, and link these calls with the endpoint transparently. Although, inside the DevOps framework environment, this allows us to access to other tools microservices by importing their corresponding API RESTs as local clients.

There exist other Spring Cloud modules that are interesting for future versions, like the authentication or security modules. Moreover, most of these components includes smaller ones inside, such as ***Hystrix***, which handles the circuit break control and over latency problems inside the cloud, or ***Ribbon***, that manages the load balancing for HTTP requests between microservices.

2.2.2.2 DevOps framework - Backend.

API Gateway

The API Gateway acts as an entrance for the rest of the project submodules. It implements Zuul module from Spring Cloud, so any request made to the platform could be redirected correctly. It also has the frontend dashboard website, which main page is redirected when accessing to the gateway default access point.

Application Description microservice

This service handles the communication with all the remote Application Descriptions included in the DevOps platform. The service contains an API REST to communicate with the *AppManager* tool [2], which provides an accessible communication interface to configure and manage the Git repository where the application description JSON is stored. It also implements the Eureka service discovery, so any other microservice in the platform can retrieve or update any value contained in the application description.

DECIDE tools microservices (OPTIMUS, ARCHITECT …)

This group refers to all the containers whose main goal is to provide a communication interface between the DevOps framework and the remote tool site itself. Depending on the tool, the service will contain specific remote API calls, that retrieve useful data to be displayed in the UI, or acts as a proxy obtaining an embedded UI directly from the component site. Due to the early stage of the tools implementation, the functionalities offered by each tool API are focused on testing the communication obtaining relevant information about the state of the remote service, or displaying a placeholder description about the tool installation process.

Eureka registry server

This component must be initialized first, so once another microservice is launched, it should be registered in the discovery server to be reachable by the rest of the platform microservices.

Database

This module contains a very simple database where the platform saves several information about the tools and DECIDE application basic information, like the location of its application description remote repository, or several information that is still not possible to fulfil from the UI, or be provided by the tools endpoints. It is currently not represented in the architecture diagram because it is not implemented yet, since the details of user and application management are not defined at this stage

2.2.2.3 DevOps framework - Frontend.

All backend microservices are consumed by the main DevOps framework dashboard. Because of the frontend technology used in the implementation, it also has a modular architecture based on components, defined through TypeScript descriptive language. The dashboard consumes de API REST pointing to the gateway endpoint, being able to access each microservice's API REST calls. The content is displayed in the webpage using Angular 2 for server communication, and HTML to structure the application layer.

The DevOps platform's style guidelines follow the Material Design [3] pattern, which have been agreed with the rest of DECIDE tools developers in order to establish a common design pattern for the integrated view.

2.2.3 Technical specifications

As we have presented in previous sections, we have chosen a microservices architecture based on isolated containers that communicate with each order to obtain the required data and serve the frontend resources. Each microservice has been developed using Java programming language, and more concisely using the Spring Boot framework based on MVC architecture. In order to link all the components to each other, the platform has integrated the Spring Cloud toolkit, which offers a complete solution for cloud services development. In the other hand, the frontend is mainly composed by a dashboard which uses Angular 2 technology, where it is represented the tools information obtained from the microservices.

This modular architecture has been achieved by containerizing each component using Docker containers technology. This allows to deploy each tool regardless from the rest of the platform elements, and pass unit tests for each DECIDE tool before test the whole integration together. Each tool has a *Dockerfile* attached so it can be built as a standalone instance, and the DevOps framework contains a *docker compose* file where the instructions to build up the platform with all the tools running as microservices are detailed step by step. This cloud architecture also allows to deploy the scenario as a distributed system, installing each microservice within a cloud provider machine instead of running it inside a local virtualized environment. We will provide a deep explanation on how to deploy the DevOps framework in the next sections.

3 Delivery and usage

3.1 Package information

Each microservice container has almost the same package architecture, because every microservice has been developed using Spring Boot framework. The global architecture of the DevOps framework is represented below.

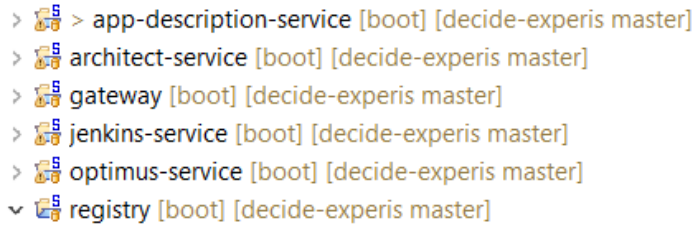

Figure 3. DevOps framework general project's file structure

We will briefly detail the architecture of each component without going too deep, because there are lots of files involved, so we will just explain those that are more representative for a better understanding of the project architecture.

Gateway

Here it is contained the API gateway code, and the frontend code, developed in Angular 2+.

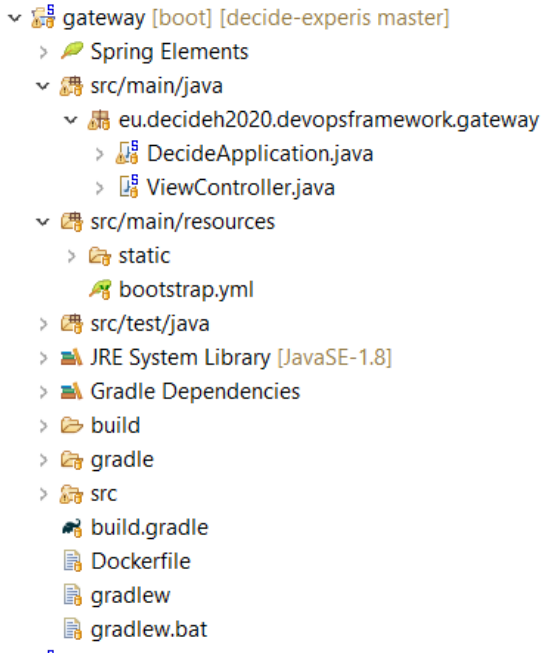

Figure 4. Gateway microservice file structure

ARCHITECT service

It manages the communication with ARCHITECT, in this case the retrieval of the patterns hosted in the ARCHITECT remote tool endpoint.

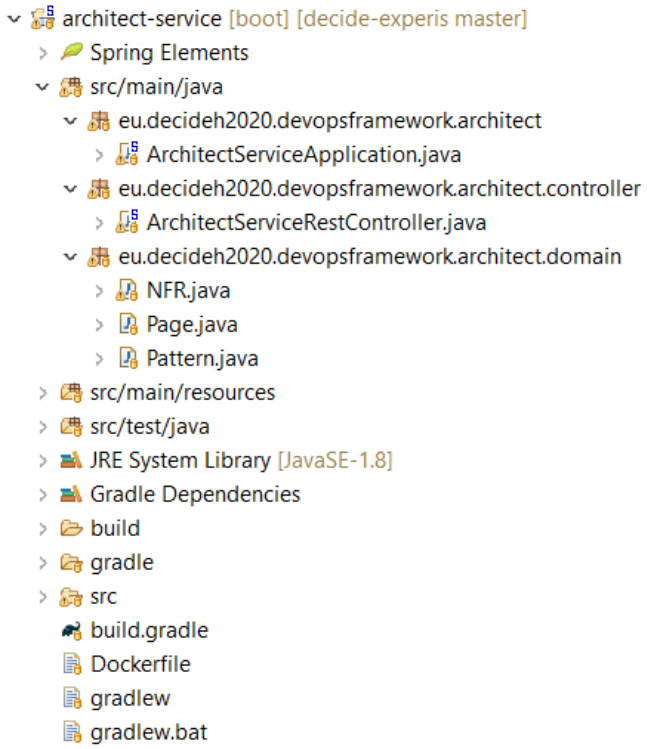

Figure 5. ARCHITECT microservice file structure

OPTIMUS service

It manages the communication with OPTIMUS, in this case the retrieval of the installation process, which is a static content in the current version.

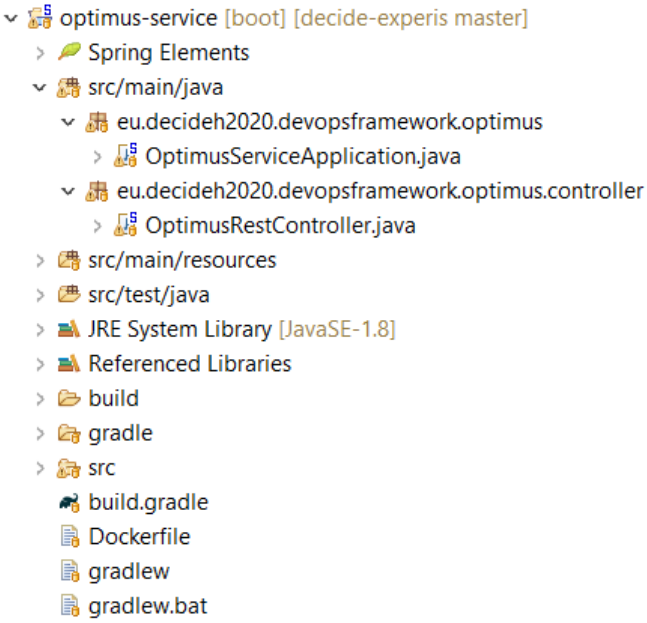

Figure 6. OPTIMUS microservice file structure

App Manager service

It manages the manipulation of the remote application, and the creation of a new DECIDE app. It directly communicates with the Git repository, and has imported the *App Manager* jar file.

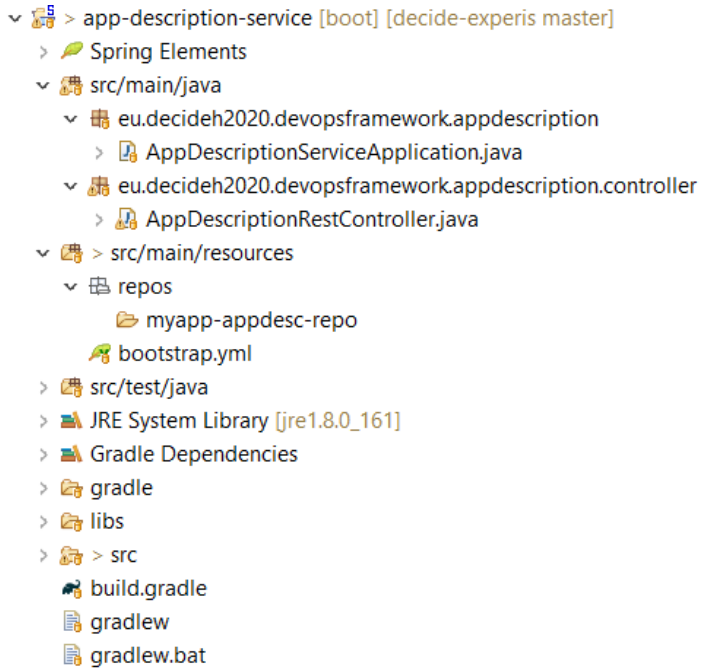

Figure 7. App. Manager microservice file structure

JENKINS service

It obtains the information from the remote Jenkins endpoint associated. It basically retrieves all the jobs status for the linked profile.

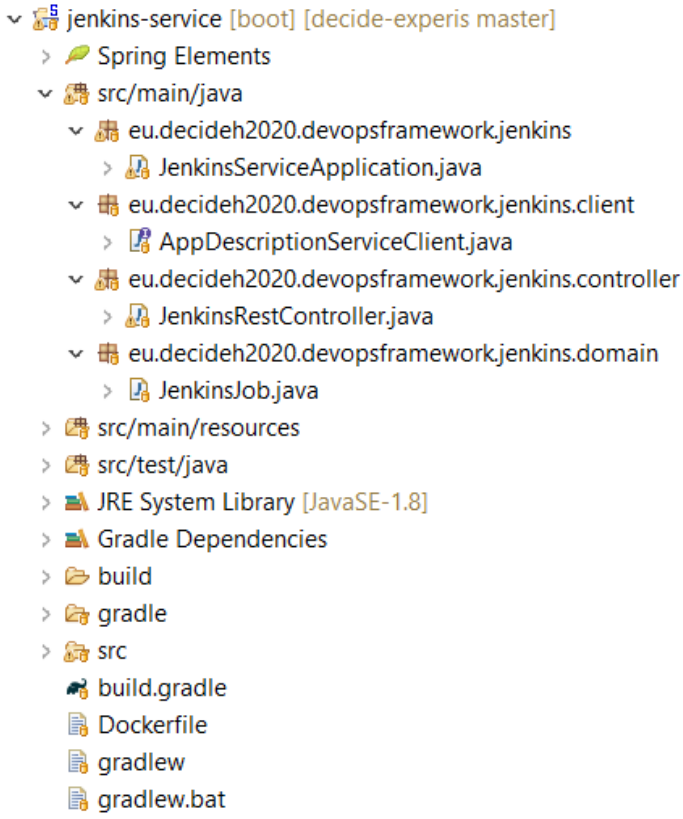

Figure 8. JENKINS microservice file structure

Eureka registry service

It enables service discovery between the rest of the microservices using Eureka Spring Cloud module.

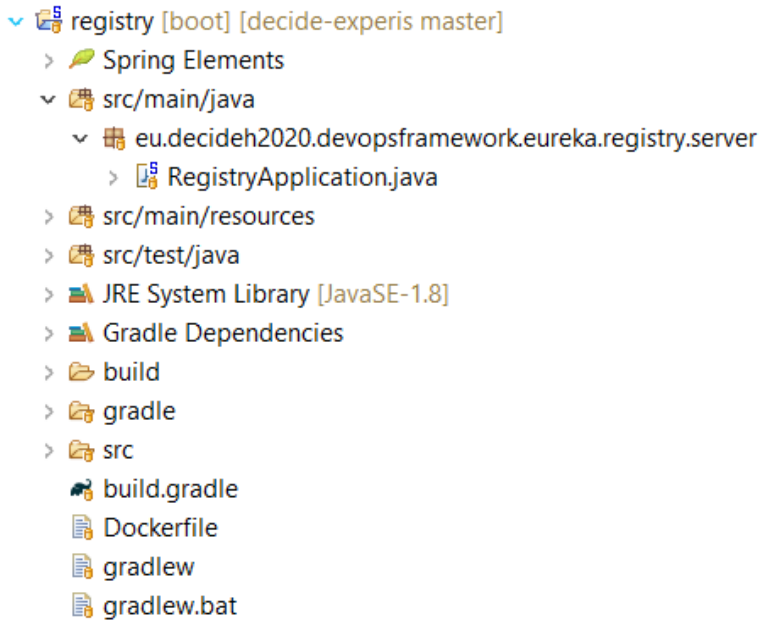

Figure 9. Eureka registry microservice file structure

3.2 Installation instructions

To deploy each container in an easy way, we have created a *docker compose* configuration file so once the user begins the installation process, it starts the initialization of the required services in a background task. The user can also build the Docker images for each microservice by compiling the *Dockerfile* included in each module directory, but this could be a bit tedious, and the services should be instantiated in a certain order so Spring Cloud modules are initialized correctly, and also because module may communicate with others.

Installation requirements

- Have Docker tool installed in your machine and accessible from the terminal.
- Have Git installed, or just unzip the compressed file downloaded from the repository (see section 3.5).
- We also recommend running the DECIDE DevOps framework in a powerful machine, because the project is composed of a total of 6 Docker containers and that may consume some of your RAM resources. Our recommendation is to have a minimum of 4Gb RAM resources and about 1GB free for storage.

Getting started

1. Clone the DevOps framework Git repository in your computer.
2. Navigate to the main root directory of the project
3. Run in the console the command `docker-compose up`
4. It will automatically deploy all the microservices containers in your *localhost* domain. This deployment may take a few minutes (about 1 minute), to be fully configurated and accessible from your browser.

5. Access to the main DevOps framework web page in http://localhost:4000 in your local machine browser.
6. You can access with any credentials, since authentication module is still not defined. Try with ***admin*** as user and password***.***

Production deployment

A production deployment will be available for all DECIDE team members.

3.3 User Manual

Once the microservices scenario deployment process has finished, the user should access to the gateway entrance to visualize the DevOps framework (http://localhost:4000). The URL and port mapping for each container is the following:

- DevOps framework dashboard (gateway): http://localhost:4000
- Jenkins service API endpoint: http://localhost:4000/jenkins
- Application description API endpoint: http://localhost:4000/appdesc
- Architect API endpoint: http://localhost:4000/architect
- OPTIMUS API endpoint: http://localhost:4000/optimus
- Eureka registry dashboard: http://localhost:8761

3.4 Licensing information

This component is offered under the MIT license.

3.5 Download

The source code is uploaded in WP2 DECIDE git repository and available here:

https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/master/DevOpsFramework

You can also access to our testable deployed version by accessing to:

http://mng.experis.es/decide

4 Conclusions

This document presents the initial prototype of the DevOps framework, corresponding to the M15 release. It describes the prototype from a functional and a technical point of view, and it contains usage and installation instructions for the component.

The next release of this deliverable (D2.7) will document the second version of the DevOps framework, which will provide new functionalities and a higher-level of integration with all the DECIDE tools and KRs.

References

[1] DECIDE Consortium, “D2.1 - Detailed requirements specification v1,” 2017.

[2] DECIDE Consortium, “AppManager,” 2018. [Online]. Available: https://git.code.tecnalia.com/decide/WP3/tree/master/AppManager. [Accessed 20 February 2018].

[3] Google, "Material Design," [Online]. Available: https://material.io/. [Accessed 20 February 2018].