



---

## Deliverable D2.7

### Intermediate DECIDE DevOps Framework Integration

---

<b>Editor(s):</b>	José Manuel López Javier Gavilanes Ruano
<b>Responsible Partner:</b>	Experis IT
<b>Status-Version:</b>	Final – v1.0
<b>Date:</b>	28/02/2019
<b>Distribution level (CO, PU):</b>	PU

<b>Project Number:</b>	GA 726755
<b>Project Title:</b>	DECIDE

<b>Title of Deliverable:</b>	D2.7 Intermediate DECIDE DevOps Framework Integration
<b>Due Date of Delivery to the EC:</b>	28/02/2019

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

<b>Abstract:</b>	This deliverable will provide an intermediate version of the integrated DECIDE DevOps Framework. This intermediate version will augment the initial version functionalities taking into consideration the feedback coming for the use cases.
<b>Keyword List:</b>	DevOps framework, integration, multi-cloud, microservice.
<b>Licensing information:</b>	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.
<b>Disclaimer</b>	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	05.02.2019	First draft version	Experis IT
V0.2	20.02.2019	Sent to internal review	TECNALIA
V0.3	25.02.2019	Modified after internal review	Experis IT
V1.0	28.02.2019	Ready for submission	TECNALIA

---

## Table of Contents

---

Table of Contents .....	4
List of Figures.....	5
List of Tables.....	5
Terms and abbreviations.....	6
Executive Summary .....	7
1 Introduction.....	8
1.1 About this deliverable.....	8
1.2 Document structure.....	8
2 DECIDE Development and Integration .....	9
2.1 Staging integration environment:.....	9
2.2 Production integration environment.....	15
2.3 Code structure .....	16
3 DECIDE UI .....	16
4 DECIDE Orchestration.....	18
5 Secrets sharing .....	21
6 Implementation.....	22
6.1 Functional description .....	22
6.1.1 Fitting into overall DECIDE Architecture.....	25
6.2 Technical description .....	26
6.2.1 Prototype architecture .....	26
6.2.2 Components description .....	27
6.2.2.1 User and application management.....	27
6.2.2.2 Vault.....	27
7 Delivery and usage .....	30
7.1 Package information .....	30
7.1.1 DevOps Framework Client.....	30
7.1.2 DevOps Framework Server .....	33
7.2 Installation instructions .....	33
7.3 User Manual.....	34
7.4 Licensing information .....	35
7.5 Download.....	35
8 Conclusions.....	36
References.....	37
Annex A. Code snippets.....	38
Guards .....	38
Login .....	38

NFR Editor HTML .....	39
Services (decide.gateway.service.ts) .....	40
DevOps Framework Server (appManagerController) .....	47

---

## List of Figures

---

<b>FIGURE 1.</b> DEVELOPMENT PROCESS IN DECIDE .....	9
<b>FIGURE 2.</b> GIT WHERE THE TOOLS' CODE IS STORED .....	10
<b>FIGURE 3.</b> CODE STRUCTURE WITHIN GIT .....	10
<b>FIGURE 4.</b> STAGING VARIABLES IN THE INTEGRATION ENVIRONMENT .....	11
<b>FIGURE 5.</b> OVERVIEW OF THE BUILD SETUP .....	12
<b>FIGURE 6.</b> OVERVIEW OF THE TASKS NEEDED TO START MICROSERVICES .....	13
<b>FIGURE 7.</b> RELEASE CREATION MENU .....	14
<b>FIGURE 8.</b> REDEPLOYMENT MENU .....	14
<b>FIGURE 9.</b> OVERVIEW OF STEPS REQUIRED FOR THE DEPLOYMENT .....	15
<b>FIGURE 10.</b> SAMPLE OF A REDEPLOYMENT LOG .....	15
<b>FIGURE 11.</b> GENERAL EDITOR. MICROSERVICES CREATION .....	16
<b>FIGURE 12.</b> GENERAL EDITOR. NFRS DEFINITION .....	17
<b>FIGURE 13.</b> ACSMI DISCOVERY'S IFRAME IN THE DEVOPS FRAMEWORK .....	17
<b>FIGURE 14.</b> DEVOPS FRAMEWORK DASHBOARD. OPERATION SECTION .....	18
<b>FIGURE 15.</b> DECIDE STATE MACHINE DIAGRAM .....	19
<b>FIGURE 16.</b> VAULT BASIC WORKFLOW [2] .....	21
<b>FIGURE 17.</b> DEVOPS FRAMEWORK WITHIN DECIDE .....	25
<b>FIGURE 18.</b> DEVOPS FRAMEWORK PROTOTYPE'S ARCHITECTURE DIAGRAM .....	26
<b>FIGURE 19.</b> SCHEMA OF THE DATABASE FOR USER MANAGEMENT .....	27
<b>FIGURE 20.</b> VAULT IN DECIDE .....	28
<b>FIGURE 21.</b> DEVOPS FRAMEWORK CLIENT'S FILE STRUCTURE .....	30
<b>FIGURE 22.</b> HTML CODE OF ACSMI CONTRACTING MODULE .....	31
<b>FIGURE 23.</b> TS CODE OF ACSMI DISCOVERY MODULE .....	31
<b>FIGURE 24.</b> STRUCTURE OF THE "COMPONENTS" MODULE .....	31
<b>FIGURE 25.</b> STRUCTURE OF THE "MODELS" MODULE .....	32
<b>FIGURE 26.</b> CODE SNIPPET OF THE "MODELS" MODULE .....	32
<b>FIGURE 27.</b> STRUCTURE OF THE "SERVICES" MODULE .....	33
<b>FIGURE 28.</b> DEVOPS FRAMEWORK SERVER'S FILE STRUCTURE .....	33

---

## List of Tables

---

<b>TABLE 1.</b> RELATIONSHIP BETWEEN STATES AND ENABLED TOOLS .....	19
<b>TABLE 2.</b> REQUIREMENTS COVERED BY THE M27 PROTOTYPE .....	23
<b>TABLE 3.</b> ENDPOINTS OF DECIDE COMPONENTS .....	34

---

## Terms and abbreviations

---

ACSml	Advanced Cloud Service (meta-) Intermediator
ADAPT DO	ADAPT Deployment Orchestrator
ADAPT MM	ADAPT Monitoring Manager
API	Application Programming Interface
EC	European Commission
GUI	Graphical User Interface
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 second release of this component offers some improvements at a graphical level, it integrates all the DECIDE tools, and provides new functionalities compared to the first release, such as infrastructure to share confidential information amongst tools with Vault.

Since the first release, an integration environment has been set up to provide access to partners to the DECIDE Platform. This environment is configured as a two-stages environment and is described within the document.

The graphical interface has also been improved and serves as an entry point to access the UIs of all components.

The DevOps Framework also handles the DECIDE workflow. A first approach to implement this functionality has been tackled in this document, with the creation of a state machine to control the contents of the application description.

Lastly, the technical implementation of this component can be found at the end of the document.

# 1 Introduction

## 1.1 About this deliverable

This deliverable explains the architecture of the second DECIDE DevOps framework prototype. It introduces the new implemented functionalities for the M27 prototype and includes a technical description of this component.

Furthermore, the deliverable gives an overview of the integration environment that has been set up in AIMES premises, to test the integration of the DECIDE KRs and validate it in the use cases.

## 1.2 Document structure

The document is structured in six (6) main sections:

- Section 2 explains the DECIDE integration environment.
- Section 3 details the DevOps Framework UI.
- Section 4 describes the orchestration strategy, based on a state machine.
- Section 5 provides detail of the deployment of Vault, a component to share secrets amongst the DECIDE tools.
- Section 6 contains the technical description of the DevOps framework and,
- Section 7 provides delivery and usage information



## 2 DECIDE Development and Integration

This section presents the process followed in DECIDE that has been set up for development, testing and integration of its tools and KRs. It builds upon and updates the integration strategy presented in Deliverable D2.3 Integration and validation strategy [1], with the actual implemented methodology, tools and approach.

The development process in DECIDE is set up in three stages, as the following figure shows:

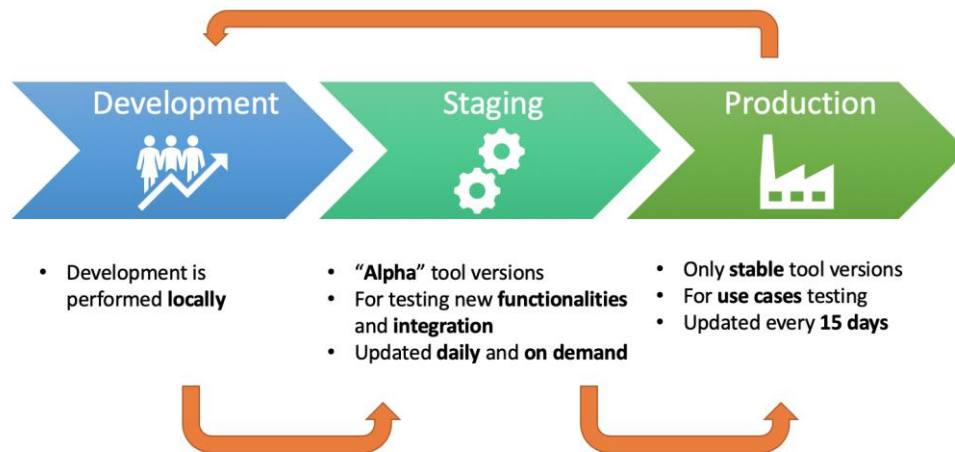


Figure 1. Development process in DECIDE

- During the **development** stage, the different tools are developed by the responsible partners, locally, and tested in isolation before moving to the staging stage.
- In the **staging** stage, all DECIDE KRs are deployed in a common environment, where integration tests can take place. This environment is meant to check the correct integration of the components, and said components are redeployed whenever the developer changes the code.
- Lastly, in the **production** stage, only stable versions of the tools are deployed, after making sure that they are working as intended. This environment is manually rebuilt every 15 days.

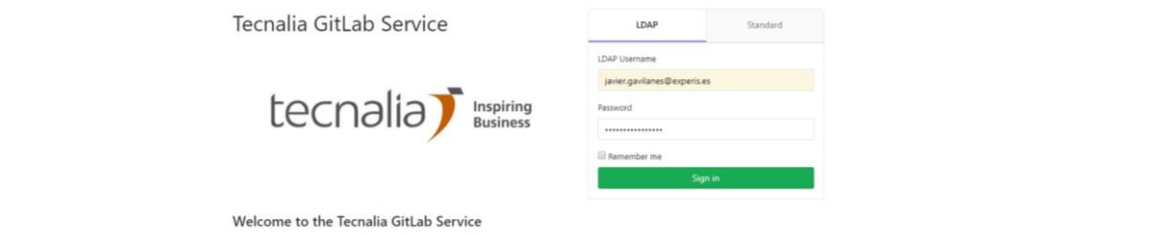
As it has been mentioned, the development is performed locally by the partners, but the staging and production stages are integration environments, set up in AIMES premises, to which all partners have access. These environments will be described in the following section.

### 2.1 Staging integration environment:

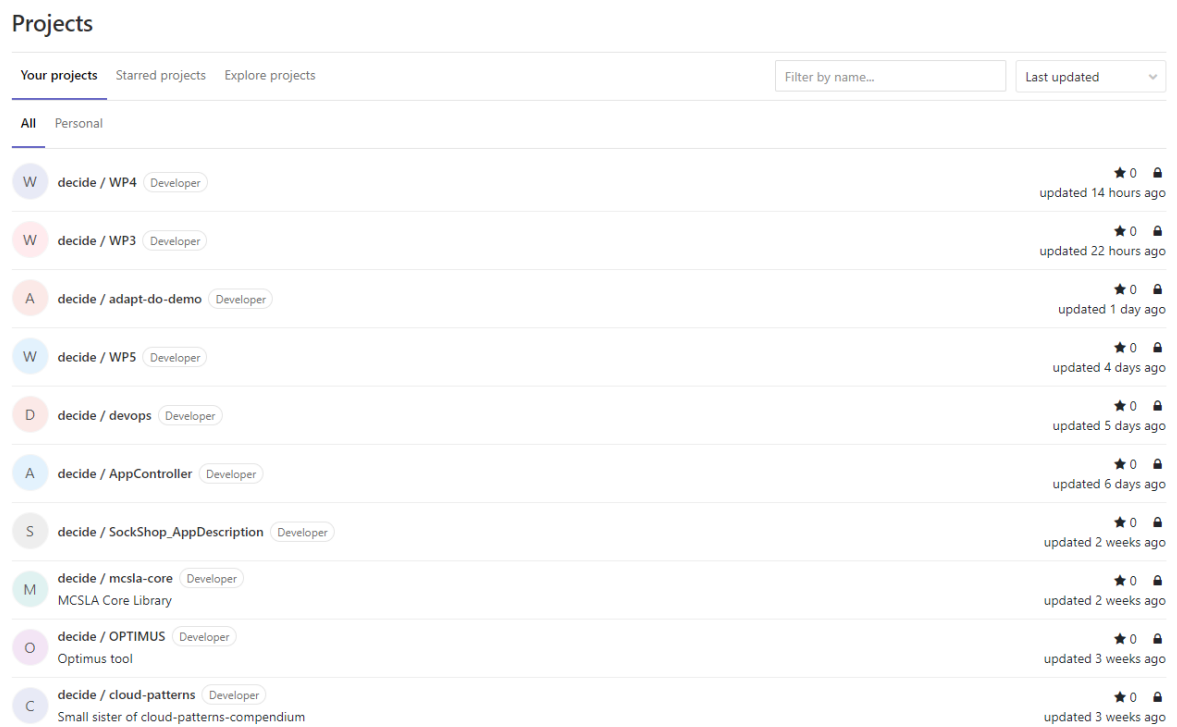
The integration environment is located in an AIMES machine, to which a public IP has been assigned. This environment is meant to test recent changes to the tools and the integration of the different components. As such, whenever a change is committed to a tool, said tool is automatically redeployed.

The integration environment is composed of the following tools:

- **Git**: there exists a remote repository for the DECIDE project where the different code versions of each of the developed services are stored. The git repository is hosted at TECNALIA.



**Figure 2.** Git where the tools' code is stored



**Figure 3.** Code structure within Git

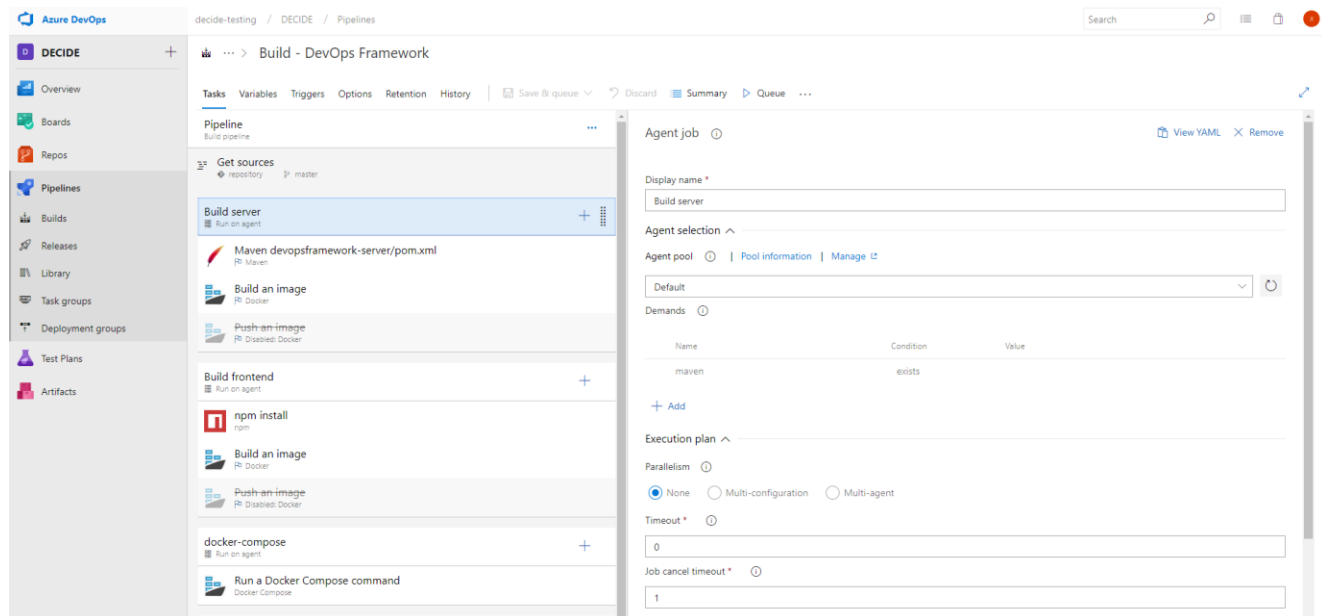
- **Azure Devops:** this tool allows to automate processes within the lifecycle of applications, such as updating the code or provisioning a new environment in a simple way. It covers the commonly defined as continuous integration and continuous delivery in the DevOps philosophy. The steps to follow to automate the services are:
  - Define variables: In the “Pipelines - Library” section, the staging variables used by the DECIDE microservices are defined:

## Variables

Name ↑	Value	🔒
ACSMI_VERSION	d3657b5b9af7db9957dc4473b2ac452edd54f988	
ACSMIBackendContainerName	acsmi-discovery-server	
ACSMIBackendImageTag	acsmi-discovery-server	
ACSMIBackendPortMapping	3307:3306	
ACSMIBaseServerImageTag	eu.decideh2020.int.springboot.server.repo	
ACSMIDiscoveryNetworkName	acsmi-discovery	
ACSMIFrontendContainerName	acsmi-discovery-client	
ACSMIFrontendImageTag	acsmi-discovery-client	
ACSMIFrontendPortMapping	8087:8080	
ACSMIMonitoringContainerName	acsmi-monitoring	
ACSMIMonitoringImageTag	acsmi-monitoring	

**Figure 4.** Staging variables in the integration environment

- **Make a build:** for the first deployment, it is necessary to generate a new build and define all required tasks:



**Figure 5.** Overview of the build setup

- **Build Task:** each of the necessary tasks to start a micro service (get Git sources, Maven compile, build images...):

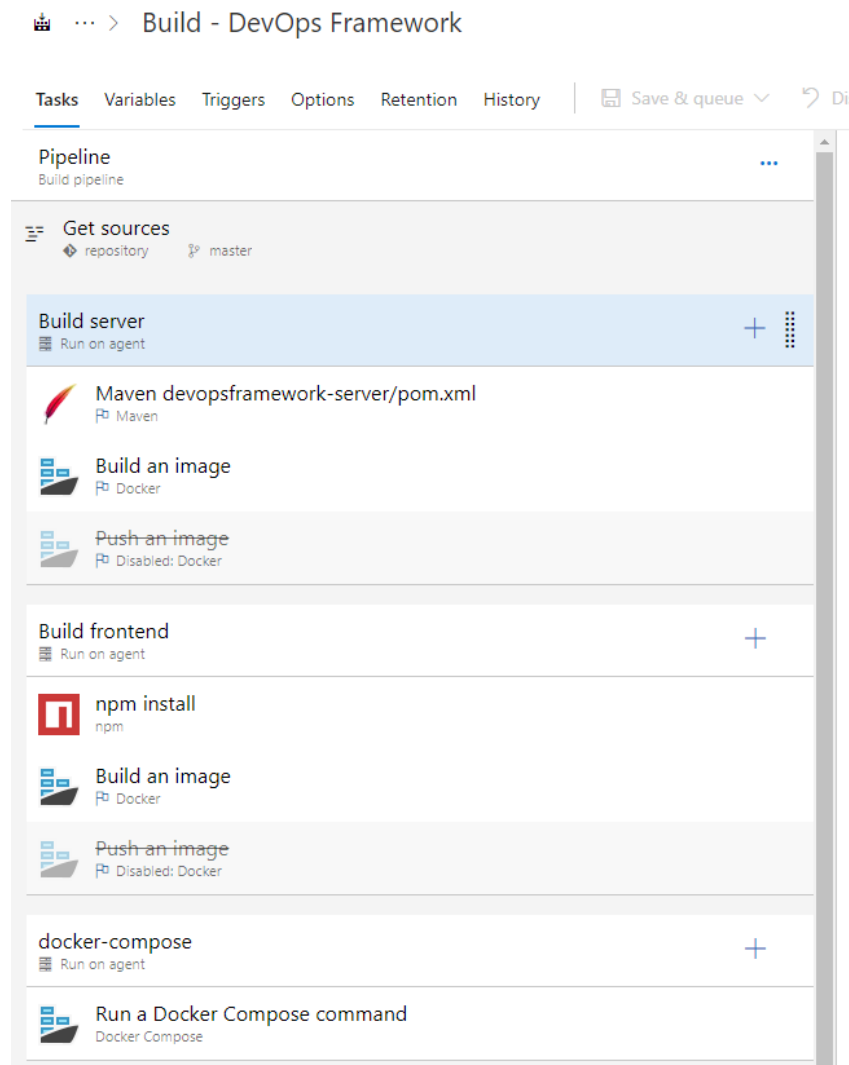


Figure 6. Overview of the tasks needed to start microservices

- **Make a release:** When a build is running properly, it is time to create a new release connected to the build:

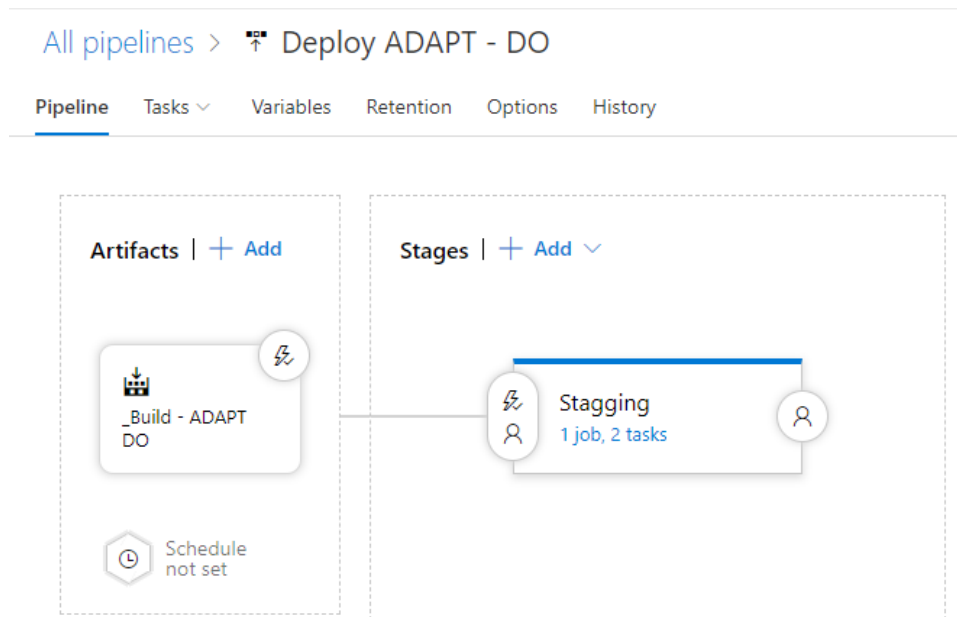


Figure 7. Release creation menu

- Redeploy: Normally this operation is launched automatically when the code changes, but there exists the option to force the redeployment manually:

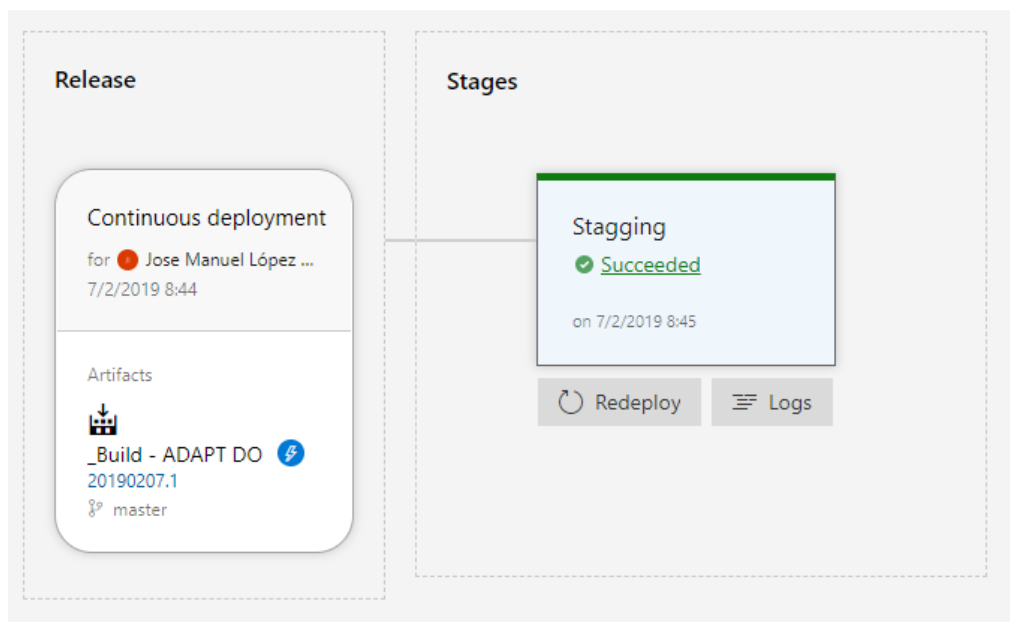


Figure 8. Redeployment menu

Logs

Summary

Tests

✓

Build server Job

Succeeded

✓

Build frontend Job

Succeeded

✓

docker-compose Job

Succeeded

✓

Agent phase Job

Succeeded

Build server Job

Pool: Default

Agent: decide-02

Started: 14/2/2019 17:36:19

... 1m 49s

✓

Prepare job

succeeded

<1s

✓

Initialize Job

succeeded

<1s

✓

Get Sources

succeeded

3s

✓

Maven devopsframework-server/pom.xml

succeeded

1 warning

1m 37s

✓

Build an image

succeeded

5s

✓

Post Job Cleanup

succeeded

<1s

Figure 9. Overview of steps required for the deployment

- **Logs:** all redeployment operations generate logs. These logs are useful to know if the microservices can start or return an error:

```

Maven devopsframework-server/pom.xml
Previous task Next task X

1 #[section]Starting: Maven devopsframework-server/pom.xml
2 =====
3 Task : Maven
4 Description : Build with Apache Maven
5 Version : 2.147.1
6 Author : Microsoft Corporation
7 Help : [More Information](https://go.microsoft.com/fwlink/?linkID=613723)
8 =====
9 [command]/usr/bin/mvn -version
10 Apache Maven 3.3.9
11 Maven home: /usr/share/maven
12 Java version: 1.8.0_171, vendor: Oracle Corporation
13 Java home: /usr/lib/jvm/java-8-oracle/jre
14 Default locale: en_GB, platform encoding: UTF-8
15 OS name: "linux", version: "4.4.0-141-generic", arch: "amd64", family: "unix"
16 [command]/usr/bin/mvn -f /home/javier/vstsagent/_work/10/s/devopsframework-server/pom.xml help:effective-pom
17 [INFO] Scanning for projects...
18 [INFO]
19 [INFO] -----
20 [INFO] Building devopsframework-server 0.0.1-SNAPSHOT
21 [INFO] -----
22 [INFO]
23 [INFO] --- maven-help-plugin:2.2:effective-pom (default-cli) @ devopsframework-server ---
24 [INFO]
25 Effective POMs, after inheritance, interpolation, and profiles are applied:
26
27 <?xml version="1.0" encoding="UTF-8"?>
28 <!-- ===== -->
29 <!-- -->
30 <!-- Generated by Maven Help Plugin on 2019-02-14T04:36:36 -->
31 <!-- See: http://maven.apache.org/plugins/maven-help-plugin/ -->
32 <!-- -->
33 <!-- ===== -->
34
35 <!-- ===== -->
36 <!-- -->
37 <!-- Effective POM for project -->
38 <!-- 'eu.h2020.devopsframework.server:devopsframework-server:jar:0.0.1-SNAPSHOT' -->
39 <!-- -->
40 <!-- ===== -->
41
42 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
43 <modelVersion>4.0.0</modelVersion>
44 <parent>
45 <groupId>org.springframework.boot</groupId>
46 <artifactId>spring-boot-starter-parent</artifactId>
47 <version>1.5.9.RELEASE</version>
48 </parent>
49 <groupId>eu.h2020.devopsframework.server</groupId>
50 <artifactId>devopsframework-server</artifactId>
51 <version>0.0.1-SNAPSHOT</version>
52 <name>devopsframework-server</name>
53 <description>Base for Spring Boot projects with MongoDB and JWT authentication</description>
54 <url>http://projects.spring.io/spring-boot/devopsframework-server</url>

```

Figure 10. Sample of a redeployment log

## 2.2 Production integration environment

As stated in the introduction of the section, there is a second integration environment in place, where only stable versions of the tools are deployed. The goal of this environment is to have a more static version of DECIDE, for high-level testing and for use-cases testing.

This production environment is set up exactly the same way as the staging one, in a second AIMES machine, with the only difference that the automatic redeployment is disabled. It has been agreed that a new deployment will take place every 15 days, to give time to new tool versions to be polished in the staging environment before moving onto production.

## 2.3 Code structure

To setup the integration environment described above, the code structure within Git must follow a common approach, distinguishing between stable versions of the tools and versions in development. Thus, the following structure has been agreed upon all partners:

Two different branches will be created:

- **Master** branch: this branch will host the most recent version of the tools, where new functionalities will be implemented. The code on the *master* branch will be deployed in the staging environment whenever a new version is released.
- **Release** branch: this branch will hold only stable versions of the tools. It will be updated with a new version once it has been proved that said version is bug-free and working properly. The code on the *release* branch will be deployed in the production environment every 15 days.

Besides, the different tool releases in the *release* branch will be tagged to give the possibility of rolling back to a previous version, in case something goes wrong with the tool.

## 3 DECIDE UI

The DECIDE Framework provides a graphical user interface to provide access to the different Key Results.

On one hand, the framework includes a “General Editor” that allows a user to create a DECIDE application, that is, introduce the most relevant data about the application (name, location of the code, number of microservices, NFRs, ...) which will later be written into the Application Description. It also provides the option of importing a previously created Application Description. The current version of the DevOps framework is adapted to the latest NFR format. The images below show the General Editor:

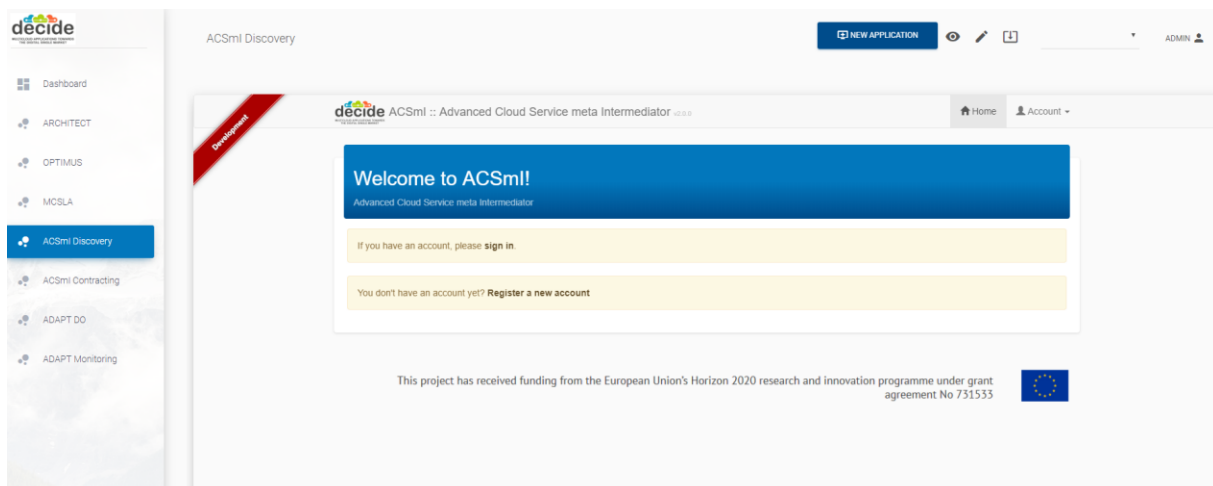
The screenshot shows the 'General Editor' interface for creating microservices. At the top, there are four tabs: 'General', 'Microservices', 'NFRs', and 'Preview'. The 'Microservices' tab is currently selected. Below the tabs, the title 'Microservices' is displayed. The main content area is titled 'Microservice 1' and contains several input fields and checkboxes. The 'NAME' field has a placeholder 'Enter DECIDE app name'. The 'PROGRAMMING LANGUAGE' field has a placeholder 'Enter microservice programming language'. There are two checkboxes: 'Stateful' and 'Public IP'. Below these is a 'TAGS' section with a text input field and an 'ADD' button. At the bottom of the form, there is a purple button labeled 'Application' with a star icon. In the bottom right corner of the interface, there are two buttons: 'SOCKSHOP MICROSERVICES' and a '+' button.

Figure 11. General editor. Microservices creation



**Figure 12.** General editor. NFRs definition

On the other hand, it integrates the graphical interfaces of the Key Results, so that they can be accessed from one point. The preferred means of integration is through the use of an iframe, but for some of the tools, the DevOps Framework constructs their UI, with an API-based approach. More information about the integration of tools at a GUI level can be found on deliverable D2.2 [2]. The following figure shows an example of an iframe integrated in the DevOps Framework (ACSml Discovery's UI):



**Figure 13.** ACSml Discovery's iframe in the DevOps Framework

Lastly, the DevOps Framework provides a dashboard to give an overview of the status of the application, regarding code development, patterns or deployment situation. The following image shows the detail of the "Operation" part of the Dashboard:

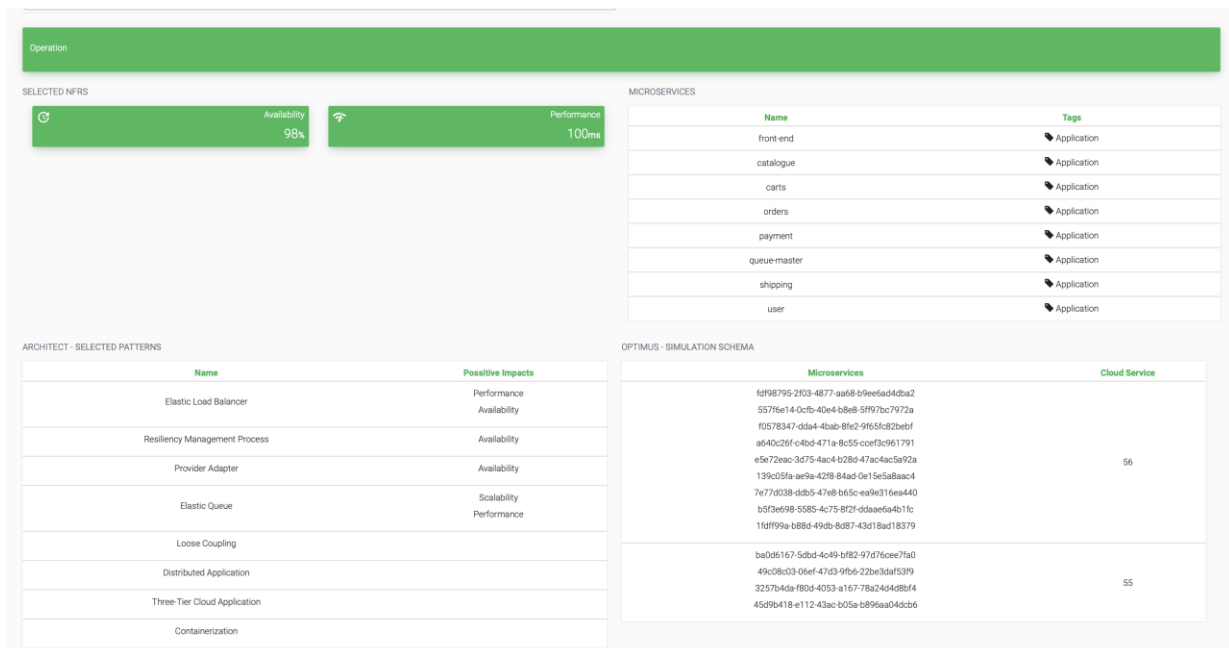


Figure 14. DevOps Framework Dashboard. Operation section

## 4 DECIDE Orchestration

There are a lot of tools that intervene in DECIDE and that have to be called at the appropriate moment. Furthermore, these tools need that certain information about the project being developed has been introduced in the application description in order to work properly.

Because of this reason, a component that takes care of these issues is required. This component has to invoke the correct tool at the right time and should also enable or disable access to the tools, depending on the information already included in the application description. It is the DevOps Framework the tool that handles this, since it is the entry point to DECIDE and the most centralized component.

Due to the complexity of the DECIDE workflow and sub-workflows, this is not a trivial task. As a first step towards solving it, a state machine has been developed, to understand and clarify the information needs of the tools and the dependencies between the application description variables.

The state machine described below aims at giving an overview of what variables in the Application Description<sup>1</sup> are needed for each tool to be able to work. States represent tools that are ready to be used and transitions represent the introduction of a set of variables in the Application Description.

This state machine is a first version that will be expanded and implemented for the final release:

<sup>1</sup> The Application Description is the procedure that DECIDE has selected to make all tools interoperable

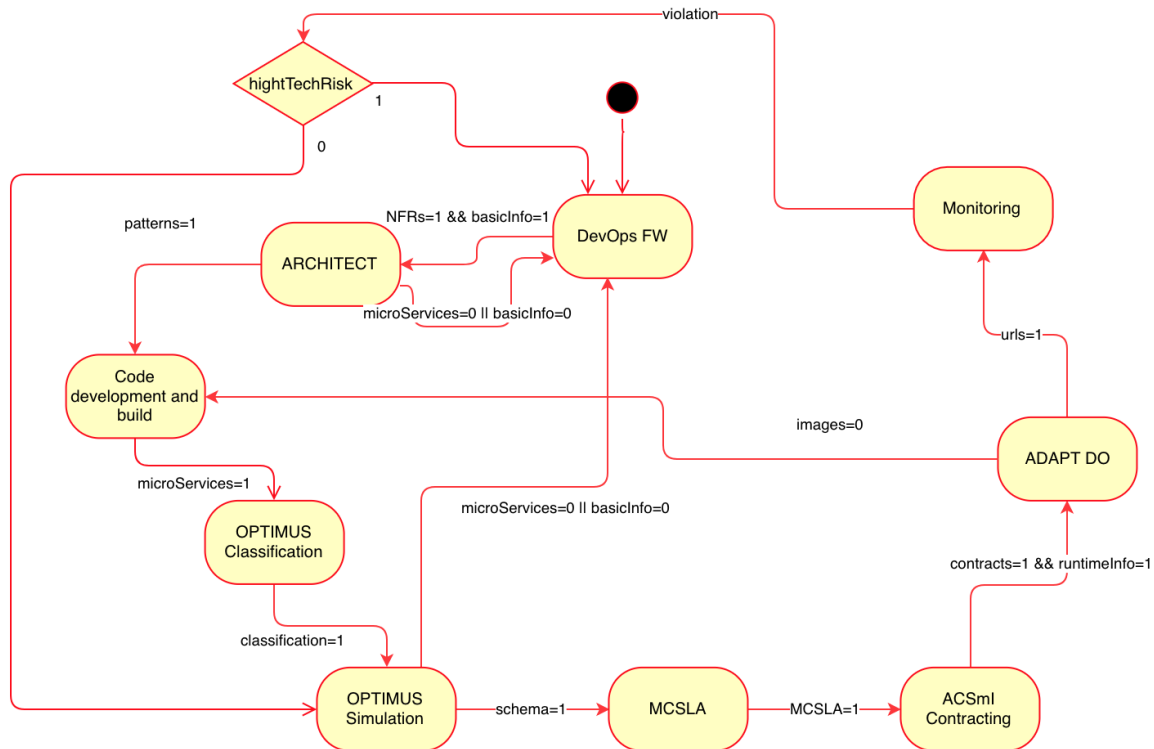


Figure 15. DECIDE State machine diagram

As stated above, the different states of the diagram represent the tools that are “enabled”, that is, they have all the necessary information to work. For simplicity, an enabled tool indicates that all “previous” tools (tools that have intervened before) are also enabled. For example, the state “OPTIMUS Simulation” represents that the OPTIMUS Simulation tools is enabled, but also the Code development and build, ARCHITECT and DevOps Framework tools are enabled in that state. The following table shows the relationship between the states and the tools enabled in each state:

Table 1. Relationship between states and enabled tools

State	Enabled tools
DevOps Framework	DevOps Framework
ARCHITECT	DevOps Framework ARCHITECT
Code development and build	DevOps Framework ARCHITECT (Eclipse)
OPTIMUS Classification	DevOps Framework ARCHITECT OPTIMUS Classification
OPTIMUS Simulation	DevOps Framework ARCHITECT OPTIMUS Classification OPTIMUS Simulation
MCSLA	DevOps Framework ARCHITECT OPTIMUS Classification OPTIMUS Simulation

State	Enabled tools
	MCSLA
ACSml Contracting	DevOps Framework ARCHITECT OPTIMUS Classification OPTIMUS Simulation MCSLA ACSml Contracting
ADAPT DO	DevOps Framework ARCHITECT OPTIMUS Classification OPTIMUS Simulation MCSLA ACSml Contracting ADAPT DO
Monitoring	DevOps Framework ARCHITECT OPTIMUS MCSLA ACSml Contracting ADAPT DO Monitoring (ADAPT MM and ACSml Monitoring)

The “Code development and build” state represents that all the basic metadata about the project (name, description, NFRs) and patterns are already available, hence allowing the development team to start developing. The following tools, however, do not require that the code or the images are available, except ADAPT DO.

The “Monitoring” state represents that both ADAPT monitoring and ACSml monitoring are enabled.

As for transitions, they reference the group of variables in the Application Description that allow a tool to work properly, i.e. the minimum amount of information the tool needs. Once that information has been introduced in the Application Description, the state (tool) is enabled. Below, these transitions are explained:

- **basicInfo**: basic project’s information, introduced from the Wizard (name, Git repository...)
- **microServices**: information regarding the microservices
- **NFRs**: selected NFRs
- **patterns**: patterns selected by the user from the list provided by ARCHITECT
- **images**: images of the developed microservices are available
- **classification**: information generated by OPTIMUS classification and required by OPTIMUS Simulation
- **schema**: schema proposed by OPTIMUS and selected by the user
- **MCSLA**: MCSLA created
- **contracts**: contracts are created
- **runtimeInfo**: info that ADAPT needs for deployment
- **urls**: URLs generated by ADAPT where the microservices are deployed
- **violation**: a violation is received

## 5 Secrets sharing

The DevOps framework must provide the necessary infrastructure for tools to be able to share sensible information amongst themselves. For this task, it has been opted to use Vault.

Vault is a tool, developed by HashiCorp, for securely storing and accessing sensitive information, or *secrets*, such as API keys, passwords or certificates.

The main features of Vault are [1]:

- **Secure Secret Storage:** Vault can store arbitrary key/value secrets. These secrets are encrypted prior to writing them to persistent storage, so gaining access to the raw storage is not enough to access the secrets.
- **Dynamic Secrets:** Vault can generate secrets on-demand for some systems, such as AWS or SQL databases. After creating these dynamic secrets, Vault will also automatically revoke them after the lease is up.
- **Data Encryption:** Vault can encrypt and decrypt data without storing it. This allows to define encryption parameters and to store encrypted data in a location without having to design specific encryption methods.
- **Leasing and Renewal:** All secrets in Vault have a *lease* associated with them. At the end of the lease, Vault will automatically revoke that secret. Clients are able to renew leases via built-in renew APIs.
- **Revocation:** Vault can revoke not only single secrets, but a tree of secrets, for example all secrets read by a specific user, or all secrets of a particular type.

### VAULT IN DECIDE

The basic working process of Vault in DECIDE is depicted in the figure below:

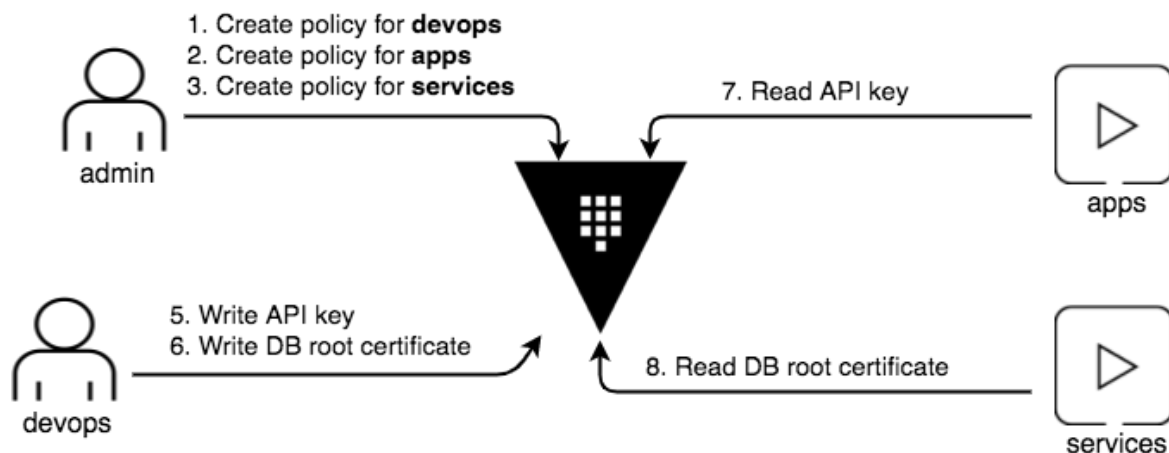


Figure 16. Vault basic workflow [2]

1. An administrator is in charge of creating the policies that define what secrets can be accessed by each component.
2. A DevOps operator (this role can coincide with the administrator role) will store the secrets inside Vault.
3. Tokens, which are associated to a certain policy, thus controlling what data they can access, are created by the administrator and distributed to the tools that will access the vault.
4. Whenever a tool or component needs a secret, it will request it to Vault, obtaining access to it by means of the provided token.

## 6 Implementation

### 6.1 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 provides an entry point to DECIDE and handles the interconnection between all the elements involved, providing a global overview about the state of the application to the end user. Furthermore, the DevOps Framework takes care of the user and application management and provides the necessary infrastructure to safely store and share sensitive information.

#### 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 [3] and updated in D2.5 [5].
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.
5. *User and application management:* The DevOps framework will manage user access to the platform and the application(s) that each user is working on.
6. *Secrets management:* The framework will provide infrastructure to share and store secrets (credentials, tokens, certificates) in a secure manner through the use of Vault [1].

DECIDE DevOps framework will follow an incremental strategy, according to which different prototypes of the framework are periodically released (in months 15, 27 and 33). The current M27 prototype improves upon the M15 version and has the following coverage of the expected functionalities:

1. *Entry point.* **Covered.** This prototype provides a platform with centralized access to all DECIDE tools.
2. *KR integration.* **Covered.** The prototype gives access to all DECIDE KRs and enables communication amongst them.
3. *Workflow orchestration.* **Partially covered.** The DevOps framework provides the means to launch the corresponding tool, and automatically triggers some components. Full workflow orchestration will be supported on the final version.
4. *Application configuration.* **Covered.** The prototype lets users create and configure applications, by letting them introduce all the necessary information about them either from the General editor or from the corresponding tab of the tool.
5. *User and application management.* **Covered.** The prototype provides infrastructure to manage user access and the application(s) that each user is working on.
6. *Secrets management.* **Covered.** The prototype provides access to Vault, a component that safely stores sensitive information and enables its secure sharing.

#### Requirements:

The global requirements for the DECIDE DevOps Framework have been analyzed, reviewed and gathered in D2.1 [3] and revised in D2.2 [4]. The following table provides the status of the implementation of these requirements in the M27 prototype. The table represents an update on the requirements implemented for the M15 release and documented in section 2.1 of deliverable D2.6 [7].

**Table 2.** Requirements covered by the M27 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 provides 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 provides 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 to give an overview of the application status
KR1-REQ4	The system must receive ARCHITECT's patterns.	Although the prototype does not receive patterns as such due to design reasons, it provides access to the patterns repository and allows a user to select what patterns will be applied to the application.
KR1-REQ5	The developer must have access to a development environment with the received patterns.	Requirement rejected. ARCHITECT's patterns do not include code snippets that can be received by a development environment.
KR1-REQ6	The developer must have access to a development environment with preloaded DECIDE configurations.	The prototype allows its users to import Application Description files, which would load a certain DECIDE configuration.
KR1-REQ7	The system must allow the developer to submit their code.	This functionality is provided by Eclipse.
KR1-REQ8	The system must be able to version the code submitted by the developer.	This functionality is provided by Git.
KR1-REQ9	The system must be able to resolve the dependencies of the submitted code.	This functionality is provided by Eclipse/Git.
KR1-REQ10	The system must compile the code without errors.	This functionality is provided by Jenkins.
KR1-REQ11	The system must receive the testing activities that have to be performed on the code.	This functionality is provided by SonarQube.
KR1-REQ12	The system must be able to perform the received testing activities.	This functionality is provided by SonarQube.
KR1-REQ13	The system must present the results from the testing activities.	This functionality is provided by SonarQube.

Req. ID	Req. Description	Requirement coverage by the prototype
KR1-REQ14	The system must guarantee the continuity of the code within DECIDE's workflow.	The code resides 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.	Requirement rejected.
KR1-REQ17	DECIDE DevOps framework must provide support for NFR gathering.	The prototype provides a General Editor that will let the user specify the application's NFRs.
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 provides a General Editor that will let the user specify application's NFPs.
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 provides 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.	The MCSLA editor is integrated in the prototype.
KR1-REQ21	The system must include an Application Controller.	The prototype utilizes the Application Controller to update the Application Description file.
DEVOPS-REQ1	DECIDE framework must facilitate small and frequent updates of the code.	The prototype provides continuous integration, which facilitates small and frequent updates of the code.
DEVOPS-REQ2	DECIDE framework must support the automatic deployment of the infrastructure required for the development.	Requirement rejected. Development is performed locally, there is no need to deploy a development environment.
DEVOPS-REQ4	DECIDE framework must use microservices.	The prototype is built following a microservices architecture.
DEVOPS-REQ5	DECIDE framework must support the continuous integration of the developed apps.	The prototype supports 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.



Req. ID	Req. Description	Requirement coverage by the prototype
KR1-REQ22	DECIDE framework must provide a way to securely share sensitive information amongst the different Key Results	The prototype integrates Vault, a component for securely sharing and storing secrets.
KR1-REQ23	DECIDE framework must provide a way to manage its users and the projects that these users can access	The prototype provides user and application management.

### 6.1.1 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 means of the Application Controller.

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

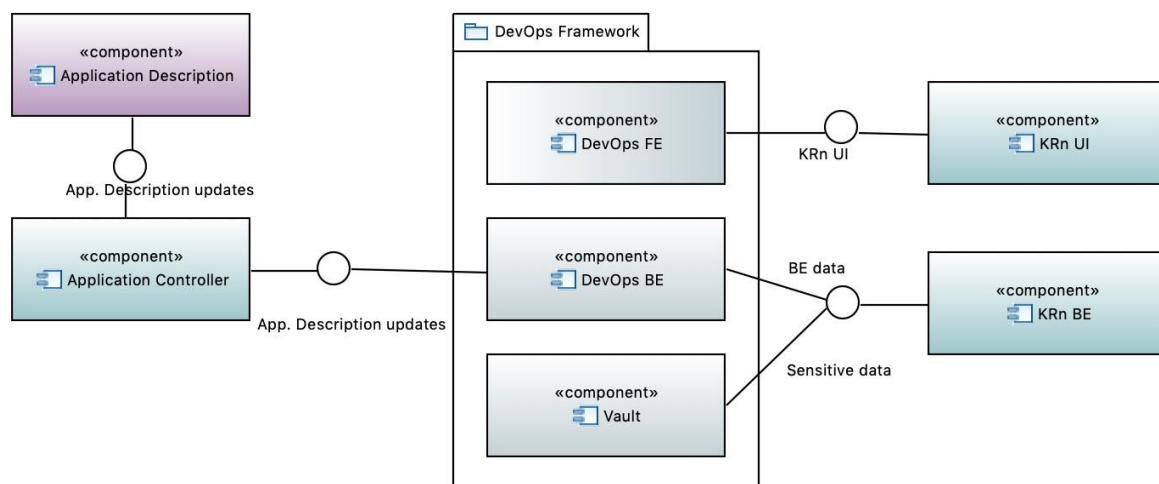


Figure 17. DevOps Framework within DECIDE

The DevOps Framework is composed of a backend, responsible for storing and manipulating data, a frontend that, on one hand, unifies the UIs of the different tools and, on the other, provides a Dashboard to give an overview of the status of the application.

Besides, a Vault instance is deployed within the DevOps Framework to handle storage and sharing of sensitive information. The DECIDE KRs will access this component when they require any secret.

## 6.2 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.

### 6.2.1 Prototype architecture

The DevOps Framework is designed as 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 intermediate 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, or load balancing to control traffic inside the containers network.

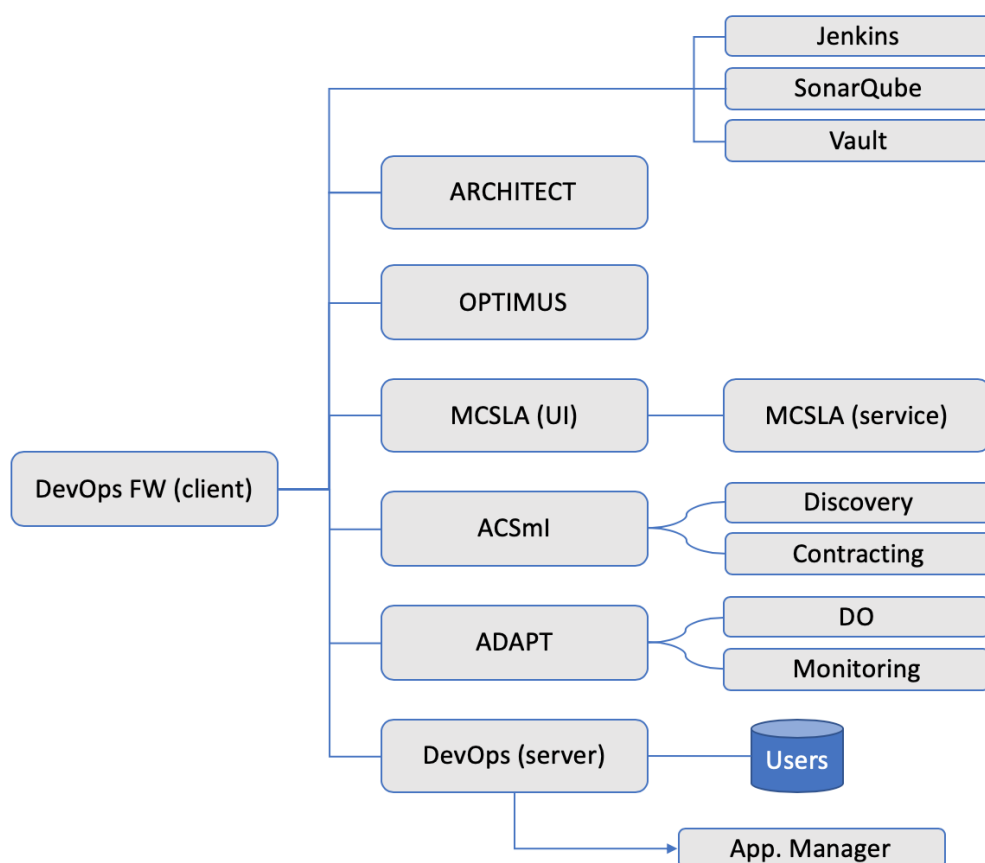


Figure 18. DevOps Framework prototype's architecture diagram

The DevOps Framework interacts with the microservices that correspond with the DECIDE KRs (ARCHITECT, OPTIMUS, MCSLA, ACSml and ADAPT). It also deploys instances of SonarQube and Jenkins, and Vault, for secrets sharing.

In addition, the framework communicates with a local database to store data relative to user access and application management. The details of this process, along with the Vault system, will be detailed in section 6.2.2.

Regarding the isolation of each microservice, and as mentioned in deliverable D2.6 [5], the DevOps platform has been deployed using Docker technology, 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.

## 6.2.2 Components description

This section aims at describing the detail of the DevOps Framework's components. The implementation of most of them has not changed since the first prototype of the framework and has already been described in deliverable D2.6 [5], so this section will only analyse those components that have been added for this release: Vault and the user and application management component.

### 6.2.2.1 User and application management

The DevOps Framework must keep track of the relationship between projects and owners of said projects. To achieve that, it keeps a local database with the following information:

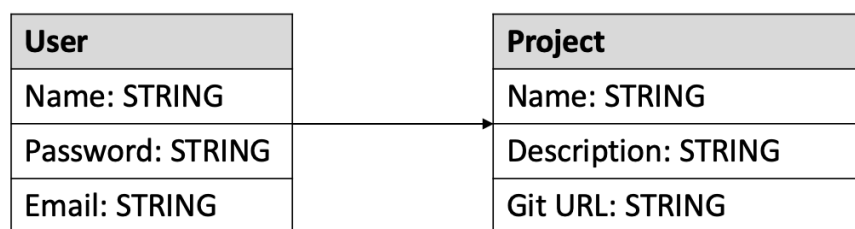


Figure 19. Schema of the database for user management

- *User Name, Password* and *Email* are requested during user registration on the DECIDE platform. The email address of the user will be used to notify them of violations.
- *Project Name, Description* and *Git URL* information are requested when a project (DECIDE application) is created. Git URL is stored here to be able to provide it to the tools whenever they are called. An alternative approach, in which the Git URL is stored in Vault is being considered as well.

### 6.2.2.2 Vault

Within DECIDE, a series of considerations have to be taken into account in order to use Vault:

- Tools are authenticated in the Vault DB through a token provided by the DevOps FW.
- This token will be provided as an environment variable at deployment time. Tools can access Vault with `VAULT_TOKEN`.
- When a project is created, the DevOps FW will include the Git token (introduced in the Wizard) in the Vault DB.
- Git tokens are stored in Vault in the following path:

`secrets/username/app_N`, with `app_N` being the project stored in the Git that is accessed with the token.

Within that path, Git tokens are stored as a K/V pair, with the format:

`key=git_token`

- Then, when a tool is called, the call will contain, besides the Git URL, the username and the application being worked on. With that information, the tool will request the Git Token from Vault. The following figure illustrates this process.

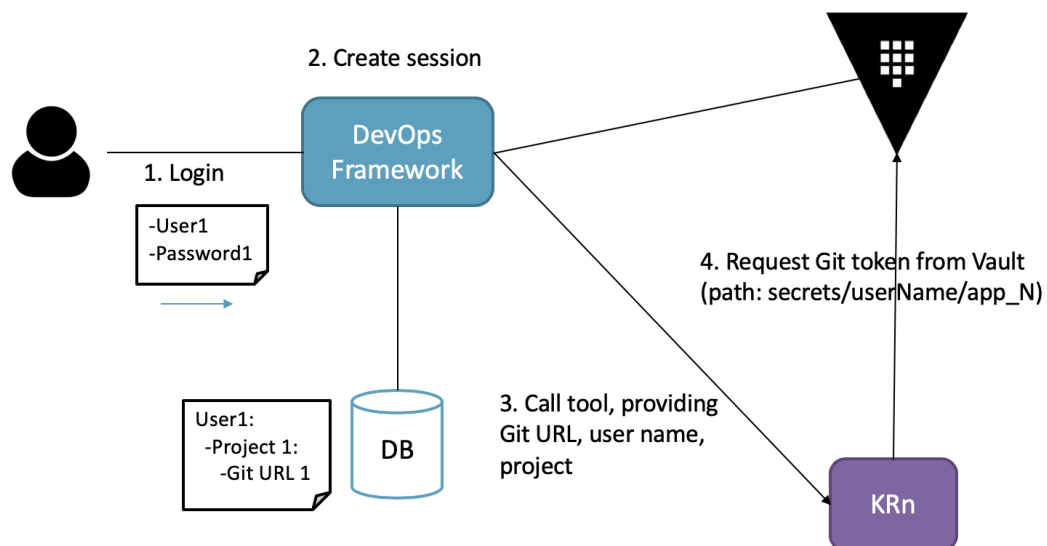


Figure 20. Vault in DECIDE

## USAGE

Vault is deployed in the integration environment. It can be accessed on:

<http://85.41.90.245:8200>

All secrets are stored under the *secrets/* path, with the convention mentioned before:

*secrets/username/app\_N*

In order to access Vault, a token is required. This token will be provided by the DevOps framework. A policy is in place, limiting access for the tools to read-only.

Components can interact with Vault via its API. To obtain a certain secret, the following call must be sent:

```
curl -H "X-Vault-Token: $VAULT_TOKEN" -X GET
http://85.91.40.245:8200/v1/secret/\[username\]/\[app N\]
```

This call will return a JSON file containing the secret:

```
{
  "request_id": "ac6beaef-200a-a9a3-fe20-cc161cc3c7e9",
  "lease_id": "",
  "renewable": false,
  "lease_duration": 2764800,
  "data": {
    "key": "123"
  },
  "wrap_info": null,
  "warnings": null,
  "auth": null
}
```

```
}
```

In the prior example, the key is “123”.

Vault provides a graphical UI, that can be useful for operators to check the status of the vault. It can be accessed here: <http://85.91.40.245:8200/ui/vault/auth?with=token> .

## 7 Delivery and usage

### 7.1 Package information

This section will briefly detail the architecture of each component. Since there are a lot of files involved, only the most representative ones will be explained to provide a better understanding of the DevOps Framework architecture.

#### 7.1.1 DevOps Framework Client

Here it is contained the front-end code developed in Angular 6. It provides the interface with which users can interact, and the communication with the back-end application.

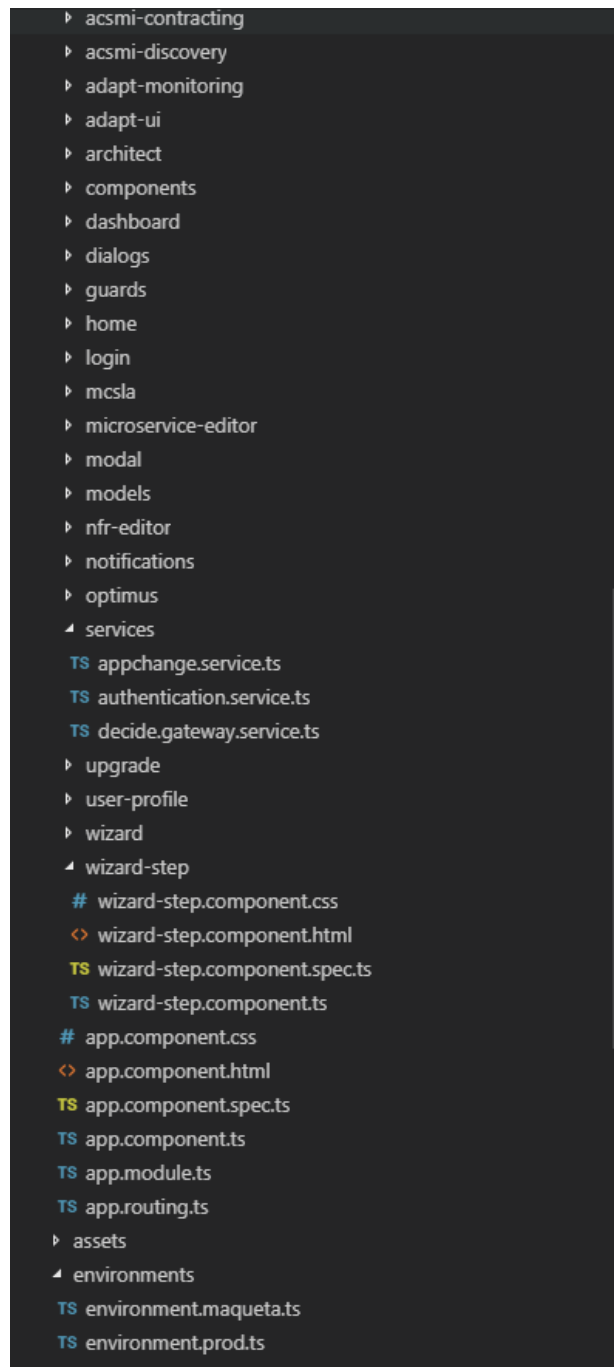


Figure 21. DevOps Framework Client's file structure

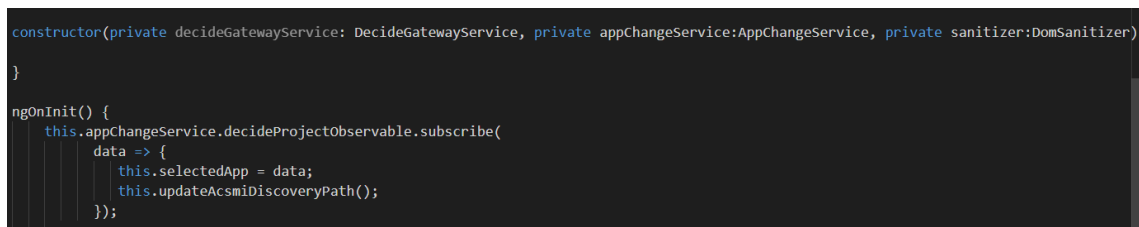
## Acsmi-contracting, Acsmi-discovery, Adapt Monitoring, Adapt UI, MCSLA, Architect, and optimus

These modules request the correct URL of the service from .ts file and through the .html file shows an iframe or a table to show the data to the user in front-end. The following figures show two sample code excerpts of these components:



```
<div class="main-content">
  <div class="">
    <div class="">
      <div class="col-md-12">
        <div class="">
          <h3>{{selectedApp.name}}</h3>
          <div class="card-content">
            <div class="iframe-container hidden-sm hidden-xs">
              <iframe [src]='getAcsmiContractingPath()' style="display:block; width:100%; height:80vh;">
                <p>Your browser does not support iframes.</p>
              </iframe>
            </div>
            <div class="col-md-6 hidden-lg hidden-md text-center">
              <h5>The icons are visible on Desktop mode inside an iframe. Since the iframe is not working on Mobile and Tablets
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Figure 22. HTML code of ACSml Contracting module



```
constructor(private decideGatewayService: DecideGatewayService, private appChangeService: AppChangeService, private sanitizer: DomSanitizer) {
}

ngOnInit() {
  this.appChangeService.decideProjectObservable.subscribe(
    data => {
      this.selectedApp = data;
      this.updateAcsmiDiscoveryPath();
    });
}
```

Figure 23. TS code of ACSml Discovery module

## Components, Dialogs, Modal, Wizard, Wizard-step, Notifications

These modules contain “auxiliary” components to import in the application, such as footer or navbar.



Figure 24. Structure of the “components” module

### Guards

This component contains the file auth.guards.ts to manage the permissions. A code sample of this component can be found in the Annex ([Guards](#)).

### Login

Provides the view and the logic to implement the login view. A code sample of this component can be found in the Annex ([Login](#)).

### Microservice-editor and nfr-editor

Both components implement the form to insert new microservices, or new NFRs during the creation of the project. A code sample of the NFR editor can be found in the Annex ([NFR Editor HTML](#)).

## Models

This module contains the data models. Below, the structure of this module is shown, as well as a code snippet.

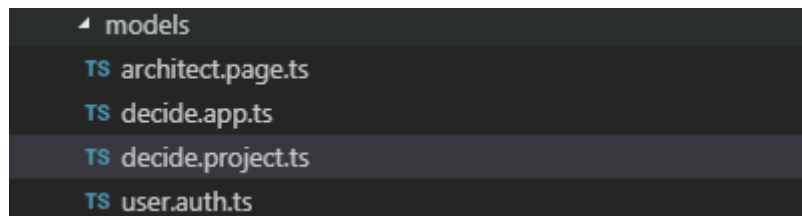


Figure 25. Structure of the “models” module

```
export class DecideProject {  
  //General project parameters  
  name:string;  
  description:string;  
  gitRef:string;  
  token:string;  
  
  //Application Description  
  highTechnologicalRisk:boolean;  
  schemaVersion:string;  
  microservices:Microservice[];  
  nfrs: NFR[];  
}  
  
export class Microservice{  
  name:string;  
  type:string;  
  stateful:Boolean;  
  publicIp:Boolean;  
  programmingLanguage:string;  
  tags: any[];  
}  
  
export class NFR{  
  kind:string;  
  tags: any[];  
}
```

Figure 26. Code snippet of the “models” module

## Services

This module contains the services that enables the communication of the DevOps Framework server with the following services:



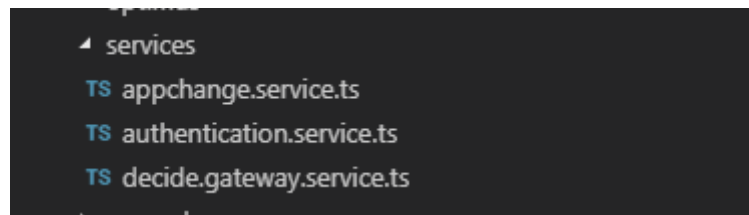


Figure 27. Structure of the “services” module

The most important service is *decide.gateway.service.ts*, which provides the communication of the backend with the external manager. A code sample of this component can be found in the Annex (*Services (decide.gateway.service.ts)*).

### 7.1.2 DevOps Framework Server

This component contains the back-end code developed in Java (Spring framework).

It receives the requests from the front-end side and manages the communication between the web application and the Application Manager Service, and with the external services.

Below, the structure of this component is shown:

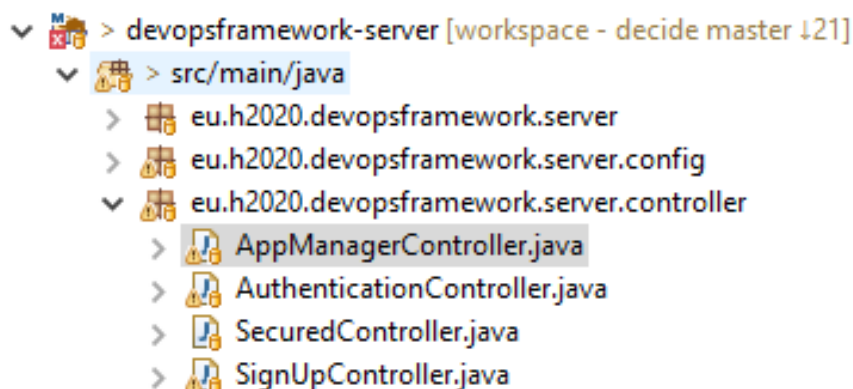


Figure 28. DevOps Framework Server's file structure

The most relevant component is the *AppManagerController*, which handles the requests to the Application Manager to update the Application Description. A code sample of this component can be found in the Annex (*DevOps Framework Server (appManagerController)*).

## 7.2 Installation instructions

This section refers to the instructions that would have to be followed if it were desired to install a local instance of the DevOps Framework. However, the best way to access it at this point would be to access the instance that is deployed in an AIMES' machine, and accessible here:

<http://85.91.40.245:8084/decide/>

More details about this deployment can be found in section 7.3.

To deploy the different containers, a *docker compose* configuration file has been created, 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 several 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 configured and accessible from your browser.
5. Access to the main DevOps framework web page in <http://localhost:4000> in your local machine browser.

## 7.3 User Manual

As mentioned above, there is a deployment of the DevOps Framework available on <http://85.91.40.245:8084/decide/> with the DECIDE KRs integrated.

The following table shows the endpoints where each DECIDE component is deployed:

**Table 3.** Endpoints of DECIDE components

Component	Deployment port
<b>ADAPT</b>	
ADAPT DO	8081
ADAPT monitoring	8088
VH	8095
<b>MCSLA</b>	
MCSLA service	8082
MCSLA ui	8083
Cloud Compendium	8001
<b>AppController</b>	
AppController	<i>Not required</i>
<b>OPTIMUS</b>	
OPTIMUS server	8090
<b>ARCHITECT</b>	
ARCHITECT server	8001
<b>ACSml</b>	
ACSml discovery registry	-
ACSml discovery server	8087
ACSml discovery client	8087
ACSml contracting	8089

DevOps Framework	
Devops FW server	8000/devopsframework-server
Devops FW client	8084
Other components	
Jenkins	8091
SQ	8092
Grafana	8093
Shockshop UI	8079
Nginx GATEWAY	8000

## 7.4 Licensing information

This component is offered under the MIT license.

## 7.5 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](https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/master/DevOpsFramework)

## 8 Conclusions

This document has presented the second prototype of the DevOps framework, corresponding to the M27 release. The new implemented functionalities have been described, as well as the integration environment that has been set up for the deployment of the DevOps Framework and the rest of the DECIDE Key Results.

The document also contains a description of 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 (D2.8) will document the third version of the DevOps framework and will be delivered by M33.

## References

- [1] Hashicorp, "Introduction to Vault," 2019. [Online]. Available: <https://www.vaultproject.io/docs/what-is-vault/index.html>. [Accessed 20 February 2019].
- [2] Hashicorp, "Vault documentation," 2019. [Online]. Available: <https://learn.hashicorp.com/vault/secrets-management/sm-static-secrets>. [Accessed 20 February 2019].
- [3] DECIDE Consortium, "D2.1 - Detailed requirements specification v1," 2017.
- [4] DECIDE Consortium, "D2.2 Detailed requirements specification v2," 2018.
- [5] DECIDE Consortium, "D2.6 Initial DECIDE DevOps Framework Integration," 2018.

## Annex A. Code snippets

### Guards

```
import { Injectable } from '@angular/core';
import { Router, CanActivate, CanActivateChild , ActivatedRouteSnapshot, RouterStateSnapshot } from
'@angular/router';
import { DecideGatewayService } from '../services/decide.gateway.service';

@Injectable()
export class AuthGuard implements CanActivate, CanActivateChild {
  constructor(private router: Router, private decideGatewayService: DecideGatewayService) { }
  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    if (localStorage.getItem('currentUser') != null) {
      // logged in so return true
      return true;
    }
    console.log("Not Logged In");
    // not logged in so redirect to login page with the return url
    this.router.navigate(['/login'], { queryParams: { returnUrl: state.url } });
    return false;
  }
  canActivateChild(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    console.log('checking child route access');
    if (this.decideGatewayService.loggedIn) {
      // logged in so return true
      return true;
    }
    console.log("Not Logged In");
    // not logged in so redirect to login page with the return url
    this.router.navigate(['/login']);
    return false;
  }
}
```

### Login

```
login() {
  this.decideGatewayService.login(this.model.username, this.model.password)
    .subscribe(
```

```

        data => {
            if(data){
                this.router.navigate(['/dashboard']);
            },
        },
        error => {
            //this.alertService.error(error);
            this.decideGatewayService.loggedIn = false;
            this.error = "Error authenticating"
        });
    }
}

```

## NFR Editor HTML

```

<div class="container col-md-12" [formGroup]='microserviceForm'>
    <div class="form-group">
        <h6 for="msNameInput">Name</h6>
        <input type="text" formControlName="name" class="form-control" id="msNameInput"
name="msNameInput" placeholder="Enter DECIDE app name">
    </div>
    <!--<div class="form-group">
        <h6 for="msTypeInput">Type</h6>
        <select formControlName="type" class="form-control form-control-sm" id="msTypeInput">
            <option value="" selected disabled hidden>Select the microservice type</option>
            <option>Database</option>
            <option>Computing</option>
            <option>Storage</option>
        </select>
    </div-->
    <div class="form-group">
        <h6 for="msProgrammingLanguageInput">Programming Language</h6>
        <input type="link" formControlName="programmingLanguage" class="form-control"
id="msProgrammingLanguageInput" name="msProgrammingLanguageInput" placeholder="Enter microservice
programming language">
    </div>
    <div class="form-group form-check">
        <div class="col-md-2">
            <input class="form-check-input" formControlName="stateful" type="checkbox" value="stateful"
id="stateful">
            <label class="form-check-label" for="stateful">Stateful</label>

```

```

    </div>
    <div class="col-md-2">
        <input class="form-check-input" formControlName="publicIp" type="checkbox" value="publicIp"
id="publicIp">
        <label class="form-check-label" for="publicIp">Public IP</label>
    </div>
</div>
<!-- TAGS -->
<div class="card container form-group">
    <h6>Tags</h6>
    <div class="input-group">
        <input class="form-control" #tagbox type="text" required>
        <span class="input-group-btn">
            <button class="btn btn-primary" (click)="addTag(tagbox.value);tagbox.value=''>Add</button>
        </span>
    </div>
    <div class="tag-container">
        <div *ngFor='let tag of microserviceForm.controls.tags.controls; let i = index'>
            <div class="tag-label">
                <span class="glyphicon glyphicon-tag tag-icon"></span>
                <span> {{ tag.value }} </span>
                <div class="separator"></div>
                <span
                    style="cursor:pointer"
                    class="glyphicon
                    glyphicon-remove"
(click)='clearTag(i)'></span>
            </div>
        </div>
    </div>
</div>
</div>

```

## Services (decide.gateway.service.ts)

```

@Injectable()
export class DecideGatewayService {
    apiUrl = environment.gateway;
    apiArchitect = environment.apiArchitect;
    private token : string;

    private headers = new Headers({'Content-Type':'application/json'});

```



```
    loggedIn : boolean;

    //Jenkins credentials
    jenkinsUsername= environment.jenkinsUsername;
    jenkinsToken= environment.jenkinsToken;

    constructor(private http: Http) {
    }

    createAuthorizationHeader(headers: Headers) {
        headers.append('Authorization', 'Basic ' +
            this.token);
    }
    /*
    * API CALLS
    */

    //AUTH
    login(username: string, password: string): Observable<boolean>{
        this.loggedIn = false;
        console.log(JSON.stringify({username: username, password: password}));
        return this.http.post(this.apiUrl + "/api/auth", JSON.stringify({username: username, password:
password}), {headers: this.headers})
            .map((response: Response) => {
                // login successful if there's a jwt token in the response
                let token = response.json() && response.json().token;
                console.log(response)
                if (token) {
                    // set token property
                    this.loggedIn = true;
                    this.token = token;

                    // store username and jwt token in local storage to keep user logged in between page
refreshes

                    localStorage.setItem('currentUser', JSON.stringify({ username: username, token:
this.token }));

                    // return true to indicate successful login
```

```
        return true;
    }else {
        // return false to indicate failed login
        this.loggedIn = false;
        return false;
    }
})
.catch((error:any) => Observable.throw(error.json().error || 'Server error'));
}

getToken() : string{
    var currentUser = JSON.parse(localStorage.getItem('currentUser'));
    this.token = currentUser && currentUser.token;
    if(this.token != null){
        return this.token;
    }
    return "";
}

logout(): void {
    // clear token remove user from local storage to log user out
    localStorage.removeItem('currentUser');
    this.loggedIn = false;
}

//DECIDE APPS CRUD
getDecideApps(): Observable<DecideApp[]>{
    let headers = new Headers ({'Content-Type':'application/json','x-auth-token': this.getToken()});
    console.log(headers)
    return this.http.get(this.apiUrl + "/getDecideProjects", {headers: headers})
        .map((res: Response) => res.json())
        .catch((error: any) => Observable.throw('Server error'));
}

createDecideApp(decideApp:DecideApp): Observable<boolean>{
    let headers = new Headers
    headers.append("x-auth-token",this.getToken());
    headers.append("Content-Type","application/json");
```

```
//console.log(this.getToken())
return this.http.post(this.apiUrl + "/createDecideProject", decideApp, {headers: headers})
    .map((res: Response) => true)
    .catch((error: any) => Observable.throw('Server error'));
}

/*****
* APP DESC RETRIEVAL - DevOps Framework - Server
*****/

getApplicationDescription(decideProject:DecideProject): Observable<String[]> {
    let headers = new Headers
    headers.append("x-auth-token",this.getToken());
    headers.append("Accept","application/json");
    return this.http.post(this.apiUrl + "/getApplicationDescription",decideProject,{headers: headers})
        .map((res: Response) => res.json())
        .catch((error: any) => Observable.throw('Server error'));
}

getNFRs(decideProject:DecideProject): Observable<String[]> {
    let headers = new Headers
    headers.append("x-auth-token",this.getToken());
    headers.append("Accept","application/json");
    return this.http.post(this.apiUrl + "/nfrs",decideProject,{headers: headers})
        .map((res: Response) => res.json())
        .catch((error: any) => Observable.throw('Server error'));
}

getMicroservices(decideProject:DecideProject): Observable<Microservice[]> {
    let headers = new Headers
    headers.append("x-auth-token",this.getToken());
    headers.append("Accept","application/json");
    return this.http.post(this.apiUrl + "/microservices",decideProject,{headers: headers})
        .map((res: Response) => res.json())
        .catch((error: any) => Observable.throw('Server error'));
}

getRecommendedPatterns(decideProject:DecideProject): Observable<String[]>{
    let headers = new Headers
```

```

        headers.append("x-auth-token",this.getToken());
        headers.append("Accept","application/json");
        return this.http.post(this.apiUrl + "/patterns",decideProject,{headers: headers})
            .map((res: Response) => res.json())
            .catch((error: any) => Observable.throw('Server error'));
    }

    getSimulationSchema(decideProject:DecideProject): Observable<String[]>{
        let headers = new Headers
        headers.append("x-auth-token",this.getToken());
        headers.append("Accept","application/json");
        return this.http.post(this.apiUrl + "/schema",decideProject,{headers: headers})
            .map((res: Response) => res.json())
            .catch((error: any) => Observable.throw(error));
    }

    /*****
    * ARCHITECT
    *****/

    recommendPatterns(nfr: string): Observable<NFR[]> {
        let headers = new Headers
        return this.http.get(this.apiArchitect + "/inferred/patterns",{params:{'nfr':nfr},headers: headers})
            .map((res: Response) => res.json())
            .catch((error: any) => Observable.throw('Server error'));
    }

    /*****
    * OPTIMUS
    *****/

    getAllSimulations(): Observable<String> {
        let headers = new Headers
        headers.append("Accept","application/json");
        return this.http.get(environment.apiUrlOptimus + "/optimussimulation/applications",
{headers:headers})
            .map((res: Response) => res.json())
            .catch((error: any) => Observable.throw('Server error'));
    }

    /*****
    * JENKINS

```

```

*****/

pingService(): Observable<String> {
    return this.http.get(this.apiUrl + "/")
        .map((res: Response) => res.json())
        .catch((error: any) => Observable.throw(error.json().error || 'Server error'));
}

jenkinsSockshopQuery =
"/api/json?tree=jobs[name,url,healthReport[score],state,lastBuild[timestamp,result,duration,number,url],lastSuccessfulBuild[timestamp,result,duration,number,url],lastFailedBuild[timestamp,result,duration,number,url], builds[number,actions[parameters[name,value]]]&pretty=true"

getJobs(): Observable<String> {
    let headers = new Headers();
    headers.append("Authorization", "Basic " + btoa(this.jenkinsUsername + ":" + this.jenkinsToken));
    headers.append("Content-Type", "application/x-www-form-urlencoded");

    return this.http.get(environment.apiUrlJenkins + this.jenkinsSockshopQuery, {headers:headers})
        .map((res: Response) => res.json())
        .catch((error: any) => Observable.throw(error.json().error || 'Server error'));
}

buildJob(name:string): Observable<String>{
    let headers = new Headers();
    headers.append("Authorization", "Basic " + btoa(this.jenkinsUsername + ":" + this.jenkinsToken));
    headers.append("Content-Type", "application/x-www-form-urlencoded");

    return this.http.post(environment.apiUrlJenkins + "/job/" + name + "/build", {}, {headers:headers})
        .map((res: Response) => res.text())
        .catch((error: any) => Observable.throw(error.json().error || 'Server error'));
}

/*****
* SonarQube
*****/

sqToken = "75c6a2cd3747c23ba113499e0065f0b1da7cafaa";

getSonarProjects(): Observable<String>{
    let headers = new Headers();
    //headers.append("Authorization", "Basic " + btoa("admin" + ":" + "decide-admin"));
    //headers.append("Content-Type", "application/x-www-form-urlencoded");

```

```

        return this.http.get(environment.apiSonarQube + "/api/components/search?qualifiers=TRK",
{headers:headers})

        .map((res: Response) => res.json())

        .catch((error: any) => Observable.throw(error.json().error || 'Server error'));

    }

    getSonarMetrics(projectId:string): Observable<String>{
        let headers = new Headers();

        return this.http.get(environment.apiSonarQube + "/api/measures/component?component=" + projectId
+"&metricKeys=bugs,vulnerabilities,code_smells,violations,coverage", {headers:headers})

        .map((res: Response) => res.json())

        .catch((error: any) => Observable.throw(error.json().error || 'Server error'));

    }

    /*****
    * ACSmI Discovery
    *****/

    getMonitoringApplications(){
        let headers = new Headers();
        headers.append("Content-Type", "application/json");
        headers.append("Accept", "application/json");

        return this.http.get(environment.apiAdaptMonitoring + "/monitoringmanager/api/applications ",
{headers:headers})

        .map((res: Response) => res.json())

        .catch((error: any) => Observable.throw(error.json().error || 'Server error'));

    }

    /*****
    * ACSmI Discovery
    *****/

    getCSFromIds(ids:string){
        let headers = new Headers();
this.sqToken));

        headers.append("Content-Type", "application/json");
        headers.append("Accept", "application/json");

```

```

        return this.http.get(environment.apiAcsmiDiscovery
"/acsmiservices/api/services/optimus?serviceids=" + ids, {headers:headers})
        .map((res: Response) => res.json())
        .catch((error: any) => Observable.throw(error.json().error || 'Server error'));
    }
}
/*****
 * ACSmI Contracting
 *****/
getAcsmiContractingEndpoint(): Observable<String>{
    let headers = new Headers();
    headers.append("Access-Control-Allow-Origin", "*");
    headers.append("Content-Type", "application/json");

    return this.http.post(environment.apiAcsmiContracting + "/decide/acsmi/contracting/api/v1/sessions"
, {} , {headers:headers})
        .map((res: Response) => res.json())
        .catch((error: any) => Observable.throw(error.json().error || 'Server error'));
    }
}

```

## DevOps Framework Server (appManagerController)

```

@RestController
public class AppManagerController {

    @Autowired
    BasicUserService userService;

    private static final String REPOSITORY_PATH = "src/main/resources/decide-projects/";

    //Commit messages
    private static final String COMMIT_MSG_CREATE_APP_DESC = "Initialized DECIDE app name and description";

    //Errors
    private static final String ERROR_MSG_PROJECT_NOT_CREATED = "Error while creating the new DECIDE project.";
    private static final String ERROR_MSG_PROJECT_NOT_FOUND = "Error - The requested DECIDE project doesn't exists.";

    @RequestMapping(value = "/createDecideProject", method = RequestMethod.POST)

```

```

    public String createDecideProject(@RequestBody Map<String, Object> payload)
    throws IOException, AppManagerException {
        try {
            String gitRef = (String) payload.get("gitRef");
            String token = (String) payload.get("token");

            JSONObject json= new JSONObject(payload);

            //Create id
            json.remove("gitRef");
            json.remove("token");

            //Local repositories config
            String name = (String) payload.get("name");
            String description = (String) payload.get("description");
            String appDescName = "DECIDE.json";
            String repoPath= REPOSITORY_PATH + name;

            //Creation of the repository folder

            //Create Application Description file (DECIDE.json) to the new
            app_repo
            AppDescription appDescription =
            AppDescriptionFactory.fromJson(json.toString());
            System.out.println("Creating DECIDE project: " + name);
            System.out.println("Appdesc: " + json.toString());

            //Initialize the Git repo. Create folder
            System.out.println("gitRef: " + gitRef + "token: " + token +
            "repoPath: " + repoPath);

            AppManager manager = openAppManager(gitRef, token ,
            FileSystems.getDefault().getPath(repoPath));
            boolean success = true;

            if(success) {
                //Write AppDescription and close AppManager
                manager.writeAndSync(appDescription,
                COMMIT_MSG_CREATE_APP_DESC);

                //DEVOPS FRAMEWORK - MONGODB
                //Add Decide project to logged user
                System.out.println("CURRENT: " +
                userService.getCurrent().getName());
                User loggedUser =
                userService.findByUsername(userService.getCurrent().getName());

                //TODO: Switch token or username;
                loggedUser.addProject(new
                DecideProject(appDescription.getId(), gitRef, token, appDescription.getName()));
                userService.update(loggedUser.getId(), loggedUser);

                manager.sync();
                manager.close();
                return "{\"success\": \"\" + 1 + "\"}";
            }
            else {
                System.out.println("error");
                return "{\"error\": \"\" + ERROR_MSG_PROJECT_NOT_CREATED +
            "\"}";

```



```

    }
    }catch(Exception e) {
        e.printStackTrace();
        return "{\"error\":\"\" + e.getMessage() + "\"}";
    }
}

@RequestMapping(value = "/getApplicationDescription/{name}", method =
RequestMethod.GET)
public String getApplicationDescription(@PathVariable("name") String name)
throws IOException, AppManagerException {

    DecideProject selectedProject = null;
    String repoPath= REPOSITORY_PATH + name;
    System.out.println(repoPath);

    //Get project info from MongoDB
    User loggedUser
userService.findByUsername(userService.getCurrent().getName());
    for(DecideProject project: loggedUser.getProjects()) {
        if(project.getName().equals(name)) {
            selectedProject = project;
        }
    }

    if(selectedProject!=null) {
        //Creation of the repository folder
        boolean exists = new File(repoPath).mkdirs();
        try {
            if(exists) {
                System.out.println("exists");
                //Get app desc and return it
                AppManager manager
AppManager.open(FileSystems.getDefault().getPath(repoPath));
                //TODO:Pull
                //manager.sync();
                AppDescription appDescription
manager.getAppDescription();
                System.out.println(appDescription.toString());
                manager.close();
                return new
ObjectMapper().writeValueAsString(appDescription);
            }else {
                System.out.println("exists no");

                //Fetch the repo
                //System.out.println("Not found");
                System.out.println("gitRef: " +
selectedProject.getGitRef() + "token: " + selectedProject.getToken() + "repoPath: "
+ repoPath);

                AppManager manager
openAppManager(selectedProject.getGitRef(), selectedProject.getToken(),
FileSystems.getDefault().getPath(repoPath));
                AppDescription appDescription
manager.getAppDescription();

                //TODO: Save in user database?

```

```

                                return new
ObjectMapper().writeValueAsString(appDescription);
    }
    }catch(Exception e) {
        e.printStackTrace();
        return "{\"error\":\"" + ERROR_MSG_PROJECT_NOT_FOUND +
"\}";
    }
    }
    return "{\"error\":\"" + "Project not found " + "\}";

}
/*
 * PROJECTS management
 */
@RequestMapping(value = "/getDecideProject/{name}", method =
RequestMethod.GET)
public DecideProject getDecideProject(@PathVariable String name){
    User loggedUser =
userService.findByUsername(userService.getCurrent().getName());
    for(DecideProject project: loggedUser.getProjects()) {
        if(project.getName().equals(name)) {
            return project;
        }
    }
    return null;
}

@RequestMapping(value = "/getDecideProjects", method = RequestMethod.GET)
public List<DecideProject> getDecideProjects() {
    System.out.println("getDecideProjects");
    User loggedUser =
userService.findByUsername(userService.getCurrent().getName());
    List<DecideProject> projects = new ArrayList<>();
    projects = loggedUser.getProjects();
    return projects;
}

/*
 *Look info stored in the app description
 */
//NFRs
@RequestMapping(value = "/nfrs", method = RequestMethod.POST)
public List<Nfr> getNFRs(@RequestBody DecideProject decideProject) {
    String repoPath= REPOSITORY_PATH + decideProject.getName();
    List<Nfr> nfrs = new ArrayList();
    AppManager manager;
    try {
        System.out.println("getNFRs - " + decideProject.getGitRef());
        manager = openAppManager(decideProject.getGitRef(),
decideProject.getToken(), FileSystems.getDefault().getPath(repoPath));
        //Pull
        manager.sync();
        nfrs = manager.getAppDescription().getNfrs();
        //manager.close();
    } catch (AppManagerException | IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

```

        return nfrs;
    }

    //Microservices
    @RequestMapping(value = "/microservices", method = RequestMethod.POST)
    public List<Microservice> getMicroservices(@RequestBody DecideProject
decideProject) {
        String repoPath= REPOSITORY_PATH + decideProject.getName();
        List<Microservice> microservices = new ArrayList();
        AppManager manager;
        try {
            System.out.println("getMicroservices - " +
decideProject.getGitRef());
            manager = AppManager.open(decideProject.getGitRef(),
decideProject.getToken(), FileSystems.getDefault().getPath(repoPath));
            //Pull
            manager.sync();
            microservices = manager.getAppDescription().getMicroservices();
            //manager.close();
        } catch (AppManagerException | IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return microservices;
    }

    //Patterns
    @RequestMapping(value = "/patterns", method = RequestMethod.POST)
    public List<Pattern> getPatterns(@RequestBody DecideProject decideProject) {
        String repoPath= REPOSITORY_PATH + decideProject.getName();
        List<Pattern> patterns = new ArrayList();
        AppManager manager;
        try {
            System.out.println("getPatterns - " +
decideProject.getGitRef());
            manager = openAppManager(decideProject.getGitRef(),
decideProject.getToken(), FileSystems.getDefault().getPath(repoPath));
            //Pull
            manager.sync();
            patterns =
manager.getAppDescription().getRecommendedPatterns();
            //manager.close();
        } catch (AppManagerException | IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return patterns;
    }

    //Simulation Schema
    @RequestMapping(value = "/schema", method = RequestMethod.POST)
    public List<SchemaElement> getSimulationSchema(@RequestBody
DecideProject decideProject) {
        String repoPath= REPOSITORY_PATH + decideProject.getName();
        List<SchemaElement> schemas = new ArrayList();
        AppManager manager;
        try {
            System.out.println("getSchema - " +
decideProject.getGitRef());

```

```
        manager = openAppManager(decideProject.getGitRef(),
decideProject.getToken(), FileSystems.getDefault().getPath(repoPath));
        //Pull
        manager.sync();
        schemas = manager.getAppDescription().getSchema();

        //Format the response

        //manager.close();
    } catch (AppManagerException | IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return schemas;
}

/*
 * Aux
 */
public AppManager openAppManager(String gitRef,String token, Path path)
throws AppManagerException{
    AppManager manager = AppManager.open(gitRef,token,path);
    return manager;
}
}
```