******

**Deliverable D2.8**

**Final DECIDE DevOps Framework Integration**

|  |  |
| --- | --- |
| **Editor(s):** | José Manuel López  Javier Gavilanes Ruano |
| **Responsible Partner:** | Experis IT |
| **Status-Version:** | Final – v1.0 |
| **Date:** | 24/09/2019 |
| **Distribution level (CO, PU):** | PU |

|  |  |
| --- | --- |
| **Project Number:** | GA 726755 |
| **Project Title:** | DECIDE |

|  |  |
| --- | --- |
| **Title of Deliverable:** | D2.8 Final DECIDE DevOps Framework Integration |
| **Due Date of Delivery to the EC:** | 31/08/2019 |

|  |  |
| --- | --- |
| **Workpackage responsible for the Deliverable:** | WP2 – DECIDE requirements and DECIDE solution integration |
| **Editor(s):** | Experis IT |
| **Contributor(s):** | Experis IT |
| **Reviewer(s):** | Kyriakos Stefanidis (FhG) |
| **Approved by:** | All Partners |
| **Recommended/mandatory readers:** | WP3, WP4, WP5, WP6 |

|  |  |
| --- | --- |
| **Abstract:** | This deliverable will provide the final version of the integrated DECIDE DevOps Framework. The final version  will include corrections and feedback coming from the implementation of the use cases. |
| **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 | 15.08.2019 | First draft version | Experis IT |
| V0.2 | 28.08.2019 | Added missing contributions | Experis IT |
| V0.3 | 09.09.2019 | Internal review | FhG |
| V0.5 | 10.09.2019 | Review comments addressed | Experis IT |
| V1.0 | 24.09.2019 | Ready for submission | TECNALIA |

# Table of Contents

[Table of Contents 4](#_Toc20212686)

[List of Figures 5](#_Toc20212687)

[List of Tables 5](#_Toc20212688)

[Terms and abbreviations 6](#_Toc20212689)

[Executive Summary 7](#_Toc20212690)

[1 Introduction 8](#_Toc20212691)

[1.1 About this deliverable 8](#_Toc20212692)

[1.2 Document structure 8](#_Toc20212693)

[2 DECIDE Development and Integration 9](#_Toc20212694)

[2.1 Staging integration environment: 10](#_Toc20212695)

[2.2 Production integration environment 10](#_Toc20212696)

[3 DECIDE Orchestration 19](#_Toc20212697)

[4 DECIDE UI 21](#_Toc20212698)

[5 Implementation 22](#_Toc20212699)

[5.1 Functional description 22](#_Toc20212700)

[5.1.1 Fitting into overall DECIDE Architecture 25](#_Toc20212701)

[5.2 Technical description 25](#_Toc20212702)

[5.2.1 Prototype architecture 26](#_Toc20212703)

[5.2.2 Components description 27](#_Toc20212704)

[5.2.2.1 State machine API 27](#_Toc20212705)

[5.2.2.2 UI updates 31](#_Toc20212706)

[6 Delivery and usage 33](#_Toc20212707)

[6.1 Package information 33](#_Toc20212708)

[6.1.1 DevOps Framework Client 33](#_Toc20212709)

[6.1.2 DevOps Framework Server 36](#_Toc20212710)

[6.2 Installation instructions 37](#_Toc20212711)

[6.3 User Manual 38](#_Toc20212712)

[6.4 Licensing information 39](#_Toc20212713)

[6.5 Download 39](#_Toc20212714)

[7 Conclusions 40](#_Toc20212715)

[References 41](#_Toc20212716)

[Annex A. DECIDE Components’ UI 42](#_Toc20212717)

# List of Figures

[**Figure 1**. Development process in DECIDE 9](#_Toc20212718)

[**Figure 2.** DECIDE State machine diagram 19](#_Toc20212719)

[**Figure 3.** DevOps framework’s UI. Final version 21](#_Toc20212720)

[**Figure 4.** DevOps Framework within DECIDE 25](#_Toc20212721)

[**Figure 5.** DevOps Framework prototype’s architecture diagram 26](#_Toc20212722)

[**Figure 6.** DevOps Framework UI. Connection to state machine 32](#_Toc20212723)

[**Figure 7.** DevOps Framework Client’s file structure 33](#_Toc20212724)

[**Figure 8**. HTML code of ACSmI Contracting module 34](#_Toc20212725)

[**Figure 9.** TS code of ACSmI Discovery module 34](#_Toc20212726)

[**Figure 10.** TS code of ACSmI Billing module 34](#_Toc20212727)

[**Figure 11.** Structure of the “components” module 35](#_Toc20212728)

[**Figure 12**. Structure of the “models” module 35](#_Toc20212729)

[**Figure 13**. Code snippet of the “models” module 36](#_Toc20212730)

[**Figure 14**. Structure of the “services” module 36](#_Toc20212731)

[**Figure 15**. DevOps Framework Server’s file structure 37](#_Toc20212732)

[**Figure 16**. Services of the DevOps Framework’s Server 37](#_Toc20212733)

[**Figure 17.** ARCHITECT’s UI 42](#_Toc20212734)

[**Figure 18.** OPTIMUS’ UI 42](#_Toc20212735)

[**Figure 19**. MCSLA’s UI 43](#_Toc20212736)

[**Figure 20.** ACSmI Discovery’s UI 43](#_Toc20212737)

[**Figure 21**. ACSmI Contracting’s UI 44](#_Toc20212738)

[**Figure 22**. ADAPT DO’s UI (1) 44](#_Toc20212739)

[**Figure 23.** ADAPT DO’s UI (2) 45](#_Toc20212740)

[**Figure 24.** ACSmI Monitoring’s UI 45](#_Toc20212741)

# List of Tables

[**Table 1**. Relationship between states and enabled tools 20](#_Toc20212742)

[**Table 2.** Requirements covered by the M33 prototype 22](#_Toc20212743)

[**Table 3**. Endpoints of DECIDE components 38](#_Toc20212744)

# Terms and abbreviations

|  |  |
| --- | --- |
| ACSmI | 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 third release of this component offers new functionalities, as a state machine to control the DECIDE workflow and that was introduced in the previous version of this deliverable, and new changes in the UI.

The state machine has been updated to include redeployment workflows and to iron out some issues, and its effect is reflected in the DECIDE UI, to give the user and indication of whether or not the tools are enabled.

The deliverable also goes deeper into the configuration of the production environment, detailing each of the pipelines set up to deploy the DECIDE components.

Lastly, the technical implementation of this component, as well as delivery information and installation and usage instructions can be found at the end of the document.

# Introduction

## About this deliverable

This deliverable explains the architecture, delivery and usage of the third and final DECIDE DevOps framework prototype. It details the newly implemented functionalities for the M33 prototype, such as the workflow orchestration through a state machine, and it includes a technical description of this component.

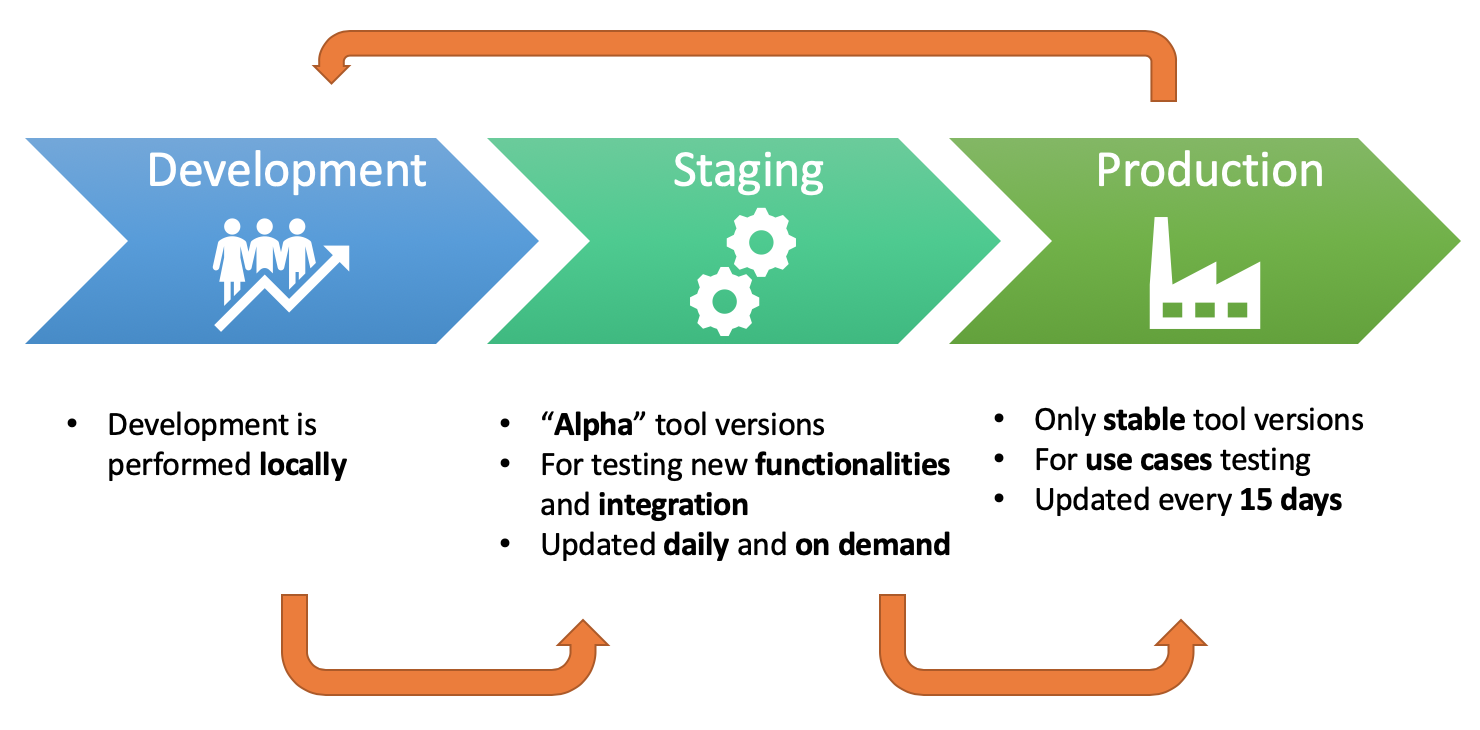
## Document structure

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

* Section 2 reviews the DECIDE integration environment.
* Section 3 details the new state machine component.
* Section 4 presents the changes to the DECIDE UI.
* Section 5 contains the technical description of the DevOps framework.
* Section 6 provides delivery and usage information.

# DECIDE Development and Integration

This section presents the production environment set up for testing the stable versions of the DECIDE tools. As it was introduced in deliverable *D2.7 Intermediate DECIDE DevOps Framework integration* [1], the development process in DECIDE is configured as a three-stage process, as shown in the following figure:



**Figure 1**. Development process in DECIDE

According to this strategy, the tools are developed locally by partners during the **development** stage, tested for correctness and proper integration during the **staging** stage, and moved to the **production** stage only when the components are working as intended.

* 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.

Besides, as explained in deliverable D2.7 [1], two different branches were be created in Git:

* ***Master*** branch: this branch hosts the most recent version of the tools, where new functionalities are implemented. The code on the *master* branch is deployed in the staging environment whenever a new version is released.
* ***Release*** branch: this branch holds only stable versions of the tools. It is 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 is deployed in the production environment every 15 days.

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

## 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. More details about the configuration of the staging environment can be found in deliverable D2.7 [1].

## Production integration environment

The goal of the production environment is to have a more static version of DECIDE, for high-level testing and for use-cases testing. This environment is set up exactly the same way as the staging one, in a second AIMES machine, only with the automatic deployment disabled and configured to redeploy tools every 15 days.

This section will describe how the pipelines to build and deploy the different tools have been set up.

**DEVOPS FRAMEWORK:**

* Get Resources from GIT - <https://git.code.tecnalia.com/decide/devops>
* Build Server:
  + Execute maven clean package command
  + Build image decideapp/devopsframework-server
* Build frontend:
  + Execute command npm install
  + Build image decideapp/devopsframework-client
* Init containers
  + Execute command docker up -d with docker-compose.yml

version: "2"

services:

nginx:

image: nginx:latest

container\_name: nginx

restart: always

ports:

- "8000:8000"

volumes:

- ./config/nginx-dev.conf:/etc/nginx/conf.d/nginx.conf

depends\_on:

- devopsframework-server

networks:

- decide

devopsframework-mongodb:

image: mongo:latest

container\_name: devopsframework-mongodb

restart: on-failure

ports:

- 27017:27017

networks:

- decide

devopsframework-vault:

image: vault:latest

container\_name: devopsframework-vault

restart: on-failure

ports:

- 8200:8200

networks:

- decide

devopsframework-server:

build: ./devopsframework-server

container\_name: devopsframework-server

restart: always

ports:

- 4000:4000

- 5005:5005

environment:

- SPRING\_PROFILES\_ACTIVE=dev

links:

- devopsframework-mongodb

depends\_on:

- devopsframework-mongodb

networks:

- decide

devopsframework-client:

build: ./devopsframework-client

container\_name: devopsframework-client

restart: always

ports:

- 81:80

volumes:

- ./devopsframework-client:/usr/src/app

environment:

- ENVIRONMENT=dev

depends\_on:

- devopsframework-server

networks:

- decide

networks:

decide:

external: true

**ARCHITECT:**

* Get Resources from GIT - <https://git.code.tecnalia.com/decide/architect-ui>
* Build:
  + Execute command npm install
  + Execute command npm custom
  + Build image localhost:5000/decideapp/architect-ui with Dockerfile

FROM nginx:1.15-alpine

MAINTAINER benjamin.dittwald@fokus.fraunhofer.de

ADD dist.tar.gz /usr/share/nginx/html/

COPY nginx.vh.default.conf /etc/nginx/conf.d/default.conf

# The following steps are needed because of the OpenShift security constraints

# Create some temp folders for later permission granting

RUN mkdir /var/cache/nginx/uwsgi\_temp && \

mkdir /var/cache/nginx/client\_temp && \

mkdir /var/cache/nginx/proxy\_temp && \

mkdir /var/cache/nginx/fastcgi\_temp && \

mkdir /var/cache/nginx/scgi\_temp && \

chmod g+rwx /var/cache/nginx /var/run /var/log/nginx /var/cache/nginx/client\_temp

EXPOSE 8080

**OPTIMUS:**

* Get Resources from GIT - <https://git.code.tecnalia.com/decide/WP3>
* Build Server:
  + Run docker command

build -f

Optimus/eu.decideh2020.int.optimus.server.src.dvp/src/main/docker/Dockerfile

--build-arg GIT\_CREDENTIALS=$(GIT\_CREDENTIALS)

--build-arg APPMANAGER\_VERSION=$(APPMANAGER\_VERSION\_OPTIMUS)

--build-arg ACSMI\_VERSION=$(ACSMI\_VERSION)

--rm --no-cache -t $(DecideRegistry)/$(Repository):$(OptimusImageTag) Optimus/eu.decideh2020.int.optimus.server.src.dvp/src/main/docker

**MCSLA:**

* Get Resources from GIT - <https://git.code.tecnalia.com/decide/mcsla-service>
* Build mcsla-service:
  + Execute maven clean package command
  + Build image decideapp/mcsla-service with Dockerfile

FROM openjdk:8-jre

ENV VERTICLE\_FILE mcsla-service-fat.jar

# Set the location of the verticles

ENV VERTICLE\_HOME /usr/verticles

EXPOSE 8080

RUN addgroup --system vertx && adduser --system --group vertx

# Copy your fat jar to the container

COPY target/$VERTICLE\_FILE $VERTICLE\_HOME/

RUN chown -R vertx $VERTICLE\_HOME

RUN chmod -R g+w $VERTICLE\_HOME

USER vertx

# Launch the verticle

WORKDIR $VERTICLE\_HOME

ENTRYPOINT ["sh", "-c"]

CMD ["exec java -Xmx2048m -jar $VERTICLE\_FILE -Dvertx.logger-delegate-factory-class-name=io.vertx.core.logging.SLF4JLogDelegateFactory -Dvertx.metrics.options.enabled=true"]

* Get Resources from GIT - <https://git.code.tecnalia.com/decide/mcsla-ui>
* Build mcsla-ui:
  + Execute command npm install
  + Execute command npm custom
  + Build image decideapp/mcsla-ui with Dockerfile

FROM httpd:alpine

ENV APACHE2\_HOME /usr/local/apache2

COPY ./build/ $APACHE2\_HOME/htdocs/

COPY ./httpd.conf $APACHE2\_HOME/conf/httpd.conf

COPY ./runtimeconfig.sh $APACHE2\_HOME/conf/runtimeconfig.sh

RUN chown -R www-data $APACHE2\_HOME

RUN chmod -R g+w $APACHE2\_HOME

RUN chmod gu+x $APACHE2\_HOME/conf/runtimeconfig.sh

RUN apk update && apk add --no-cache jq

USER www-data

EXPOSE 8080

ENTRYPOINT [ "/usr/local/apache2/conf/runtimeconfig.sh" ]

CMD ["httpd-foreground"]

**ACSmI Discovery:**

* Get Resources from GIT - <https://git.code.tecnalia.com/decide/WP5>
* Stop Containers:
  + Execute command docker stop jhipster.registry
  + Execute command docker stop acsmi.mysql
  + Execute command docker stop acsmi.frontend
  + Execute command docker stop acsmi.backend.services
  + Execute command docker stop acsmi.backend.services.test.00
* Remove Containers:
  + Execute command docker rm jhipster.registry
  + Execute command docker rm acsmi.mysql
  + Execute command docker rm acsmi.frontend
  + Execute command docker rm acsmi.backend.services
  + Execute command docker rm acsmi.backend.services.test.00
* Build server base:
  + Execute command

Build

ACSmI\_discovery/eu.decideh2020.springboot.server.repo.src.dvp/src/main/docker -t decideh2020/jhipster.repo:latest --build-arg GIT\_CREDENTIALS=$(ACSMI\_Monitoring\_Git\_Creds) --build-arg ARTIFACTORY\_USE=N --build-arg VERSION=latest

* Build registry:
  + Execute command

build

ACSmI\_discovery/eu.decideh2020.int.acsmi.registry.server.src.dvp/src/main/docker -t decideh2020/jhipster.registry:latest --build-arg GIT\_CREDENTIALS=$(ACSMI\_Monitoring\_Git\_Creds) --build-arg ARTIFACTORY\_USE=N --build-arg VERSION=latest

* Build backend:
  + Execute command

build --no-cache

ACSmI\_discovery/eu.decideh2020.int.acsmi.backend.services.server.src.dvp/src/main/docker -t decideh2020/acsmi.backend.services:latest --build-arg GIT\_CREDENTIALS=$(ACSMI\_Monitoring\_Git\_Creds) --build-arg ARTIFACTORY\_USE=N --build-arg VERSION=latest

* Build frontend:
  + Execute command

build --no-cache

ACSmI\_discovery/eu.decideh2020.int.acsmi.frontend.server.src.dvp/src/main/docker -t decideh2020/acsmi.frontend:latest --build-arg GIT\_CREDENTIALS=$(ACSMI\_Monitoring\_Git\_Creds) --build-arg ARTIFACTORY\_USE=N --build-arg VERSION=latest

* Build MySQL:
  + Execute command

build

ACSmI\_discovery/eu.decideh2020.int.acsmi.mysql.server.src.dvp/src/main/docker -t decideh2020/acsmi.mysql:latest --build-arg VERSION=latest

* Build test00:
  + Execute command

build --no-cache

ACSmI\_discovery/eu.decideh2020.int.acsmi.backend.services.server.test.00.src.dvp/src/main/docker -t decideh2020/acsmi.backend.services.test.00:latest --build-arg VERSION=latest

* Run Containers:
  + Execute command

docker run -d --name jhipster.registry -d --restart=always --network decide

decideh2020/jhipster.registry

* + Execute command

docker run -d --name acsmi.mysql -d --restart=always --env

MYSQL\_ALLOW\_EMPTY\_PASSWORD=yes --network decide decideh2020/acsmi.mysql

* + Execute command

docker run -d --name acsmi.frontend -d --restart=always -p 8087:8080

--network decide decideh2020/acsmi.frontend

* + Execute command

docker run -d --name acsmi.backend.services -d --restart=always

--network decide decideh2020/acsmi.backend.services

* + Execute command

docker run -d --name acsmi.backend.services.test.00 -d --restart=no

--env MYSQL\_DB\_USER=root --env MYSQL\_DB\_NAME=acsmi\_backend\_services\_server

--env MYSQL\_DB\_HOST=acsmi.mysql --network decide decideh2020/acsmi.backend.services.test.00

**ACSmI Contracting:**

* Get Resources from docker hub – cloudbroker/acsmi-contracting
* Create docker-compose-yml:

version: '2'

services:

contracting:

image: cloudbroker/acsmi-contracting:latest

ports:

- '8089:80'

environment:

UPDATE\_ENVS: 'true'

PLATFORM\_URL: 'https://decide-prototype.cloudbroker.com'

PLATFORM\_EMAIL: 'acsmi.contracting@scaletools.com'

PLATFORM\_PASSWORD: \*\*\*\*\*\*

DISCOVERY\_URL: 'http://XX.XX.XX.XX:XXXX'

DISCOVERY\_USERNAME: '\*\*\*\*\*'

DISCOVERY\_PASSWORD: \*\*\*\*\*

SECRET\_KEY\_BASE: '#######################'

* Execute docker compose command down
* Execute docker compose command up -d
* Execute command docker network connect decide acsmi\_contracting\_1
* Start SQLite
  + Execute command docker exec -u app acsmi\_contracting\_1 rake db:create db:migrate db:seed

**ACSmI Billing:**

* Get Resources from docker hub – cloudbroker/acsmi-billing
* Create docker-compose-yml:

version: '2'

services:

billing:

image: cloudbroker/acsmi-billing:latest

container\_name: acsmi-billing

ports:

- '8085:80'

environment:

SECRET\_KEY\_BASE: '#######################'

* Execute docker compose command down
* Execute docker compose command up -d
* Execute command docker network connect decide acsmi\_billing
* Start SQLite
  + Execute command docker exec -u app acsmi\_billing rake db:create db:migrate db:seed

**ADAPT DO:**

* Get Resources from GIT - <https://git.code.tecnalia.com/decide/Adapt-do>
* Build:
  + Build image localhost:5000/decideapp/adapt-do
* Run
  + Execute command docker run -p 8473:80 -d --name adapt-do $(DecideRegistry)/$(Repository):$(AdaptDOImageTag)
* Consul service

sudo mkdir -p /usr/local/bin

sudo unzip -o /home/javier/consul/consul.zip -d /usr/local/bin

mkdir -p /home/javier/consul/config

cat >/home/javier/consul/config/config.json << EOF

{

"datacenter": "dc-decide",

"encrypt": "s2aLWrsvz+h0w6HM/NkXRA==",

"addresses" : {

"http": "$WAN\_ADVERTISE\_ADDRESS"

}

}

EOF

nohup /usr/local/bin/consul agent -bootstrap-expect=1 -server -data-dir= /home/javier/consul/data -config-file=/home/javier/consul/config/config.json -advertise-wan=$WAN\_ADVERTISE\_ADDRESS -bind=$BIND\_ADDRESS -ui -client=$CLIENT\_ADDRESS > /home/javier/consul/nohup.out &

**ADAPT MONITORING:**

* Get Resources from GIT - <https://git.code.tecnalia.com/decide/WP5>
* Stop Containers:
  + Execute command docker stop acsmi.influxdb
  + Execute command docker stop acsmi.monitoring.management
* Remove Containers:
  + Execute command docker rm acsmi.influxdb
  + Execute command docker rm acsmi.monitoring.management
* Build influxDB:
  + Execute command

build

ACSmI\_monitoring/eu.decideh2020.int.acsmi.monitoring.influxdb.server.src.dvp/src/main/docker -t decideh2020/acsmi.influxdb:latest --build-arg VERSION=latest

* Build Management:
  + Execute command

Build

ACSmI\_monitoring/eu.decideh2020.int.acsmi.monitoring.management.server.src.dvp/src/main/docker -t decideh2020/acsmi.monitoring.management:latest --build-arg GIT\_CREDENTIALS=$(ACSMI\_Monitoring\_Git\_Creds) --build-arg ARTIFACTORY\_USE=N --build-arg VERSION=latest

* Run Containers:
  + InfluxDB execute command

docker run -d -p 8086:8087 --name acsmi.influxdb -d -e

INFLUXDB\_DB='decideh2020acsmi' --env PASSWORD=\*\*\*\*\*\*\* --network decide decideh2020/acsmi.influxdb

* + Management execute command

run -d --restart always --name acsmi.monitoring.management -d -p

$(ACSmIMonitoringPortMapping) --link acsmi.influxdb:influxdb --env USER\_GIT=$(USER\_GIT) --env TOKEN\_GIT=$(TOKEN\_GIT) --network decide decideh2020/acsmi.monitoring.management

# DECIDE Orchestration

As it was introduced in deliverable D2.7 [1], DECIDE requires a component that handles the triggering of the appropriate tool in automatic workflows and that enables or disables these tools according to the information they have received so far.

This section provides an update for the state machine proposed in the aforementioned deliverable. The final version of the state machine provides support for the redeployment workflows, for both high and low technological risk applications. It also includes the ACSmI Billing component, which had been left out in the previous version. It has been decided as well that the state machine will be a central component that is part of the DevOps Framework. The following figure shows the state machine diagram:

Imagen que contiene texto, mapa

Descripción generada automáticamente

**Figure 2.** DECIDE State machine diagram

As with the first version of the state machine, 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.

The *ADAPT DO (& Billing)* state indicates that both ADAPT DO and ACSmI Billing will be enabled when that state is reached. The justification for this is that the Billing component has to be activated after the contracts are created, but it does not “finish its execution” as the other tools: users can check billing information at any point after they have contracted the cloud services.

The *Automatic redeployment* state is reached if a violation is received for an application with low technological risk. In this state, the DevOps Framework will invoke all tools involved in a redeployment workflow (OPTIMUS, MCSLA and ADAPT DO) sequentially and without user intervention. This state will be exited after ADAPT DO has redeployed the application, moving to the *Monitoring* state to wait for new violations.

**¡Error! No se encuentra el origen de la referencia.** details the tools that are enabled in each state:

**Table 1**. Relationship between states and enabled tools

| **State** | **Enabled tools** |
| --- | --- |
| DevOps Framework | DevOps Framework |
| ARCHITECT | DevOps Framework  ARCHITECT |
| OPTIMUS | DevOps Framework  ARCHITECT  OPTIMUS |
| MSCLA | DevOps Framework  ARCHITECT  OPTIMUS  MCSLA |
| ACSmI Contracting | DevOps Framework  ARCHITECT  OPTIMUS  MCSLA  ACSmI Contracting |
| ADAPT DO (& Billing) | DevOps Framework  ARCHITECT  OPTIMUS  MCSLA  ACSmI Contracting  ACSmI Billing  ADAPT DO |
| Monitoring | DevOps Framework  ARCHITECT  OPTIMUS  MCSLA  ACSmI Contracting  ACSmI Billing  ADAPT DO  Monitoring (ADAPT MM and ACSmI Monitoring) |
| Automatic redeployment | This state does not enable any tool. The state machine component triggers the corresponding tools sequentially until the application has been redeployed. |

Transitions are the signals that move one state to the next. They are sent by each of the KRs right after they have finished their execution and written the corresponding data in the application description. That implies that the subsequent tool is ready to be used, so access to it will be enabled. This is a departure from the first approach, which relied on the variables that had been written into the application description to change states, since the current version greatly simplifies the process. The different transitions are explained below:

* **basicInfo**: the basic project’s information has been introduced from the Wizard (name, Git repository, microservices, NFRs) and has been committed to Git.
* **patterns**: patterns have been selected by the user from the list provided by ARCHITECT
* **schema**: one of the schemas proposed by OPTIMUS has been selected by the user
* **MCSLA**: the MCSLA has been created
* **contracts**: the contracts have been created
* **deployment**: the microservices have been deployed by ADAPT
* **violation**: a violation has been received

# DECIDE UI

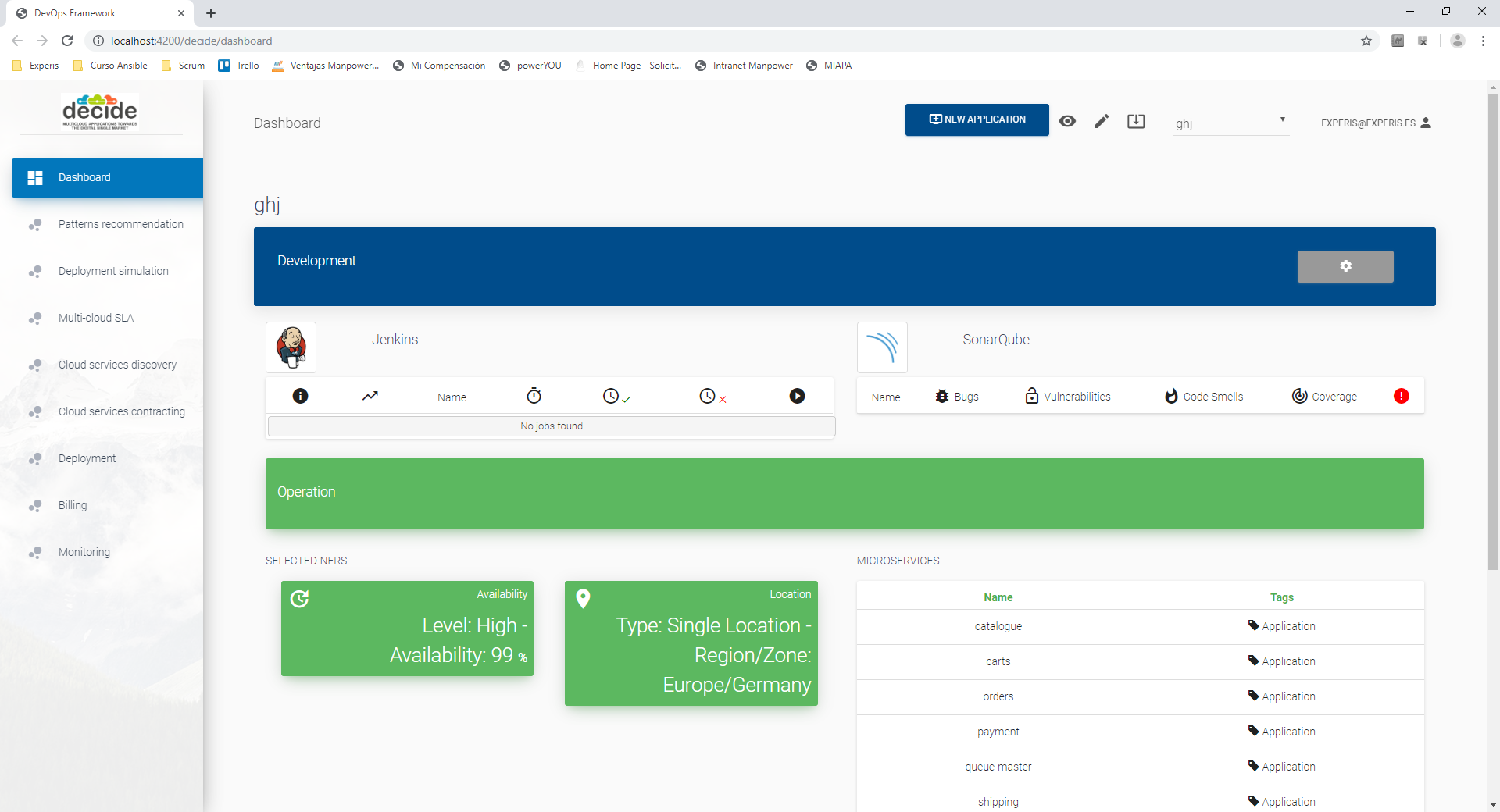
The DECIDE Framework provides a graphical user interface to provide access to the different DECIDE tools (KRs).

This graphical interface allows a user to create a DECIDE application, introducing 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 includes a dashboard, that gives an overview of the status of the DECIDE tools. Besides, the DevOps Framework integrates the UIs of the different tools, via iframe or API. More information about this can be found in Section 3 of deliverable D2.2 [2].

Furthermore, the final version of the DevOps Framework’s UI, thanks to the integration with the state machine described in the previous section, lets a user know if a certain tool is “active”, that is, whether or not it has received all the information it requires to work. Thus, the DECIDE UI disables access to the tools by showing a red icon next to the corresponding button, until they are ready to be executed, moment in which the icon changes to green.

Lastly, it is important to know that the naming of the labels that give access to the DECIDE tools have been changed to be more descriptive. This way, a user who is not familiar with the DECIDE environment will find it easier to understand the purpose of these tools.

The following image shows the new aspect of the DECIDE UI Dashboard. Annex A shows the final version of the interfaces of the integrated tools:



**Figure 3.** DevOps framework’s UI. Final version

# Implementation

## Functional description

The DECIDE DevOps Framework is the platform from which the different tools (KRs) 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. Lastly, the latest version of the DevOps Framework controls the DECIDE workflow, enabling tools as they are ready to be used and triggering the corresponding component when appropriate.

**Functionalities:**

DECIDE DevOps framework follows an incremental strategy, according to which different prototypes of the framework are periodically released (in months 15, 27 and 33). The current M33 prototype improves upon the M27 version and provides full 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*. **Covered**. The DevOps framework provides the means to launch the different tools, and automatically triggers some components. Besides, this version is able to enable or disable access to tools depending on whether or not all the necessary information has been provided to said tools, and, in case of an automatic redeployment, to trigger the appropriate tool when it corresponds.
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 [2]. The following table provides the status of the implementation of these requirements in the M33 prototype. The table represents an update on the requirements implemented for the M27 release and documented in section 6.1 of deliverable D2.7 [1].

**Table 2.** Requirements covered by the M33 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 finally 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. |
| 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 provides 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. | Requirement rejected. Out of scope of the project. |
| DEVOPS-REQ11 | DECIDE framework must provide a way for team members to plan the development process. | Requirement rejected. Out of scope of the project. |
| DEVOPS-REQ13 | DECIDE framework must support the application of best practices and design principles during the first phases of the development. | The framework, through the ARCHITECT component, provides development patterns, which are based on best practices and design principles of application development. |
| 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. |

### 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 final version of the DevOps framework can also handle access to the different tools, only enabling them when all the required information has been introduced, and can orchestrate the DECIDE workflow, triggering the corresponding tool during an automatic deployment.

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

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figure 4.** DevOps Framework within DECIDE

The DevOps Framework is composed of a backend, responsible for storing and manipulating data, and 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.

Lastly, this version of the DevOps Framework includes a state machine component, which is responsible for the orchestration of the DECIDE workflow.

## 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.

### 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 final version is shown in the diagram below. It differs from the previous (intermediate) version in that it includes the state machine component that has been described previously, and it integrates the ACSmI Billing component.

The framework 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.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figure 5.** DevOps Framework prototype’s architecture diagram

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

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, have been detailed in section 5.2.2 of deliverable D2.7 [1].

Regarding the isolation of each microservice, and as mentioned in deliverable D2.6 [4], 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.

### 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 [4], so this section will only analyse those components that have been added for this release: the state machine component and the updates to the DECIDE UI.

#### State machine API

The state machine is started directly from the State Machine component through the REST interface. In order to test it independently, it needs to be launched externally by a call to the REST interface.

<http://85.91.40.245:8095/statemachine/swagger-ui.html>**¡Error! Referencia de hipervínculo no válida.**

This is the yaml configuration file generated from the swagger editor with whole state machine API configuration:

swagger: '2.0'

info:

description: This is the API documentation for State Machine service of DECIDE h2020 european project

version: 1.0.0

title: State Machine

host: '85.91.40.245:8095'

basePath: /statemachine

tags:

- name: state-machine-api-controller

description: the state machine API

paths:

/setState:

post:

tags:

- state-machine

summary: Change state from application

description: Report a new application state in the system

operationId: setState

consumes:

- application/json

produces:

- application/json

parameters:

- in: body

name: state

description: State that wants to be reported

required: true

schema:

$ref: '#/definitions/state'

responses:

'200':

description: successful operation

'401':

description: Unauthorized

'403':

description: Forbidden

'404':

description: Not Found

/getState:

get:

tags:

- state-machine

summary: Get the state of application

description: Get the application state

operationId: getState

consumes:

- application/json

produces:

- application/json

parameters:

- in: body

name: data

description: Data to find state machine

required: true

schema:

$ref: '#/definitions/data'

responses:

'200':

description: successful operation

schema:

items:

$ref: '#/definitions/state'

'401':

description: Unauthorized

'403':

description: Forbidden

'404':

description: Not Found

definitions:

state:

type: object

required:

- app

- user

- state

properties:

app:

type: string

example: a1b2c3d4e5f6

description: Id of the application

user:

type: string

example: John

description: Id of the user

state:

type: string

example: detected

description: 'States of the application. Required values: SLA (0), CSP (1) or Microservice(2) '

description: State machine object description

data:

type: object

required:

- app

- user

properties:

app:

type: string

example: a1b2c3d4e5f6

description: Id of the application

user:

type: string

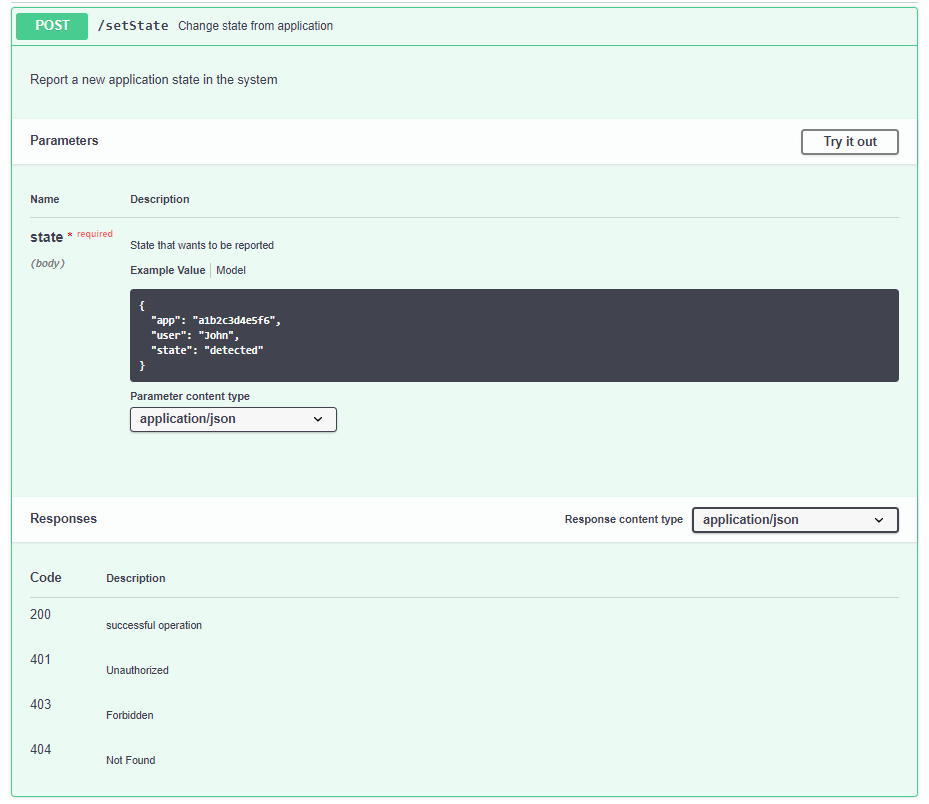
example: John

description: Id of the user

description: State machine object description

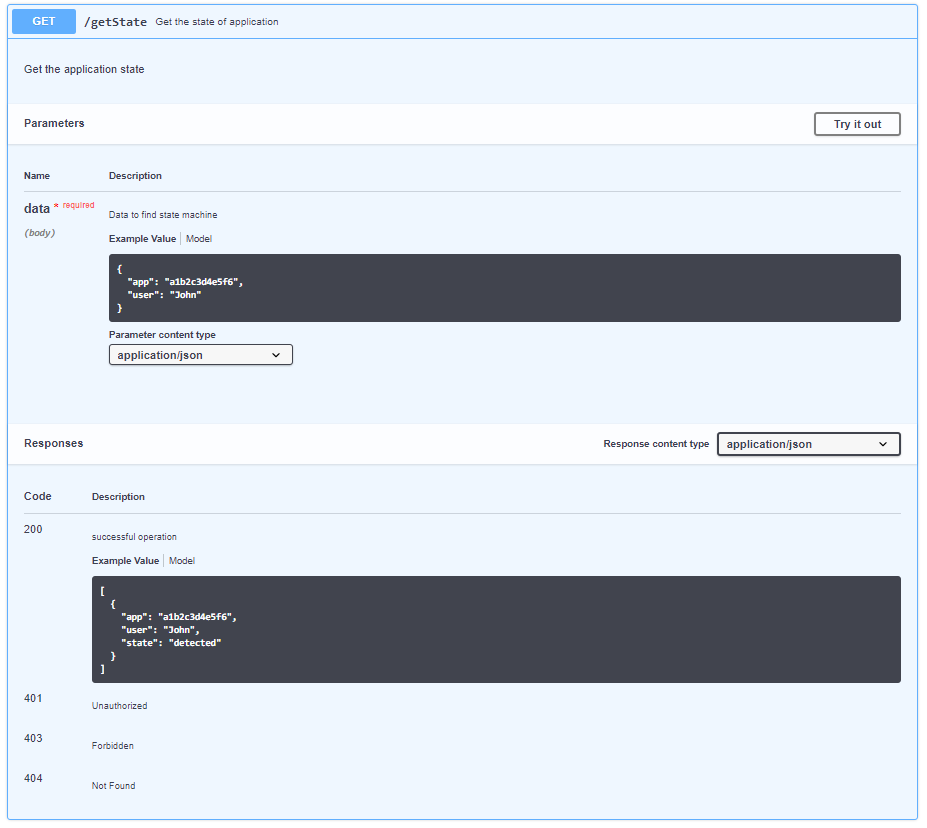
The state machine API consists of 2 methods:

* setState



When a user changes the application state using the DevOps framework, all components update the application status with this method. For testing purposes, it is necessary to pass the application name, username and the state of the application. Said state will be then updated in database.

* getState



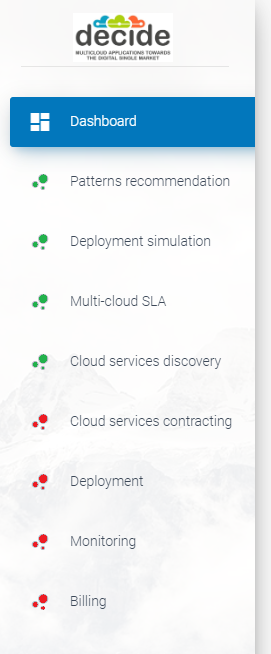
Automatically, the DevOps framework looks for the state of the current app using the getState method. For testing, it is necessary to pass the application and username and this method will look for the state in the database.

#### UI updates

The DevOps framework UI reflects the changes in the state, enabling new tools as the user advances in the workflow. When a new state is reached, the icon of the tool(s) affected by this state changes, to indicate that the tool is enabled or disabled.

A red icon means that the tools is not yet ready to be used, while a green icon represents that the tools has received all the required information and can be executed.

The following figure shows the new DevOps framework menu, where the first four tools are enabled and the last four are disabled (the Dashboard is always enabled by default):



**Figure 6.** DevOps Framework UI. Connection to state machine

# Delivery and usage

## Package information

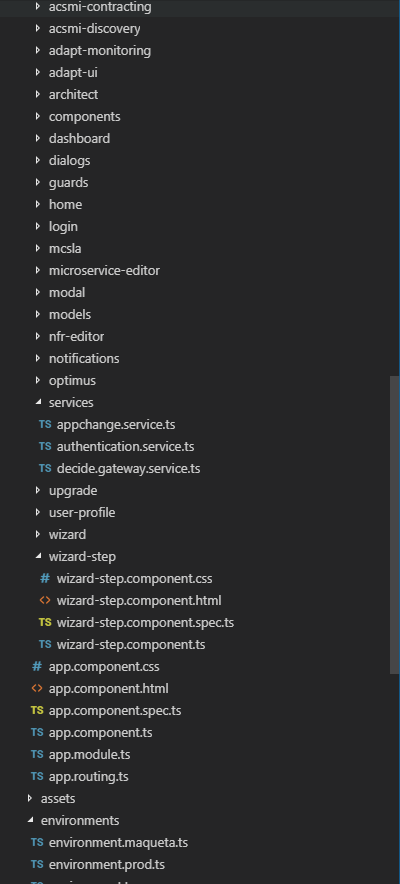
This section will briefly detail the architecture of each component. Since there are a lot of files involved, only the most representative ones will be explained to provide a better understanding of the DevOps Framework architecture.

It is important to remark that the general architecture of the DevOps framework components have not changed since the previous release, reported in deliverable D2.7 [1]. In order to have all components documented in one place, the information from the intermediate version has not been removed, highlighting instead the new additions in this release.

The annex of the deliverable mentioned above also contains multiple code snippets of the documented components.

### DevOps Framework Client

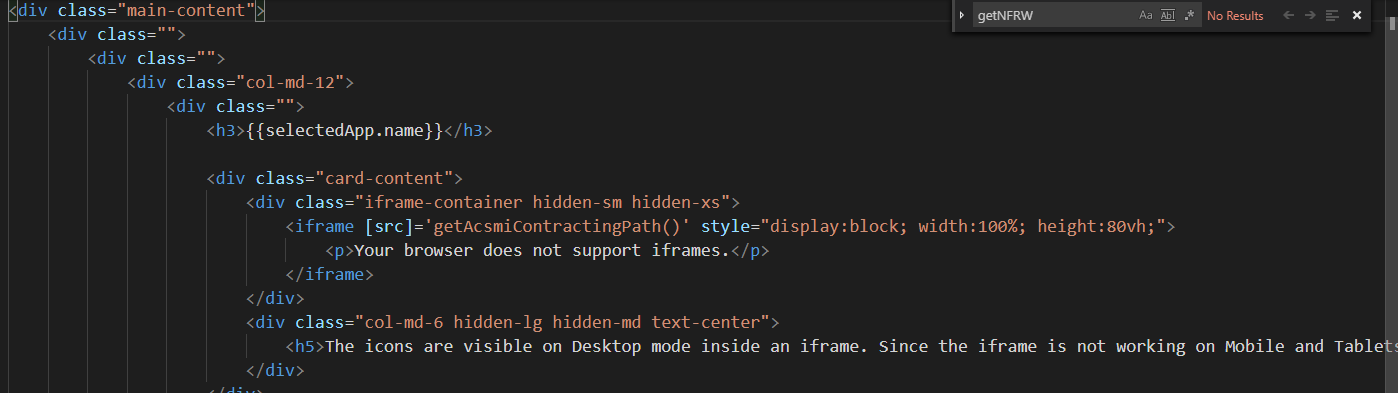
Here it is contained the front-end code developed in Angular 6. It provides the interface with which users can interact, and the communication with the back-end application.



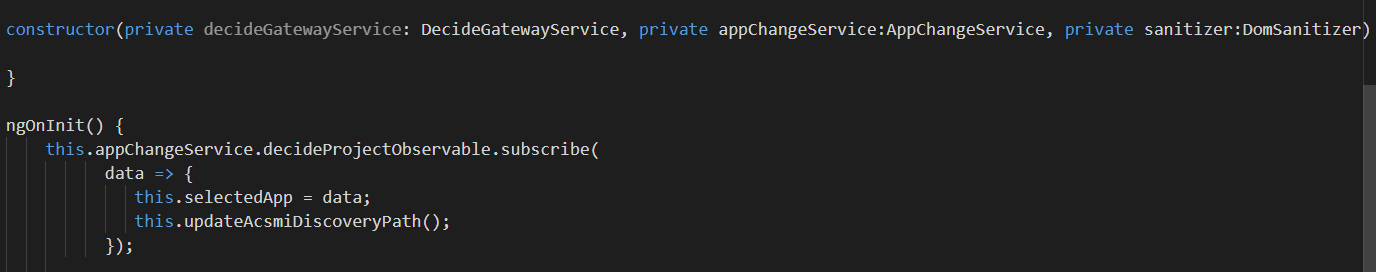
**Figure 7.** DevOps Framework Client’s file structure

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

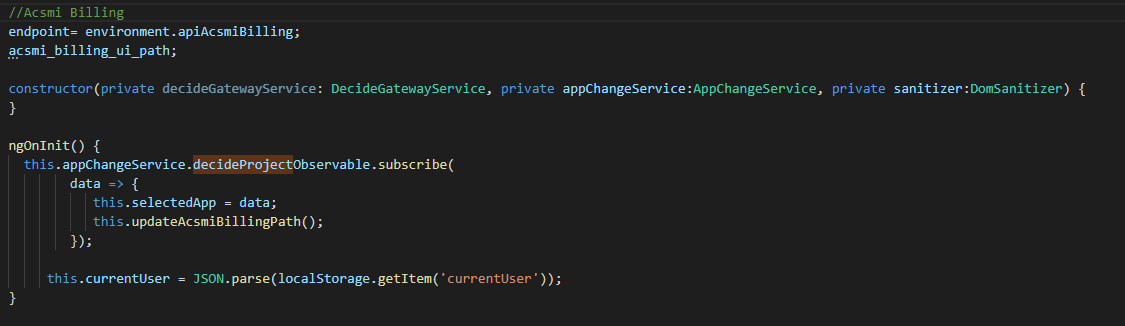
These modules request the correct URL of the service from .ts file and through the .html file shows an iframe or a table to show the data to the user in front-end. **In the latest release, ACSmI Billing has been integrated in the DevOps UI**. The following figures show two sample code excerpts of these components:



**Figure 8**. HTML code of ACSmI Contracting module



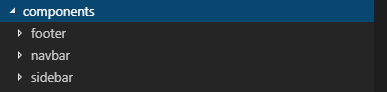
**Figure 9.** TS code of ACSmI Discovery module



**Figure 10.** TS code of ACSmI Billing module

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

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



**Figure 11.** Structure of the “components” module

**Guards**

This component contains the file auth.guards.ts to manage the permissions.

**Login**

Provides the view and the logic to implement the login view

**Edit Profile**

**This functionality has been added in the final release.** Provides the view and the logic to edit user profile.

**Credentials**

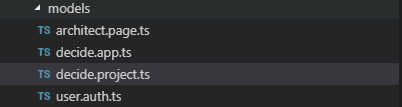
**This functionality has been added in the final release.** Provides the view and the logic to edit user credentials info and to store them safely.

**Microservice-editor and nfr-editor**

Both components implement the form to insert new microservices, or new NFRs during the creation of the project.

**Models**

This module contains the data models. Below, the structure of this module is shown, as well as a code snippet. **The latest version is updated to be coherent with the data model defined by the application controller.**



**Figure 12**. Structure of the “models” module

export class DecideProject {

    //General project parameters

    name:string;

    description:string;

    gitRef:string;

    token:string;

    //Application Description

    highTechnologicalRisk:boolean;

    importDescription:boolean;

    schemaVersion:string;

    microservices:Microservice[];

    nfrs: NFR[];

}

export class Microservice{

id:string;

name:string;

    stateful:Boolean;

    programmingLanguage:string;

tags: any[];

publicIp:Boolean;

endpoints: any[];

}

export class NFR{

type:string;

abstractValue: string;

value: String;

unit: string;

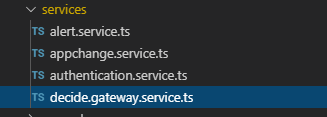
    tags: any[];

}

**Figure 13**. 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 14**. Structure of the “services” module

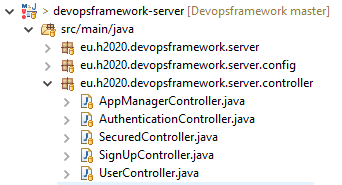
The most important service is *decide.gateway.service.ts*, which provides the communication of the backend with the external manager. **The *alert.service.ts* is a new addition in the final release**, and it is in charge of managing notifications to the user, such as the “loading” notifications.

### 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.

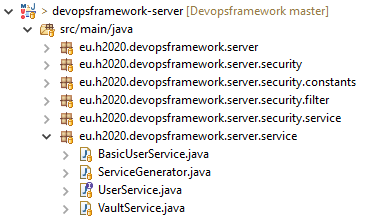
Below, the structure of this component is shown:



**Figure 15**. 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 new class has been added in the final version, to handle user management.**

The following services have been developed for DevOps framework server:



**Figure 16**. Services of the DevOps Framework’s Server

It should be noted the **integration service with Vault**, responsible for storing and retrieving the secrets used in the system.

## 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.

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.

The installation instructions, as well as the user manual presented in section 6.3 have remained unchanged since the previous release of the DevOps framework, reported in D2.7.

**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 configurated and accessible from your browser.
5. Access to the main DevOps framework web page in <http://localhost:4000> in your local machine browser.

## 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** |
| --- | --- |
| **ADAPT** | |
| ADAPT DO | 8081 |
| ADAPT monitoring | 8088 |
| VH | 8095 |
| **MCSLA** | |
| MCSLA service | 8082 |
| MCSLA ui | 8083 |
| Cloud Compendium | 8001 |
| **AppController** | |
| AppController | *Not required* |
| **OPTIMUS** | |
| OPTIMUS server | 8090 |
| **ARCHITECT** | |
| ARCHITECT server | 8001 |
| **ACSmI** | |
| ACSmI discovery registry | - |
| ACSmI discovery server | 8087 |
| ACSmI discovery client | 8087 |
| ACSmI 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 |

## Licensing information

This component is offered under the MIT license.

## 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>

# Conclusions

This document has presented the third prototype of the DevOps framework, corresponding to the M33 and final release. The new functionalities have been described, such as the implementation of the state machine and the UI changes. The pipelines that have been set up to deploy the different components have also been described.

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.

# References

|  |  |
| --- | --- |
| [1] | DECIDE Consortium, D2.7 Intermediate DECIDE DevOps Framework Integration, 2019. |
| [2] | DECIDE Consortium, “D2.2 Detailed requirements specification v2,” 2018. |
| [3] | DECIDE Consortium, “D2.1 - Detailed requirements specification v1,” 2017. |
| [4] | DECIDE Consortium, “D2.6 Initial DECIDE DevOps Framework Integration,” 2018. |
| [5] | Google, "Material Design," [Online]. Available: https://material.io/. [Accessed 20 February 2018]. |
| [6] | DECIDE Consortium, “AppManager,” 2018. [Online]. Available: https://git.code.tecnalia.com/decide/WP3/tree/master/AppManager. [Accessed 20 February 2018]. |
| [7] | Hashicorp, "Vault documentation," 2019. [Online]. Available: https://learn.hashicorp.com/vault/secrets-management/sm-static-secrets. [Accessed 20 February 2019]. |
| [8] | Hashicorp, "Introduction to Vault," 2019. [Online]. Available: https://www.vaultproject.io/docs/what-is-vault/index.html. [Accessed 20 February 2019]. |
| [9] | DECIDE Consortium, “D2.5 DECIDE Detailed architecture,” 2018. |
| [10] | DECIDE Consortium, “D4.2-Intermediate DECIDE ADAPT Architecture,” 2018. |
| [11] | DECIDE consortium, “D2.4 Detailed architecture v1,” 2017. |
| [12] | DECIDE Consortium, “D2.3 Integration and validation strategy,” 2017. |

# 

# Annex A. DECIDE Components’ UI

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figure 17.** ARCHITECT’s UI

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figure 18.** OPTIMUS’ UI

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figure 19**. MCSLA’s UI

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figure 20.** ACSmI Discovery’s UI

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figure 21**. ACSmI Contracting’s UI

Imagen que contiene captura de pantalla, interior

Descripción generada automáticamente

**Figure 22**. ADAPT DO’s UI (1)

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figure 23.** ADAPT DO’s UI (2)

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**Figure 24.** ACSmI Monitoring’s UI