MULTICLOUD APPLICATIONS TOWARDS THE DIGITAL SINGLE MARKET

Deliverable

D3.10 Initial multi-cloud native application controller

Editor(s):	Lena Farid (Fraunhofer)
Responsible Partner:	Fraunhofer
Status-Version:	Final – v1.0
Date:	30/11/2017
Distribution level (CO, PU):	Confidential

Project Number:	GA 726755
Project Title:	DECIDE

Title of Deliverable:	D3.10 Initial multi-cloud native application controller
Due Date of Delivery to the EC:	30/11/2017

Workpackage responsible for the Deliverable:	WP3 – Continuous Architecting
Editor(s):	Lena Farid (Fraunhofer)
Contributor(s):	Leo Li and Lena Farid (Fraunhofer)
Reviewer(s):	Lorenzo Blasi (HPE)
Approved by:	All Partners
Recommended/mandatory readers:	WP5, WP4, WP3, WP2

Abstract:	This software deliverable comprises the initial multi-cloud native application controller This initial version will concentrate on a specific programming language, cloud technology and standard.
Keyword List:	Application Controller, Deployment History, OPTIMUS, Deployment Topology
Licensing information:	The software documented in this deliverable is licensed under the Eclipse Public License version 2.0 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	26/10/2017	First draft version	Leo Li (Fraunhofer)
V0.2	17/11/2017	second draft version	Lena Farid (Fraunhofer)
V0.3	20/11/2017	Third draft version	Lena Farid (Fraunhofer)
V0.35	22/11/2017	Reviewed by HPE	Lorenzo Blazi (HPE)
V0.4	22/11/2017	Final version	Lena Farid (Fraunhofer)
V1.0	23/11/2017	Ready for submission	Leire Orue-Echevarria (TECNALIA)

Table of Contents

List of Figures

List of Tables

Terms and abbreviations

CSP	Cloud Service Provider
DoA	Description of Action
JAR	Java Archive
JSON	JavaScript Object Notation
SLA	Service Level Agreement
URI	Unified Resource Identifier
URL	Unified Resource Locator

Executive Summary

The document at hand accompanies the deliverable “D3.10: Initial multi-cloud native application controller” (software demonstrator) and documents it from a functional and technical perspective. This deliverable is one of two more to come.

The notion of said documentation is to provide developers with the information regarding the aim of the software, i.e. the Application Controller, how it is implemented, how it fits into the DECIDE project as a whole and how to use it.

The deliverable includes a JAVA library to create a deployment history of (micro-)services. It is packaged as a Jar file that delivers functions to create and alter entries in a JSON-based history file. The aim is to include the history in a working git repository. Functions to retrieve and push files to a repository are given as well. With this approach, the DevOps philosophy followed in the DECIDE project is further pursued. Furthermore, OPTIMUS or any interested DECIDE tool may re-use this library in order to access information regarding the current and historical deployment topology for a given application.

It is envisioned that the Application Controller will be extended in the future to include more functionality.

Future work and deviations from the Dow are discussed in the conclusion. Certain aspects have been moved to WP4 (CSP script generation) [1] and others will be part of the architectural work and implementation in Year 2 of the project.

1 Introduction

The functionality of the Application Controller as understood by the DECIDE consortium should for one reflect the status and state of the application and connect the former with the DECIDE tools in the sense of enabling each tool to understand its corresponding fulfilments.

In Year 1 of the project, the Application Controller has attained the role of assisting in managing the intelligence regarding the currently used deployment configuration and the historical ones. It keeps records whether a deployment configuration was successful and if any SLA violations had occurred in the applications operation time. With this information, OPTIMUS is able to suggest new and adequate deployment configurations.

The Application Controller is implemented as a globally re-useable library in the DECIDE framework. The main purpose is to offer global functions and processes to other components. The Application Controller is developed on top of a main git repository, which contains necessary information about the developed services.

1.1 About this deliverable

This document explains the implemented functions and processes of the current Application Controller library. Furthermore, a brief introduction is given to setup and integrate in other components.

Section 2 of this deliverable describes implementation details and Section 3 covers how to pull, build and use the library.

In section 4, the conclusion is presented along with deviations from the DoA [1] and an outline for future work.

2 Implementation

2.1 Functional description

The Application Controller component assists in managing the intelligence regarding the currently used deployment configuration and historical ones. It keeps records whether a deployment configuration was successful and if any SLA violations had occurred in the applications operation time. With this information, OPTIMUS is able to suggest new and adequate deployment configurations and not reuse a previous deployment configuration that deemed unsuccessful or faulty in terms of security, performance or legal awareness.

The following functionalities have been implemented in year 1 of the project as part of the Application Controller:

F1. Holding the intelligence of the different deployment configurations that the multi-cloud application has had in its operation time.
F2. Storing these deployment configurations in a JSON based history file in an accessible git repository, where the application description also resides.
F3. Provide OPTIMUS with the operations required in order to read and write the chosen deployment configuration. In the case of reading, avoiding those configurations that resulted problematic in terms of security, performance or legal awareness can be achieved.
F4. Provide the DECIDE DevOps Framework with the necessary operations to read from the historical configuration.
F5. The deployment history will include meta-data regarding the deployment configuration such as time and date of deployment, the current status, information on the microservice, CSP data and information regarding any SLA breaches that have taken place.
F6. The deployment history file is stored in an accessible location (git repository) and the mechanisms for accessing, updating and deleting the file and its entries are available.

The following table details the relationship between the Application Controller requirements indicated in a project internal collaborative online sheet[1] and the implemented functionalities, with a description of the coverage for each functionality.

Table 1. Relationship between Application Controller functionalities and requirements

Functionality	Req. ID	Coverage
F1 – F3	WP3-CONTR-REQ2, WP3-CONTR-REQ9, WP3-CONTR-REQ11	A library is implemented to cover this aspect and the data format (JSON) and structure has been provided to hold all relevant and needed information for OPTIMUS.
F4	WP3-CONTR-REQ1	The multi-cloud native application controller is implemented as a JAVA Library and can be used by any Java source Code very easily.
F5-F6	WP3-CONTR-REQ12	The library provides methods in order to access a git repository with the supplied credentials, push, and pull relevant information into a JSON file

[1] Application Controller Requirements Sheet: https://docs.google.com/spreadsheets/d/1R9pPkZFwxzld63Hfe_YzoxQAnabe85QguzNCLzAj968/edit#gid=1301479309

Functionality	Req. ID	Coverage
		dedicated for the historical deployment configuration.

2.1.1 Fitting into overall DECIDE Architecture

The Application Controller library acts as a facilitator for OPTIMUS in terms of creating and accessing historical information regarding the applications deployment topologies. The history file is located in a git repository adjacent to the Application Description. Both, the repository and the Application Description are initially created during the Application development phase.

The Application Description holds all necessary information for describing and classifying the application. It also holds the state of the application. The deployment history file complements the Application Description by providing information regarding the historical deployment configurations in a simple structure that is easily understood and parsed by the DECIDE tools (OPTIMUS more specifically).

As depicted in Figure 2, the Application Controller is an intermediate step between OPTIMUS and the Git Repository and provides the required interfaces for it (implemented as Java operations).

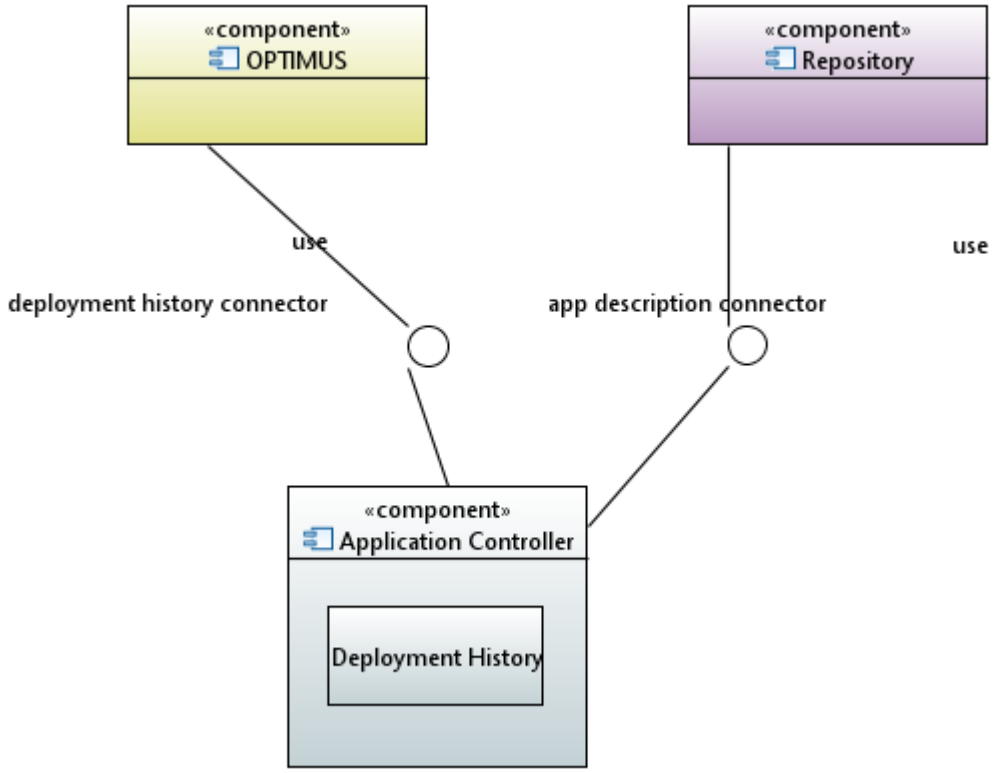

Figure 1. Application Controller in the context of DECIDE

Figure 2 depicts the use cases for the Application Controller at this stage of the project.

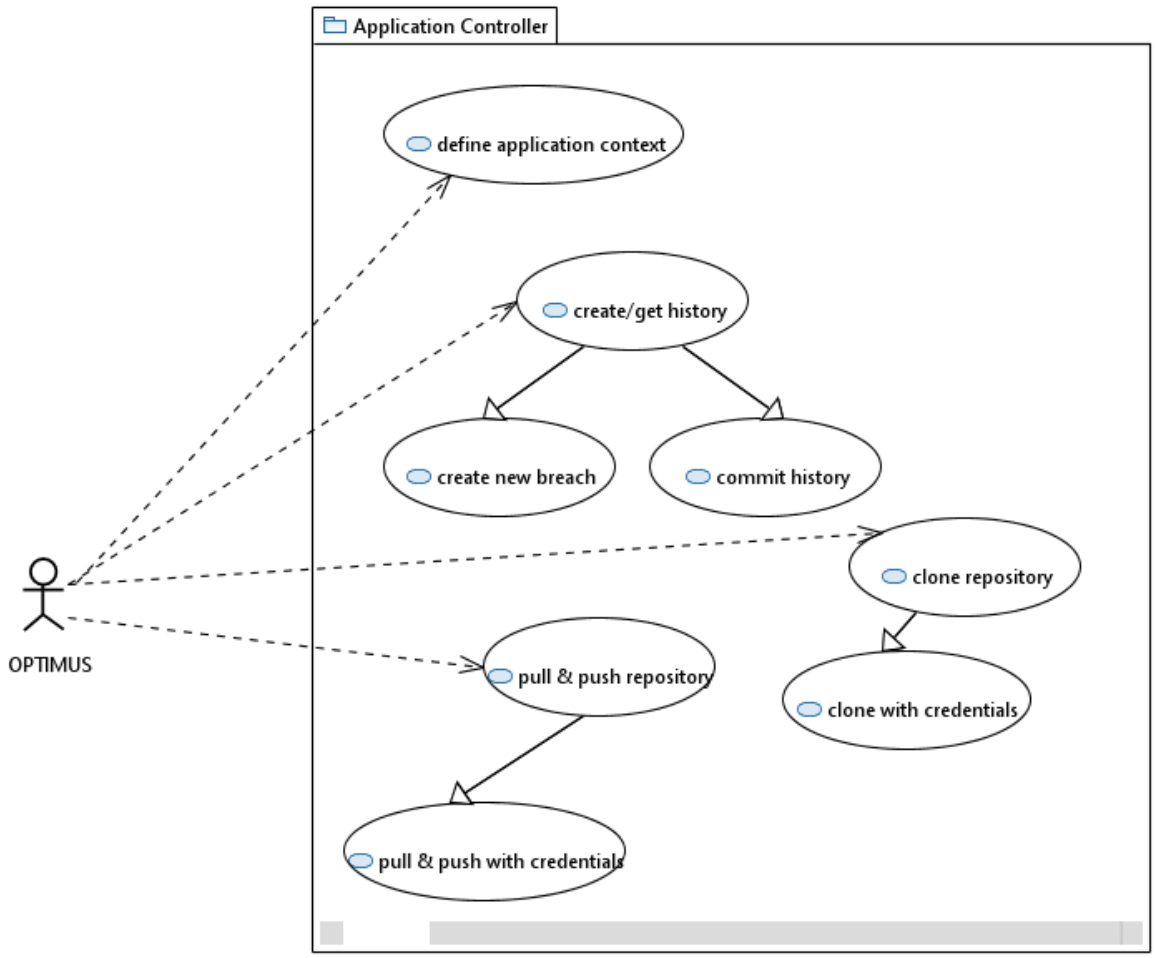

Figure 2. Use Case Diagram from Application Controller

The use cases depicted in Figure 2 are all implemented as methods as part of the JAVA Library and form an intermediate step between OPTIMUS and the Git repository. The following is a brief description of the use cases:

UC1. Define application context

This use case adds the application context to the Application Controller[2]. In terms of the git repository URI (local and remote), the git credentials (username and password) for accessing the repository (if a private one), the local directory to push changes to, and setting the URI for the Application Description.

UC2. Create/ get history

This use case is to get an existing history file or create a new one. Its specialisation lies in creating a new breach to be added to the history file and committing the history file to a local repository.

UC3. Clone repository

[2] In Java terms, this use case is for instantiating the Application Controller with the relevant properties.

This use case allows for cloning an existing repository, making sure that consistency and concurrency controls are met. A specialisation is when using a private repository; in this case, the credentials[3] are needed.

UC4. Pull & push repository

This use case allows for pushing changes and pulling from the remote repository, making sure that consistency and concurrency controls are met. A specialisation is when using a private repository; in this case, the credentials are needed.

The following is a sequence diagram to help understand how to use the library in the correct sequential order. The sequence diagram depicts the use case UC2 - Create history and create new breach.

[3] The credentials for accessing git will always only be available during the lifetime of an instance of the Application Controller and have always to be supplied when a new instance is created. The secure storage of the credentials will be managed in a future release.

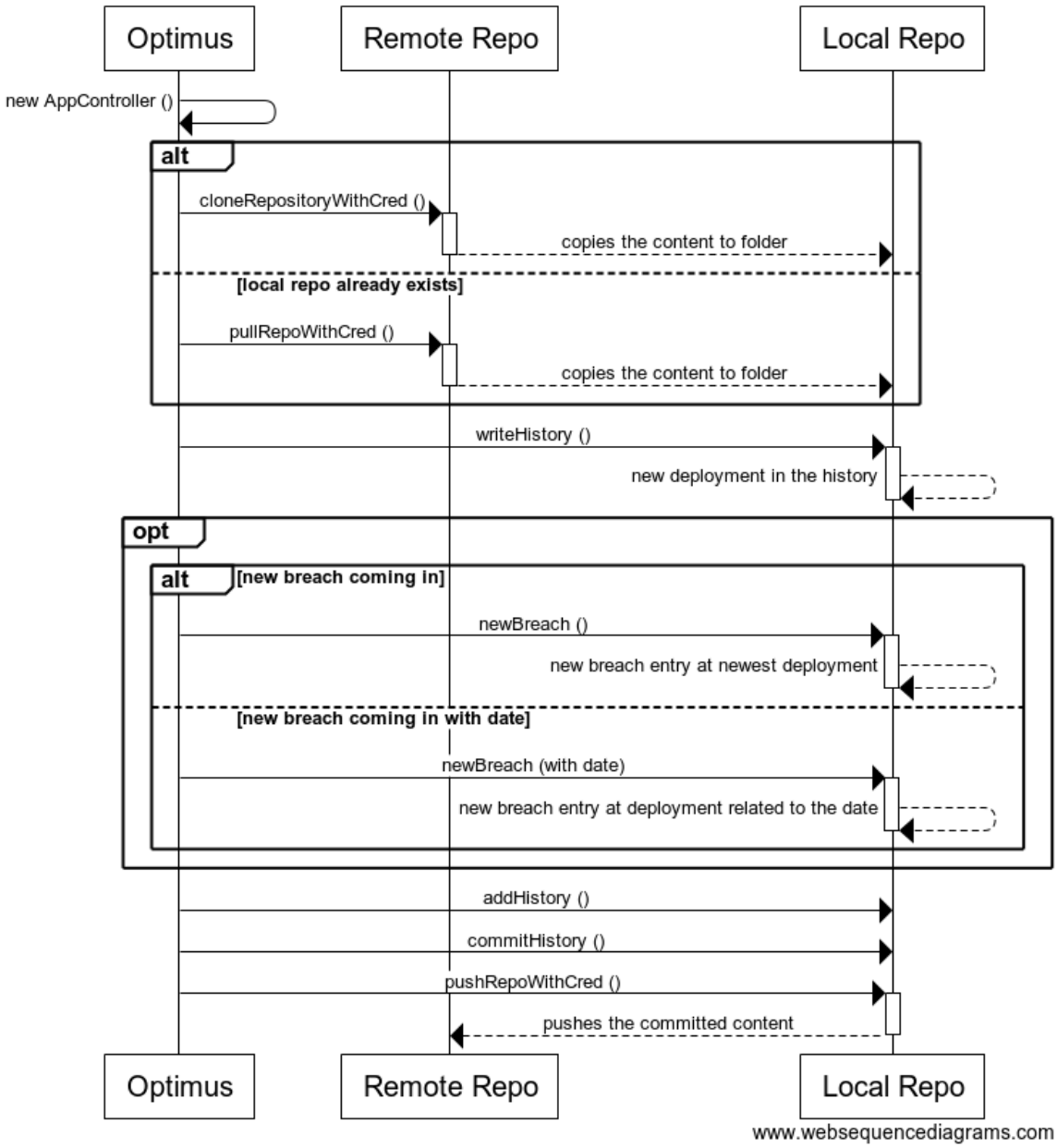

Figure 3. Sequence Diagram for creating a history file

The sequence diagram depicted in Figure 3 shows the sequential order for using the library in order to ultimately add a new deployment to the deployment history and secondly to add new breaches to an existing deployment in the history file.

First a new deployment is to be added to the deployment history and secondly to add new breaches to an existing deployment in the history.

1. *new AppController():* First start by initiating the Application Controller Class with the required input parameters.
2. *cloneRepositoryWithCred():* this method clones the remote repository to a local folder. If the repository already exists the method *pullRepoWithCred()* just updates the local repository with changes from the remote one.
3. After a successful deployment takes place a new deployment entry can be added by using the *writeHistory()* method.

4. The method *addHistory()* registers new added content to the local repository.
5. And *commitHistory()* creates a new commit with all the registered content.
6. Finally the local changes can be pushed to the remote repo with *pushRepoWithCred().*
7. At any point a new breach can be added by using the *newBreach()* method. To guarantee the repositories stay synchronized the local repository should be updated with *pullRepoWithCred()* before editing/adding new entries to the history. The *newBreach()* method can be used with a date to relate the new breach to a certain deployment.

In the case of a public remote repository, which does not expect credentials when accessing it, the developers should use the respective method names without the trailing ("*withCred*").

2.2 Technical description and specification

The library is implemented using the JAVA programming language; it connects to git repositories and writes to a JSON file. The history is structured around the different microservices that the DECIDE framework is deploying.

The prototype defines the class Application Controller that holds all necessary methods to write and update the deployment history. A Java Doc goes further into details.

The app controller uses the JGIT [2] and JSON.simple [3] libraries: the first one to communicate with a git repository to be specified and the second one to write JSON entries to the deployment history. The Application Controller library is packaged with MAVEN, which creates a reusable JAR file.

The Application Controller may be used as a normal jar file to be integrated in other components of the DECIDE framework. The library can be added to your projects and offers all the needed methods in order to be able to create the JSON History file with its related content, push it to a git repository and pull it when needed.

The Library also manages the connections including the credentials for git. The secure storage of the credentials will be managed in a future release

2.2.1 Application Controller Operations

The most up-to-date documentation for the methods, their parameters and return values can be found in the JavaDoc. An excerpt of the JavaDoc is given below:

- `void addHistory()` – This operation is git specific and ads the JSONHistory file to the index to be commited next.
- `void cloneRepository()` - Clone the defined repository to the defined directory or if it already exists opens the repository.
- `void cloneRepositoryWithCred()` - Clone the defined repository to the defined directory or if it already exists opens the repository.
- `void commitHistory(java.lang.String message)` - Commit the latest changes in the repository.
- `java.lang.String getApplicationDesc()` – get the specified Application Description.
- `java.lang.String getDirectory()` – get the local directory to commit to.
- `org.eclipse.jgit.api.Git getGitRepository()` – get the git repository itself.
- `java.lang.String getRepositoryUrl()` – return the git repository URL
- `java.lang.String getUsername()` – get the user name for accessing the git repository.

- `void newBreach(java.lang.String serviceId, org.json.simple.JSONObject slaBreach)` - Create a new SLABreach object and add it to the last saved deployment of the defined service in the existing history. Creates a new history file if it does not exist yet.
- `void newBreach(java.lang.String date, java.lang.String serviceId, org.json.simple.JSONObject slaBreach)` - Create a new SLABreach object and add it to the defined dated deployment of the defined service in the existing history. Creates a new history file if it does not exist yet.
- `void pullRepo()` - Pull new changes from the defined repository and includes them in the local directory.
- `void pullRepoWithCred()` - Pull new changes from defined repository which is private and include them into the local directory.
- `void pushRepo()` - Push new content to the repository.
- `void pushRepoWithCred()` - Push new content to the private repository.
- `void setApplicationDesc(java.lang.String applicationDesc)` – set the Application Description.
- `void setDirectory(java.lang.String directory)` – set the local directory to commit to
- `void setGitRepository(org.eclipse.jgit.api.Git gitRepository)` – set the git repository itself
- `void setRepositoryUrl(java.lang.String repositoryUrl)` – set the git repository URL
- `void setUsername(java.lang.String username)` – set the user name for accessing the git repository.
- `void writeHistory(java.lang.String serviceId, java.lang.String serviceProvider)` - Create a new history entry in the existing history. Gets the current history file or creates a new one if it doesn't exist.

It is important to note that the method `newBreach` accepts a `JSONObject` as the "sla breach". With this approach, the Application Controller library is independent by any other implementations (e.g. OPTMUS) and does not need to define any specific parameters to be supplied. The viability of this approach will be investigated in the coming months, in the DECIDE integration phase.

2.2.2 JSON for Deployment History

The history file is written in a JSON serialisation, the fields are listed in Table 2. The file is made up of an array of microservices and their nested history.

Table 2. Fields in a Deployment History JSON-based file

Key	Nested Elements	Type	Description
UUID of microservice		Array of deployment history	UUID of the microservice as in the Application Description
	Date	String	The system date at the time of registering the breach or just the deployment configuration.

Key	Nested Elements	Type	Description
	deployed_on	String	UUID of the CSP the microservice is deployed on
	sla_breaches	Array of sla breaches (JSONObjects)	Types and values for a breach. Serialisation currently left open for developers.

The following JSON description is an example of a deployment history file with an application consisting of two microservices, which are deployed and have suffered SLA breaches.

As breaches come in, they are added to their respective microservice entry in the JSON file under "sla_breaches". New deployments are added as a new deployment configuration under the respective microservice.

```
[   {
      "Microservice2":[  {
            "date":"1510152532949",
            "deployed_on":"AWS",
            "sla_breaches":[
                      {"availability":"80%"},
                      {"connectivity":"80%"}
                         ]}
      ]},
   {
      "Microservice1":[         {
            "date":"1510152815583",
            "deployed_on":"Azure",
            "sla_breaches":[
                      {"availability":"50%"}
                      ]}
      ]  } ]
```

3 Delivery and Usage

3.1 Package Information

The deliverable contains two folders and the prebuilt JAR file. The documentation folder holds the Java Doc files and this document itself. The source-code folder contains the implementation of the Application Controller and holds all necessary files to compile and build the library.

The source code for the Application Controller is organised under the following Java packages:

- eu.DECIDEh2020.appcontroller

  contains the class AppController which includes all necessary methods and functions to create and maintain the JSON based deployment history.

3.2 Build Instructions

The code is currently available on DECIDE's Public GITLAB. https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/M12/AppController

The project uses MAVEN as a build tool. In order to build the library both MAVEN and JAVA version 8 have to be installed. To build the JAR file execute the following command:

```
$> mvn clean package
```

3.3 Library Usage

The Application Controller may be used as a normal JAR file to be integrated in other components of the DECIDE framework. The library can be added to your projects and offers all needed methods in order to be able to create the JSON History file with its related content, push it to a git repository and pull it when needed.

The most up-to-date documentation of the methods, their parameters and return values can be found in the JavaDoc.

3.4 Licensing Information

The source code is licensed under the Eclipse Public License version 2.0.

4 Conclusions

In conclusion, a small library was implemented to write and update a deployment history of all (micro) services the DECIDE framework is deploying.

The history file created by the Application Controller library can be accessed programmatically via a git repository in order for OPTIMUS to avoid suggesting deployment topologies that previously have had SLA breaches by the CSP.

The library has been described from a functional and technical perspective in order for developers to understand its role in the project and how to integrate it and use it. A simple JAR file is to be added as a library to projects.

It is important to note that the description of the Application Controller in the DoA [3] and it is envisioned functionality have been conceptualised and its solutions have been introduced in various components and parts of the DECIDE Framework.

The DoA states *"Once the DECIDE Optimus tool has suggested the most convenient deployment configuration based on the requirements elicited by the user, it is time to select which deployment script is the selected one. The DECIDE application controller has a double aim. Firstly, it will* ***apply the necessary annotations*** *in the source code at component and micro-service level in order to be read by the deployment engine as well as for the self-adaptive tools to be developed in T4.1 and will then* ***create apply the corresponding deployment scripts****.* ***Standards*** *such as CIMI, ISO 19941, ISO 19944, OASIS TOSCA, etc.* ***have to be supported****, depending on available interfaces at the target CSPs. Hence, a specific interface will allow to* ***plug-in adaptors to translate topology*** *and configuration information into the respective target formats. The second aim of this controller is to* ***hold the intelligence of the different deployment configurations*** *that the multi-cloud application has had in its operation time. Storing these deployment configurations will allow avoiding those configurations that resulted problematic in terms of security, performance or legal awareness."* [3].

The following table summarises the T3.4 task's output as described in the DoA and gives insight on how they have been addressed at this stage of the project.

Table 3. Application Controller Tasks

Task	Implementation	Explanation
Apply necessary annotations in source code		Subject to talks and considerations in Year2
Create and apply the corresponding deployment scripts	Not applicable	Script generation has been moved to ADAPT. As explained in D4.1 [4], the deployment configuration scripts are dependent on the technology selected for the ADAPT implementation.
Support of standards		Subject to talks and considerations in Year2
Plug-in Adapters to translate topologies	Not applicable	Script generation has been moved to ADAPT. As explained in D4.1 [4], the deployment configuration scripts are dependent on the technology selected for the ADAPT implementation

Task	Implementation	Explanation
Hold intelligence wrt. different deployment configurations	Implemented as a Java Library	Documented in this deliverable

References

[1] DECIDE Consortium, "DECIDE Annex 1 - Description of Action," 2016.

[2] JGIT. [Online]. Available: https://www.eclipse.org/jgit/.

[3] „JSON.Simple,“ [Online]. Available: https://github.com/fangyidong/json-simple.

[4] DECIDE Consortium, „D4.1 Initial DECIDE ADAPT Architecture,“ 2017.