Deliverable D3.11

Intermediate multi-cloud native application controller

Editor(s):	Anne Barsuhn (Fraunhofer) Simon Dutkowski (Fraunhofer)
Responsible Partner:	Fraunhofer
Status-Version:	Final - v1.0
Date:	29/11/2018
Distribution level (CO, PU):	CO

Project Number:	GA 731533
Project Title:	DECIDE

Title of Deliverable:	Intermediate multi-cloud native application controller
Due Date of Delivery to the EC:	30/11/2018

Workpackage responsible for the Deliverable:	WP3 – Continuous Architecting
Editor(s):	Fraunhofer
Contributor(s):	Leo Li (Fraunhofer) Lena Farid (Fraunhofer) Anne Barsuhn (Fraunhofer) Simon Dutkowski (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 GNU Affero General Public License Version 3 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	19/11/2018	First draft version	Anne Barsuhn (Fraunhofer)
v0.2	20/11/2018	Second draft version	Simon Dutkowski (Fraunhofer)
v0.3	22/11/2018	Finalized for internal review	Simon Dutkowski (Fraunhofer)
V0.4	29/11/2018	Final editing before release	Simon Dutkowski (Fraunhofer)
V1.0	29/11/2018	Ready for submission	Leire Orue-Echevarria (TECNALIA)

Table of Contents

List of Figures

List of Tables

Terms and abbreviations

ADAPT	Application Deployment and Adaptation
API	Application Programming interface
CIMI	Cloud Infrastructure Management Interface
CO	Confidential
CSP	Cloud Service Provider
DECIDE	DEvOps for trusted, portable and interoperable multi-Cloud applications towards the Digital singlE market
DevOps	Development and Operations
DoA	Description of Actions
EC	European Commission
GA	Grant Agreement
GNU	GNU is Not Unix
ID	Identifier
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
OASIS	Organization for the Advancement of Structured Information Standards
POJO	Plain Old Java Objects
PU	Public
SCM	Source Code Management
SLA	Service Level Agreement
TOSCA	Topology and Orchestration Specification for Cloud Applications

Executive Summary

The document at hand accompanies the deliverable "D3.11: Intermediate multi-cloud native application controller" (software demonstrator) and documents it from a functional and technical perspective. This deliverable is the second of three. This document is the revision of the initial document with the same title [1] and contains content that is reused.

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 implemented Java library from the first prototype concentrated on the definitions of the main descriptor, the Application Description and its storage in a git repository. A JSON schema was specified for allowing validation of Application Description instances to ensure proper information exchange between the different DECIDE tools. A first draft version of the Deployment History was also implemented, but not integrated in the usage scenarios of Year 1. Certain aspects have been moved to WP4 (CSP script generation and topology translation) as described in D4.1 [2].

This prototype of Year 2 contains a revised Application Description schema and updated data binding to better support the different tools and also to cover the new usage scenarios like the semi-automatic re-deployment. Especially for the re-deployment scenario, the main innovation was a revised structure of the Deployment History, but it is expected not to be the final version and needs further reflection. Furthermore, the Year 2 prototype was improved regarding the general git handling, the API design and the validation analysis. 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 application or the current and historical deployment topology for a given application.

In Year 3 further efforts are necessary to finalize the Deployment History definition and to improve the management of it in all usage scenarios. Second, the general git implementation needs to be improved to proper handling any conflicts between local and remote revisions of descriptors. It is envisioned that the Application Controller will be extended in the future to include more functionality, e.g. allowing the management of logical groups of microservices and any kind of relations between NFRs, microservices, patterns or other elements of the Application Description.

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.

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.

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.

In Year 2 of the project the Application Controller reflects a revised Application Description focusing more on the deployment and runtime information. Because the re-deployment scenarios are a key part of the M24 milestone, further efforts are made to improve the deployment history definition in order to better reflect the requirements from OPTIMUS side. Furthermore, the API is enhanced with additional functionality for a better flexibility regarding the git [3] handling and possible project structure requirements.

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.

1.2 Document structure

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 [4] and an outline for future work.

2 Implementation

2.1 Functional description

Beside the general managing of the Application Description model and the encapsulation of the git [3] repository handling, 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 [5] 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 as part of the Application Controller:

F1. Holding the technical definition of the Application Description and provides controlled access for managing and validation based on a JSON schema describing the structure of the description.
F2. Storing Application Descriptions in a JSON based file in an accessible git repository.
F3. Holding the intelligence of the different deployment configurations that the multi-cloud application has had in its operation time.
F4. Storing these deployment configurations in a JSON based history file, defined by a JSON schema, in the same git repository where the Application Description resides.
F5. Provide OPTIMUS [5] 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.
F6. Provide the DECIDE DevOps Framework [6] with the necessary operations to read from the historical configuration.
F7. 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.
F8. 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 the deliverable for requirements [7] 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, F2	WP3-CONTR-REQ11, WP3-CONTR-REQ12	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.
F3, F4, F5	WP3-CONTR-REQ2, WP3-CONTR-REQ9	An additional JSON schema for the Deployment History is defined and management for a separate file in the same repository is implemented.
F6	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.

Functionality	Req. ID	Coverage
F7, F8	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 dedicated for the historical deployment configuration.

2.1.1 Fitting into overall DECIDE Architecture

The Application Controller library acts as a facilitator for OPTIMUS [5] 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).

The Application Controller provides a common way to create, update and validate all project related information. The Application Controller also maintains a technical definition of all descriptors that defines a DECIDE application project, currently the Application Description and the Deployment History, allowing to validate the correctness and integrity of the project descriptors.

2.2 Technical description

The two main aspects of the Application Controller are the maintaining of the two descriptors in a git repository. Figure 11 depicts the high-level simplified use cases that the Application Controller covers.

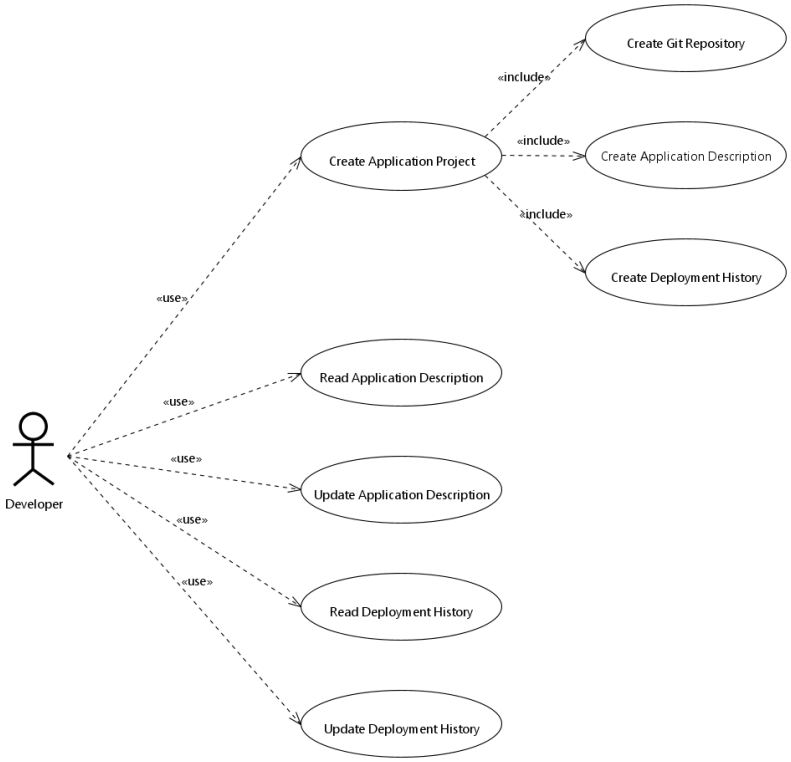

Figure 1. Use Cases Diagram

Create Application Project

The Application Controller allows the creation of a new DECIDE application project. A DECIDE application project represents an application development project within the context of the DECIDE DevOps Framework. If necessary, a local git repository is created, or an existing local or remote repository is reused. The two descriptors, the Application Description and the Deployment History are created and initialized.

Read Application Description

The Application Controller opens existing DECIDE projects for reading and writing. The Application Description will be validated against the defined JSON schema to ensure syntactical correctness. The Application Controller provides a POJO[1] based model of the Application Description for easy finding and processing of the contained information.

Update Application Description

The Model can be easily manipulated and written back to the descriptor file in the project folder. In strict mode each modification to the Application Description will be validated before applied to ensure syntactical correctness against the defined schema. If required, a synchronization with the related remote repository can be requested.

Read Deployment History

The Application Controller allows the retrieving of the deployment history from a DECIDE project. In strict mode that includes also a validation against the defined JSON schema to ensure syntactical

[1] POJO: Plain Old Java Object

correctness of the Deployment History. The Application Controller provides a POJO based model of the Deployment History for easy finding and processing of the contained information.

Update Deployment History

The retrieved model of the Deployment History can be easily manipulated and written back to the descriptor file in the project folder. In strict mode the updated Deployment History will be validated to ensure syntactical correctness against the defined schema. If required a synchronization with the related remote repository can be requested.

2.2.1 Data Model of the Deployment History

A complete data model description for the Application Description can be found in [8]. This chapter focuses on the data model for the Deployment History. It is in general a list of deployment schemas enriched with a date and corresponding Service Level Agreement (SLA) breaches. Table 22 lists all properties of element type *HistoryEntry*. The Deployment History descriptor is simply an array of *HistoryEntry* elements.

Table 2. Properties of element type *HistoryEntry*

Element Name	**HistoryEntry**		
Description	The element that holds a deployment schema and related SLA breaches		
Property	**Type**	**Cardinality**	**Definition**
date	String	1..1	The deployment date for the schema.
schema	SchemaElement	1..1	The deployment schema exactly as defined in [8] as part of the Application Description
slaBreaches	Array of Objects	0..n	A list of SLA breaches that invalidates the deployment schema and initiated a re-deployment

2.2.2 Components description

Figure 22 shows the component diagram of the Application Controller. To give more inside of each depicted component in the following each component is briefly described.

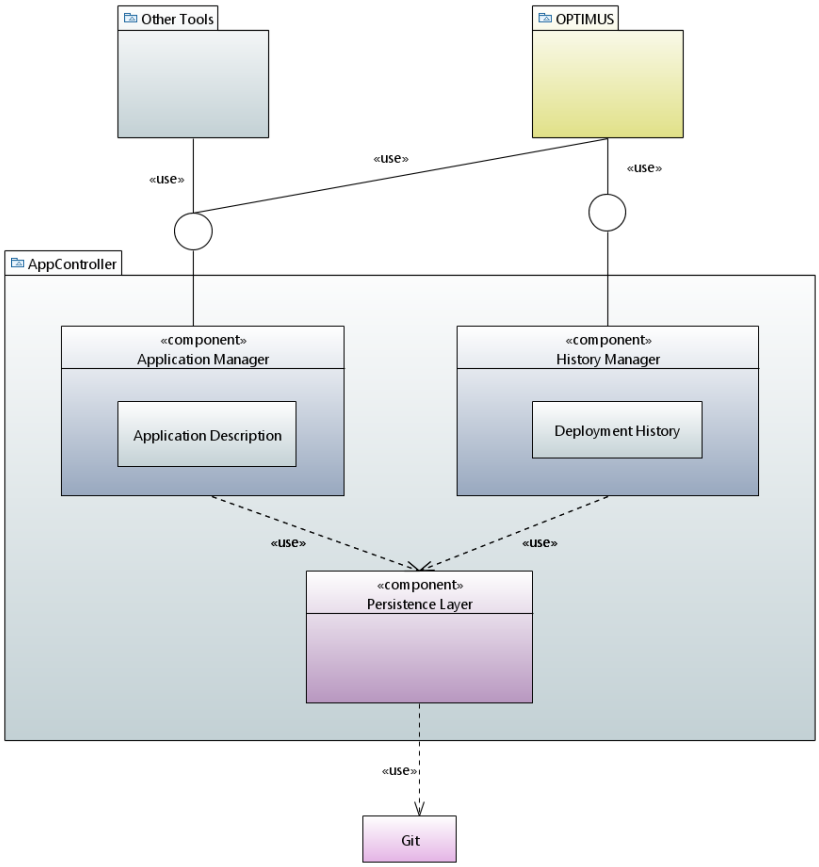

Figure 2. Application Controller Component Diagram

Application Manager

The Application Manager is the initial access point for working with the DECIDE project. It holds the logic for automatic validation, synchronization between local and remote repositories and the mapping between the JSON structure of the Application Description and the data binding to POJOs. It is accompanied by two sub-components, an Application Description factory and an Application Description Helper.

History Manager

The History Manager complements the Application Manager with the same functionalities related to the Deployment History. Except the opening of DECIDE projects, it holds the logic for the data binding between the JSON structure of the Deployment History and the corresponding POJOs. It is accompanied by a sub-component History Factory helping creation and validation of the Deployment History.

Persistence Layer

The Persistence Layer abstracts the git repository handling and encapsulates all low-level git [3] operations. This theoretically allows the utilization of other source code management (SCM) technologies, like Mercurial [9] or even Subversion [10].

3 Delivery and usage

3.1 Package information

The Application Controller is implemented as a shared library based on the Apache Maven build tool [11]. Therefore, it follows the usual maven project structure. All dependencies are defined in the *pom.xml* file.

```
|--- src
|     |--- main
|     |     |--- java
|     |     |     |--- … java packages
|     |     |--- resources
|     |           |--- application_description.schema.json
|     |           |--- optimus_history.schema.json
|     |--- test
|           |--- java
|           |     |--- … java test packages
|           |--- resources
|                 |--- … test resources
|---LICENSE.txt
|---README.md
|---pom.xml
```

The package also contains the JSON schema [12] files for the Application Description and the Deployment History. They are located in the folder *src/main/resources*:

- *application_description.schema.json* – JSON schema of the application description structure
- *optimus_history.schema.json* – JSON schema of the deployment history structure

The project tree contains beside the sources the following relevant additional files:

- *README.md* – Short installation and usage instructions
- *LICENSE.txt* – License information

3.2 Installation instructions

The project is available via a git repository. If you have access, do the following steps:

```
$ https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components.git
$ cd AppController
```

The project uses Maven as build tool [11]. After the successful build the jar and a fat jar can be find in the *target* directory. To build use the following command:

```
$ mvn clean package
```

Use *-DskipTests* option if the test repository is not accessible for you. If you would like to do the tests, edit the test class *AppManagerTest* and provide the necessary remote repository information.

For non-Maven based projects you can take the build jar file located in the target directory after executing the build command and put it in the classpath of your application. There is also a fat jar provided containing all dependencies if required.

For Maven based projects you need to install it in a Maven repository which your application can access. E.g. to put it in your local maven repository, you can simply call

```
$ mvn install
```

Finally, your application pom.xml requires the following dependency:

```
<dependency>
    <groupId>eu.DECIDEh2020</groupId>
    <artifactId>app-controller</artifactId>
    <version>0.0.15</version>
</dependency>
```

3.3 User Manual

The User Manual describes the public API of the Application Controller. More examples are provided in the test classes located in *src/test*. In general, the library provides its API through the following main classes:

- *AppManager*
  The initial entry point for working with a DECIDE project, including the git repository.
- *AppDescriptionFactory*
  This class provides static methods for creating, loading, saving and validating app description instances.
- *AppDescriptionHelper*
  This class provides convenient methods to access internal information of the app description by processing any conventions made by the project. E.g. retrieves groups and lists defined by tags.
- *HistoryManager*
  Entry point for working with the Deployment History. Before getting this manager an *AppManager* must be initialized.
- *HistoryFactory*
  Helper methods for creating correct history entries.

For convenience and for avoiding the developer to work with native JSON structures, which is error prone when working on a complex structure like the Application Description, the Application Controller provides a data binding to Plain Java Objects (POJO). These model classes are located in the java package *eu.DECIDEh2020.appManager.models*

3.3.1 Opening or Creating a DECIDE Application Project

Before the Application Controller can be used, it initially needs to be pointed to a DECIDE application project, currently represented through a git repository with at least Application Description file and optionally a Deployment History file. The *AppManager* class offers a set of methods to open the project (Table 33). If the local path does not contain a git repository it will be implicitly converted to a git repository.

Table 3. Static open methods of *AppManager*

Class	**AppManager**	
Method	**Parameter**	**Description**
static open	Path localPath	Open a local directory as DECIDE project. If it is not already a git repository it will be initialized as git repository and all files will be added.
	String gitRef String username String password Path localPath	Open a remote git repository as DECIDE project with user credentials. If the local path is not already a git repository, the remote will be cloned. Otherwise the

		local repository will be updated (pulled) from the remote repository.
	String gitRef String token Path localPath	Open a remote git repository as DECIDE project with a deployment token. If the local path is not already a git repository, the remote will be cloned. Otherwise the local repository will be updated (pulled) from the remote repository.

3.3.2 Access to the Application Description

The complete Application Description itself is hold by the model class *AppDescription*. A small example on getting the *AppDescription* and saving it using the *AppManager* (exception handling omitted due to better readability):

```
AppManager appManager = AppManager.open(gitRef, username, password, localPath);

// get the Appdescription
AppDescription appDescription = appManager.getAppDescription();

// do something with the AppDescription

// then save
appManager.writeAndSync(appDescription, "Added new Microservices");

// close the AppManager
appManager.close();
```

Since the *AppManager* also implements the *Closeable* interface you can also use the try-with-resources statement to work on the *AppDescription*.

```
Path path = FileSystems.getDefault().getPath("path/to/git/dir");

try (AppManager appManager = AppManager.open("https://git.ref/", "username", "password", path)) {
    AppDescription appDescription = appManager.getAppDescription();
    //work on the AppDescription & save it with AppManager
    appManager.writeAndSync(appDescription, "Added NFRs");
} catch (AppManagerException | IOException e) {
    e.printStackTrace();
}
```

For further examples please take a look at the test cases in *src/test/java*.

3.3.3 Validation Exception Handling

A *DECIDEValidationException* is usually a wrapper for the underlying validation library validation errors. The Application Controller utilizes the JSON Schema Validator from everit-org [13]. You can easily access the original *ValidationException* if you need more details beyond the main message.

```
try {
    AppDescription appDescription = appManager.getAppDescription();
} catch (DECIDEValidationException e) {
    ValidationException original = (ValidationException)e.getCause();

    // get a list of all sub messages
    List<String> messages = original.getAllMessages();

    // get all sub exceptions. Each one is again a ValidationException
    List<ValidationException> original.getCausingExceptions();

    // getting a fancy pretty printed json structure containing all sub errors
```

```
    String json = original.getJSON().toString(4);

} catch (IOEXception e) {
    e.printStackTrace();
}
```

Please note that *DECIDEValidationException* is also a subclass of *AppManagerException*.

3.3.4 Access to the Deployment History

The Deployment History is represented as a list of *HistoryEntry* elements. A small example on getting the *List<HistoryEntry>* and saving it using the *HistoryManager* (exception handling omitted due to better readability):

```
// First open the DECIDE project
AppManager appManager = AppManager.open(gitRef, username, password, localPath);

// retrieve the history manager object
HistoryManager historyManager = appManager.getHistoryManager();

// get the history
List<HistoryEntry> history = historyManager.getHistory();

// do something with the history

// write back any changes
historyManager.writeAndCommit(history, "commit message");
```

For further examples please take a look at the test cases in *src/test/java*.

3.4 Licensing information

The source code is licensed under the GNU Affero General Public License Version 3. (See also the LICENSE.txt inside the source package)

3.5 Download

The complete source code can be downloaded as a compressed archive file from https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/master/AppController (tag M24)

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 Cloud Service Providers (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 or better defined as dependency if the project is Maven based.

It is important to note that the description of the Application Controller in the DoA [4] and its 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."* [2].

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 4. Application Controller Tasks

Task	Implementation	Explanation
Apply necessary annotations in source code	Not applicable	This is dropped as it turns out it is not needed due to the final approach followed in DECIDE.
Create and apply the corresponding deployment scripts	Not applicable	Script generation has been moved to ADAPT. As explained in D4.1 [2], the deployment configuration scripts are dependent on the technology selected for the ADAPT implementation.
Support of standards	JSON and JSON Schema	All information is stored and specified in JSON and JSON schema format.
Plug-in Adapters to translate topologies	Not applicable	Script generation has been moved to ADAPT. As explained in D4.1 [2], the deployment configuration scripts are

Task	Implementation	Explanation
		dependent on the technology selected for the ADAPT implementation
Hold intelligence wrt. different deployment configurations	Implemented as a Java Library	Documented in this deliverable

5 References

[1] DECIDE, “Deliverable 3.10 - Initial multi-cloud native application controller,” 2018.

[2] DECIDE Consortium, “D4.1 Initial DECIDE ADAPT Architecture,” 2017.

[3] “git - distributed is the new centralized,” [Online]. Available: https://git-scm.com/. [Accessed 2018].

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

[5] DECIDE, “Deliverable D3.8 Intermediate DECIDE OPTIMUS,” 2018.

[6] DECIDE, “Deliverable D2.6 - Initial DECIDE DevOps Framework Integration,” 2017.

[7] DECIDE, “Deliverable 2.2 - Detailed requirements specification V2,” 2018.

[8] DECIDE, “Deliverable 2.5 - DECIDE Detailed architecture,” 2018.

[9] “Mercurial SCM,” [Online]. Available: https://www.mercurial-scm.org/. [Accessed 2018].

[10] Apache, “Apache Subversion,” [Online]. Available: https://subversion.apache.org/. [Accessed 2018].

[11] A. S. Foundation, “Apache Maven project,” [Online]. Available: https://maven.apache.org/. [Accessed 2018].

[12] “JSON Schema,” [Online]. Available: https://json-schema.org/. [Accessed 2018].

[13] everit-org, “JSON Schema Validator,” [Online]. Available: https://github.com/everit-org/json-schema. [Accessed 2018].