Deliverable D3.12

Final multi-cloud native application controller

Editor(s):	Simon Dutkowski (Fraunhofer)
Responsible Partner:	Fraunhofer
Status-Version:	Final – v1.0
Date:	29/05/2019
Distribution level (CO, PU):	CO

Project Number:	GA 731533
Project Title:	DECIDE

Title of Deliverable:	Final multi-cloud native application controller
Due Date of Delivery to the EC:	31/05/2019

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):	Javier Gavilanes (Experis)
Approved by:	All Partners
Recommended/mandatory readers:	WP5, WP4, WP3, WP2

Abstract:	This software deliverable comprises the final multi-cloud native application controller. This final 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	21/05/2019	First draft Version. JSON Schemata in Annex	Simon Dutkowski (Fraunhofer)
v0.2	23/05/2019	Git and Validation chapter	Simon Dutkowski (Fraunhofer)
v0.3	24/05/2019	Revised conclusion	Simon Dutkowski (Fraunhofer)
v0.4	28/05/2019	Document reviewed	Javier Gavilanes (Experis IT)
v0.5	29/05/2019	Incorporate comments	Simon Dutkowski (Fraunhofer)
V1.0	29/05/2019	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 "D3.12: Final multi-cloud native application controller" (software demonstrator) and documents it from a functional and technical perspective. This deliverable is the final of three. This document is the revision of the initial and intermediate documents with the same title [1] [2] 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 [3].

This prototype of Year 3 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. Furthermore, the Year 3 prototype was improved regarding the general git handling, especially implementing a well-chosen git merge strategy, the API design and the validation analysis. 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.

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 [4] handling and possible project structure requirements.

In Year 3 of the project the Application Controller focused on the maintenance of the schemata for the two descriptors Application Description and Deployment History. Additionally, some considerations about the git merge problematic are made and a practical decision for a strategy used within the project context is made.

This document updates, amends and extends D3.11 [2], with changes throughout the whole document, and whose sections have been kept for readability issues. Completely new sections this deliverable are 2.2.2., 3.3.4 and Annex A and B.

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 final conclusion is presented along with deviations from the DoA [5] and some recommendations for further improvements beyond this project.

2 Implementation

2.1 Functional description

Beside the general managing of the Application Description model and the encapsulation of the git [4] 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 [6] 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 [6] 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 [7] 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 [8] 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	Status
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.	Satisfied
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.	Satisfied
F6	WP3-CONTR-REQ1	The multi-cloud native application controller is implemented as a Java Library	Satisfied

Functionality	Req. ID	Coverage	Status
		and can be used by any Java source Code very easily.	
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.	Satisfied

2.1.1 Fitting into overall DECIDE Architecture

The Application Controller library acts as a facilitator for OPTIMUS [6] 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 *1*** depicts the high-level simplified use cases that the Application Controller covers.

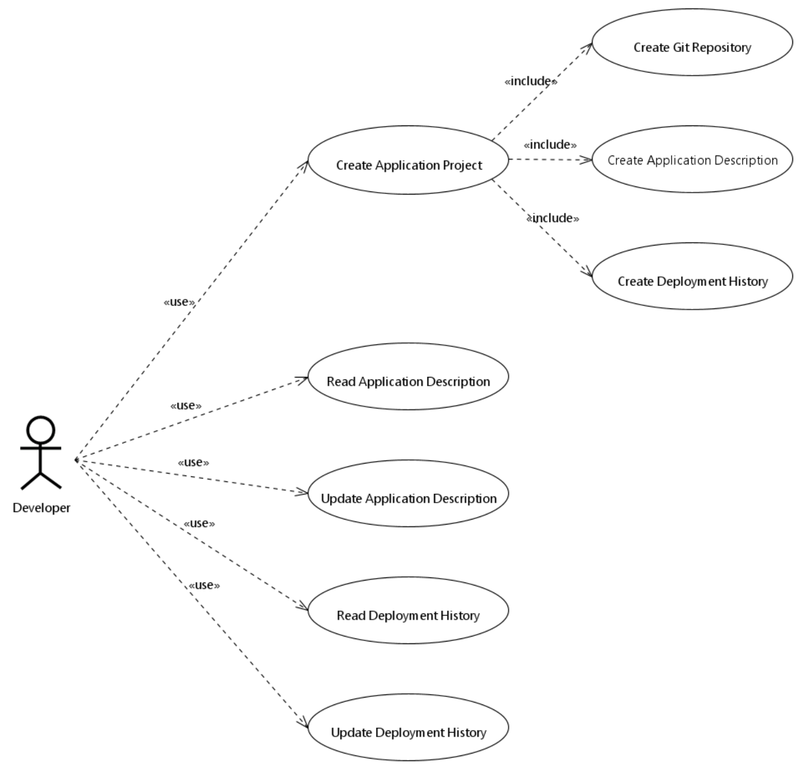
Developer
«use»
«use»
«use»
«use»
«use»
Create Application Project
«include»
«include»
«include»
Create Git Repository
Create Application Description
Create Deployment History
Read Application Description
Update Application Description
Read Deployment History
Update Deployment History

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 based model of the Application Description for easy finding and processing of the contained information.

Update Application Description

The Model can easily be 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 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 easily be 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 [9], the corresponding technical JSON schema can be found in Annex B. 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 *2*** lists all properties of element type *HistoryEntry*. The Deployment History descriptor is simply an array of *HistoryEntry* elements. The complete technical JSON schema description can be found in Annex A.

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 [9] 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 Git merge complex of problems

This chapter describes the potential problems when many tools working in parallel on one descriptor. For simplicity, a single remote repository serving as a single point of truth is assumed. Also, the use of different branches is not part of the examination.

The Application Descriptor and the Deployment History are managed in a git repository. Git repositories by nature are distributed. Each tool has to clone the repository, usually from the specific remote one that serves as the single point of truth, and has to work on its local working copy. There are two main challenges when the tools work with the distributed git repositories. First, they need to make sure that they work on the latest version, which means they probably should make a pull before start a session. After they are done, the changes should be pushed back to the origin of the local clone, so other tools can see the result of the session. Both, pull and push operation are potential sources for merge conflicts because the remote version could be changed between two pulls by other tools. Usually, git should be able to merge the different versions without conflicts, but when there are conflicts, an appropriate strategy needs to be chosen for dealing with the situation.

Technically, git supports several alternative merge strategies:

- *Recursive*
  This is the default merge strategy. Like *resolve* it can only resolve two heads using a 3-way merge algorithm. Without going into details, this is a little bit more “intelligent” merge process that involves common ancestors to apply the 3-way merge algorithm. It is reported that it results in fewer merge conflicts and hence is may be a better candidate than **resolve**. This merge strategy can be further customized with several options. The most interesting are:

    - *Ours*
      When there is a conflict, favouring "ours". This is not the same as the top level "Ours" strategy.
    - *Theirs*
      When there is a conflict, favouring "theirs".
    - *Patience*
      Take some more effort and extra time to avoid mis-merges. Can be helpful when the two heads differ wildly.
- *Resolve*
  It uses a 3-way merge algorithm. It covers the most common use cases where only two heads need to be merged. It tries to carefully detect criss-cross merge ambiguities and is considered generally safe and fast. A great candidate for DECIDE but does not prevent conflicts hundred percent.
- *Octopus*
  This strategy is automatically used when more than two heads are involved. This should not be necessary for the DECIDE workflow.
- *Ours*
  This strategy can also be applied on more than two heads and more or less ignoring any changes except "ours".
- *Subtree*
  An extension of the *recursive* strategy. When merging A and B, if B is a child subtree of A, B is first updated to reflect the tree structure of A, this update is also done to the common ancestor tree that is shared between A and B.

In general, the most appropriate way to handle merge conflicts is to involve the end user and guide him through a decision process where the user will decide how conflicts should be solved. Because this means bigger efforts, which cannot be performed by the project, a more practical solution is needed.

Before choosing the most appropriate merge strategy, one major step to minimize possible merge conflicts is a well-designed structure of the descriptors. It should be ensured that each tool needs to write exclusively in specific parts of the descriptors. Where the deployment history is only used by OPTIMUS, the application description is carefully structured towards this requirement. The remaining critical situation is where a tool is used in parallel by several users on the same project.

Taking the general DECIDE use case workflow and phase model into account, we can assume that the tools are usually used in a sequence order. In most cases there is one centralized repository for exchanging the latest version. So, each tool should start a session by pull the last result from the remote, working on the local copy, and finally submit the session result back to the remote repository. For this scenario, where only two heads need to be merged, the default strategy *recursive* seems to be sufficient. For any remaining conflicts the implementation will force the "theirs" favouring. The tool may give the user a hint that this happened and should immediately present the result to allow a proper reaction if necessary.

2.2.3 Components description

Figure *2* shows the component diagram of the Application Controller. To give more insight of each depicted component in the following each component is briefly described.

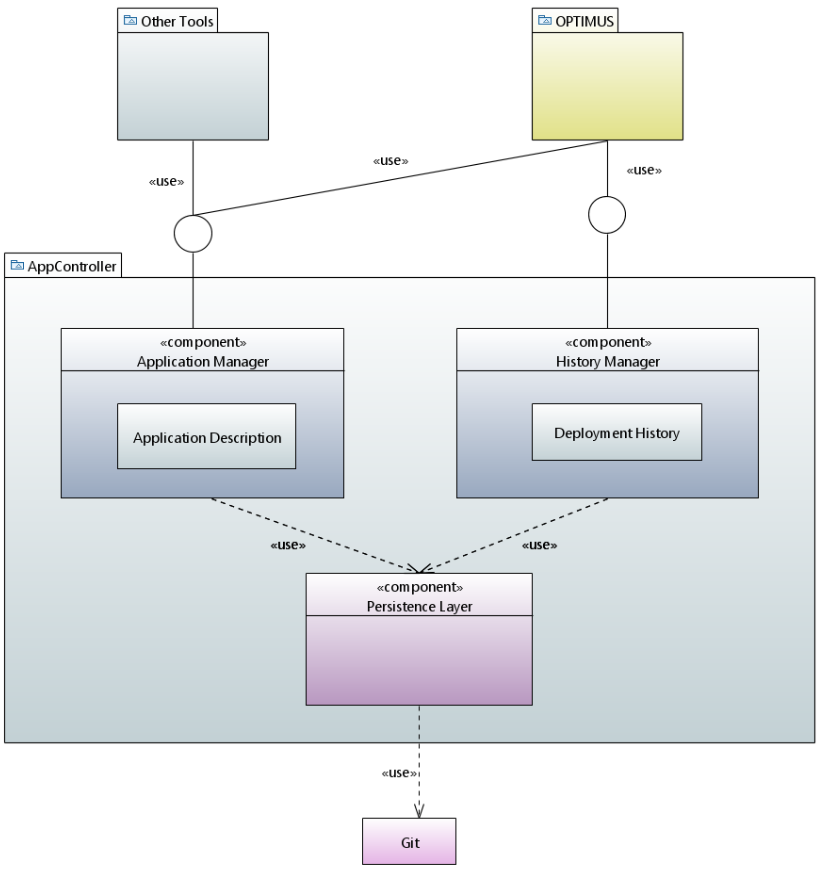

Figure 2. Application Controller Component Diagram

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 [4] operations. This theoretically allows the utilization of other source code management (SCM) technologies, like Mercurial [10] or even Subversion [11].

3 Delivery and usage

3.1 Package information

The Application Controller is implemented as a shared library based on the Apache Maven build tool [12]. 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 [13] 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:

```
$ git clone https://git.code.tecnalia.com/decide/AppController.git
$ cd AppController
```

The project uses Maven as build tool [12]. 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>0.0.16</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 and validating 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 *3***). 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 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*.

3.3.4 Implicit and explicit validation

Default behavior of the library is to validate the descriptors *Application Description* and *Deployment History* against the corresponding JSON schema whenever they passed to the *AppManager* or read from a file. This should help to ensure that no defective structures are stored and therefore could harm any other tools build on these descriptors. The JSON schema descriptions try to be as specific as possible to define each field with a proper type, cardinality, default values, and enumerations. Nevertheless, the library can also be used in a so called "non-strict" mode. This can easily be done by setting the strict mode to false before accessing the *AppDescription* object:

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

// Set strict mode to false
appManager.setStrict(false);
```

Now, no validation takes place at all. In order to validate explicitly from the application logic at any time, use the following static method from the *AppDescriptionFactory*:

```
// Validate an AppDescription object via the factory
AppDescriptionFactory.validateAppDescription(appDescription);
```

The same can be applied to the *HistoryManager*:

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

// Set strict mode to false
historyManager.setStrict(false);

// And for explicit validation of List<HistoryEntry> use the factory
HistoryFactory.validateHistory(historyList);
```

In case of validation errors, methods throw a *DECIDEValidationException*:

A *DECIDEValidationException* is usually a wrapper for the underlying validation library validation errors. The Application Controller utilizes the JSON Schema Validator from everit-org [14]. 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*.

3.4 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 …

In addition, protected access to the git repository can be requested in order to clone or fork directly the repositories. The repository can be found here:

https://git.code.tecnalia.com/DECIDE_Public/

AppController (Application Controller Library) can be found in this directory:
https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/master/AppController

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 [5] 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 and 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 final 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.
Create and apply the corresponding deployment scripts	Not applicable	Script generation has been moved to ADAPT. As explained in D4.1 [3], 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 [3], 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

The Application Controller as a major shared library ensures, when used by the tools, that the main information exchange mechanism, the application description and deployment history stored in a git repository, is always in a valid state. The centralized implementation and maintenance of the application description and deployment history schema together with the encapsulation of implementation details and a convenient API minimize the error rate when processing the DECIDE workflow. Nevertheless, there is room for further improvements especially in regard to the convenient API, a more flexible git merge handling (e.g. selection of merge strategies from the application level) and additional features like an intelligent tag management for relating and grouping elements inside the application description. These improvements are left off to any potential follow up project or any kind of subsequent exploitation.

5 References

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

[2] DECIDE, “Deliverable 3.11 - Intermediate multi-cloud native application controller,” 2018.

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

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

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

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

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

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

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

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

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

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

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

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

Annex A – JSON schema of Deployment History

This Annex contains JSON schema of the history element. The main object (DeploymentHistory) contains an array of history entries (HistoryEntry). Each history entry contains the date of the deployment, the deployment schema (SchemaElement) which is identical to the deployment schema from the application description, and a list of slaBreaches that describes the reasons why this deployment scenario was dropped.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "$id": "http://eu.DECIDEh2020/schemas/optimus-history",
  "title": "DeploymentHistory",
  "description": "DECIDE Deployment History",
  "type": "array",
  "items": {
    "$ref": "#/definitions/HistoryEntry"
  },
  "definitions": {
    "HistoryEntry": {
      "type": "object",
      "required": [ "date", "schema" ],
      "properties": {
        "date": {
          "type": "string",
          "format": "date-time"
        },
        "schema": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/SchemaElement"
          }
        },
        "slaBreaches": {
          "type": "array",
          "items": {
            "type": "object"
          }
        }
      }
    },
    "SchemaElement": {
      "type": "object",
      "properties": {
        "microservices": {
          "type": "array",
          "items": {
            "type": "string",
            "format": "uuid"
          },
          "minItems": 1,
          "uniqueItems": true
        },
        "csId": {
          "type": "string"
        },
        "index": {
          "type": "number"
        }
      }
    }
  }
}
```

Annex B – Application Description JSON schema

This Annex lists the complete Application Description JSON schema.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "AppDescription",
  "description": "DECIDE Application Description",
  "type": "object",
  "required": [
    "id",
    "name",
    "version",
    "highTechnologicalRisk",
    "microservices"
  ],
  "properties": {
    "id": {
      "type": "string",
      "format": "uuid"
    },
    "name": {
      "type": "string"
    },
    "description": {
      "type": "string"
    },
    "version": {
      "type": "string"
    },
    "highTechnologicalRisk": {
      "type": "boolean"
    },
    "jenkinsEndpoint": {
      "type": "string",
      "format": "uri"
    },
    "jenkinsToken": {
      "type": "string"
    },
    "preferredProvider": {
      "type": "string"
    },
    "monitoring": {
      "$ref": "#/definitions/Monitoring"
    },
    "recommendedPatterns": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/Pattern"
      }
    },
    "microservices": {
      "type": "array",
      "minItems": 1,
      "items": {
        "$ref": "#/definitions/Microservice"
      }
    },
    "mcsla": {
      "$ref": "#/definitions/Mcsla"
    },
    "nfrs": {
      "type": "array",
      "items": {
        "anyOf": [
          {
            "$ref": "#/definitions/AvailabilityNfr"
          },
          {
            "$ref": "#/definitions/PerformanceNfr"
          },
          {
            "$ref": "#/definitions/ScalabilityNfr"
          },
          {
            "$ref": "#/definitions/LocationNfr"
          },
          {
            "$ref": "#/definitions/CostNfr"
          }
        ]
      }
    },
    "schema": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/SchemaElement"
      }
    },
    "virtualMachines": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/VirtualMachine"
      }
    },
    "containers": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/Container"
      }
    },
    "applicationInstanceId": {
      "type": "string"
    }
  },
  "definitions": {
    "Microservice": {
      "type": "object",
      "required": [ "id", "endpoints" ],
      "properties": {
        "id": {
          "type": "string",
          "format": "uuid"
        },
        "name": {
          "type": "string"
        },
        "tags": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "uniqueItems": true
        },
        "sourceRepository": {
          "type": "string",
          "format": "uri"
        },
        "deploymentOrder": {
          "type": "number"
        },
        "programmingLanguage": {
          "type": "string"
        },
        "containerId": {
          "type": "string"
        },
        "containerRef": {
          "type": "string"
        },
        "endpoints": {
          "type": "array",
          "minItems": 1,
          "items": {
            "type": "string"
          }
        },
        "stateful": {
          "type": "boolean",
          "default": "false"
        },
        "classification": {
          "type": "string"
        },
        "dependencies": {
          "type": "array",
          "items": {
            "type": "string"
          }
        },
        "safeMethods": {
          "type": "array",
          "items": {
            "type": "string"
          }
        },
        "publicIP": {
          "type": "boolean"
        },
        "infrastructureRequirements": {
          "$ref": "#/definitions/InfrastructureRequirements"
        },
        "detachableResources": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/DetachableResource"
          }
        }
      }
    },
    "Mcsla": {
      "type": "object",
      "required": [ "csSlas" ],
      "properties": {
        "sla": {
          "$ref": "#/definitions/Sla"
        },
        "csSlas": {
          "type": "object",
          "description": "Map between cloud services and Slas. Keys are the cloud service
ids.",
          "additionalProperties": {
            "$ref": "#/definitions/Sla"
          }
        }
      }
    },
    "Nfr": {
      "type": "object",
      "required": [ "type" ],
      "properties": {
        "type": {
          "enum": [ "Availability", "Performance", "Scalability", "Location", "Cost" ]
        },
        "tags": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "uniqueItems": true
        }
      }
    },
    "AvailabilityNfr": {
      "allOf": [
        {
          "$ref": "#/definitions/Nfr"
        },
        {
          "type": "object",
          "required": [ "abstractValue", "value" ],
          "properties": {
            "type": {
              "const": "Availability"
            },
            "abstractValue": {
              "type": "string",
              "enum": [ "Low", "Medium", "High" ]
            },
            "value": {
              "type": "number",
              "minimum": 0,
              "maximum": 100
            },
            "unit": {
              "type": "string"
            }
          }
        }
      ]
    },
    "PerformanceNfr": {
      "allOf": [
        {
          "$ref": "#/definitions/Nfr"
        },
        {
          "type": "object",
          "required": [ "abstractValue", "value" ],
          "properties": {
            "type": {
              "const": "Performance"
            },
            "abstractValue": {
              "type": "string",
              "enum": [ "Low", "Medium", "High" ]
            },
            "value": {
              "type": "number",
              "exclusiveMinimum": 0
            },
            "unit": {
              "type": "string"
            }
          }
        }
      ]
    },
    "ScalabilityNfr": {
      "allOf": [
        {
          "$ref": "#/definitions/Nfr"
        },
        {
          "type": "object",
          "required": [ "abstractValue", "value" ],
          "properties": {
            "type": {
              "const": "Scalability"
            },
            "abstractValue": {
              "type": "string",
              "enum": [ "Low", "Medium", "High" ]
            },
            "value": {
              "type": "number"
            },
            "unit": {
              "type": "string"
            }
          }
        }
      ]
    },
    "LocationNfr": {
      "allOf": [
        {
          "$ref": "#/definitions/Nfr"
        },
        {
          "type": "object",
          "required": [ "abstractValue", "value" ],
          "properties": {
            "type": {
              "const": "Location"
            },
            "abstractValue": {
              "type": "string",
              "enum": [ "Single Location", "Single Country", "Cross Border" ]
            },
            "value": {
              "type": "array",
              "items": {
                "type": "string"
              },
              "uniqueItems": true
            }
          }
        }
      ]
    },
    "CostNfr": {
      "allOf": [
        {
          "$ref": "#/definitions/Nfr"
        },
        {
          "type": "object",
          "required": [ "abstractValue", "value" ],
          "properties": {
            "type": {
              "const": "Cost"
            },
            "abstractValue": {
              "type": "string",
              "enum": [ "Low", "Medium", "High" ]
            },
            "value": {
              "type": "number",
              "exclusiveMinimum": 0
            },
            "unit": {
              "type": "string"
            }
          }
        }
      ]
    },
    "VirtualMachine": {
      "type": "object",
      "properties": {
        "id": {
          "type": "string"
        },
        "cspId": {
          "type": "string"
        },
        "cspName": {
          "type": "string"
        },
        "ram": {
          "type": "integer",
          "minimum": 1
        },
        "cores": {
          "type": "integer",
          "minimum": 1
        },
        "storage": {
          "type": "integer",
          "minimum": 1
        },
        "image": {
          "type": "string"
        },
        "openedPorts": {
          "type": "array",
          "items": {
            "type": "integer",
            "minimum": 0
          }
        }
      }
    },
    "Container": {
      "type": "object",
      "required": [
        "imageName",
        "imageTag",
        "hostname",
        "restart",
        "dockerHostNodeName"
      ],
      "properties": {
        "containerName": {
          "type": "string"
        },
        "containerRef": {
          "type": "string"
        },
        "imageName": {
          "type": "string"
        },
        "imageTag": {
          "type": "string"
        },
        "dockerPrivateRegistryIp" : {
          "oneOf": [
            {
              "type": "string",
              "format": "ipv4"
            },
            {
              "type": "string",
              "format": "ipv6"
            }
          ]
        },
        "dockerPrivateRegistryPort": {
          "type": "integer"
        },
        "dockerPrivateRegistryUser": {
          "type": "string"
        },
        "dockerPrivateRegistryPassword": {
          "type": "string"
        },
        "hostname": {
          "type": "string",
          "format": "hostname"
        },
        "restart": {
          "type": "string",
          "enum": [ "always" ]
        },
        "command": {
          "type": "array",
          "items": {
            "type": "string"
          }
        },
        "entrypoint": {
          "type": "array",
          "items": {
            "type": "string"
          }
        },
        "dockerHostNodeName": {
          "type": "string"
        },
        "networks": {
          "type": "array",
          "items": {
            "type": "string"
          }
        },
        "volumeMapping": {
          "type": "array"
        },
        "environment": {
          "type": "array",
          "items": {
            "type": "string"
          }
        },
        "consulKvProviderNodeName": {
          "type": "string"
        },
        "addConsulService": {
          "type": "integer"
        },
        "addConsulTraefikRules": {
          "type": "integer"
        },
        "portMapping": {
          "type": "array",
          "items": {
            "type": "object",
            "required": [ "hostPort", "containerPort" ],
            "properties": {
              "hostPort": {
                "type": "string"
              },
              "containerPort": {
                "type": "string"
              }
            }
          }
        },
        "endpoints": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Endpoint"
          }
        }
      }
    },
    "InfrastructureRequirements": {
      "type": "object",
      "required": [ "minDisk", "maxDisk", "minRam", "maxRam" ],
      "properties": {
        "cpuCores": {
          "type": "integer"
        },
        "minDisk": {
          "type": "integer"
        },
        "maxDisk":  {
          "type": "integer"
        },
        "minRam": {
          "type": "integer"
        },
        "maxRam":  {
          "type": "integer"
        }
      }
    },
    "DetachableResource": {
      "type": "object",
      "required": [ "id", "name" ],
      "properties": {
        "id": {
          "type": "string"
        },
        "name": {
          "type": "string"
        },
        "db": {
          "type": "boolean",
          "default": false
        },
        "sql": {
          "type": "boolean",
          "default": false
        },
        "specificDb": {
          "type": "string"
        },
        "size": {
          "type": "string",
          "enum": [ "small", "medium", "large" ]
        },
        "classification": {
          "type": "string",
          "enum": [ "db", "storage", "queue system" ]
        }
      }
    },
    "Pattern": {
      "type": "object",
      "required": [ "title", "selected" ],
      "properties": {
        "title": {
          "type": "string"
        },
        "uriRef": {
          "type": "string",
          "format": "uri"
        },
        "positiveImpacts": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "uniqueItems": true
        },
        "categories": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "uniqueItems": true
        },
        "tags": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "uniqueItems": true
        },
        "selected": {
          "type": "boolean"
        }
      }
    },
    "Sla": {
      "type": "object",
      "required": [ "description" ],
      "properties": {
        "description": {
          "type": "string"
        },
        "visibility": {
          "type": "string"
        },
        "validityPeriod": {
          "type": "integer",
          "minimum": 0
        },
        "coveredServices": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "minItems": 1,
          "uniqueItems": true
        },
        "objectives": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/ServiceObjective"
          }
        }
      }
    },
    "ServiceObjective": {
      "type": "object",
      "required": [ "termName" ],
      "properties": {
        "type": {
          "type": "string",
          "enum": [ "slo", "sqo" ]
        },
        "termName": {
          "type": "string"
        },
        "comment": {
          "type": "string"
        },
        "value": {
          "type": "string"
        },
        "unit": {
          "type": "string"
        },
        "conditionStatement": {
          "type": "string",
          "enum": [ "greater", "less", "greaterOrEqual", "lessOrEqual", "equal" ]
        },
        "violationTriggerRules": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/ViolationTriggerRule"
          }
        },
        "metrics": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Metric"
          }
        },
        "remedy": {
          "$ref": "#/definitions/Remedy"
        }
      }
    },
    "Metric": {
      "type": "object",
      "required": [ "id", "scale", "expression" ],
      "properties": {
        "descriptor": {
          "type": "string"
        },
        "id": {
          "type": "string"
        },
        "source": {
          "type": "string"
        },
        "scale": {
          "type": "string",
          "enum": [ "nominal", "ordinal", "interval", "ratio" ]
        },
        "note": {
          "type": "string"
        },
        "category": {
          "type": "string"
        },
        "expression": {
          "$ref": "#/definitions/Expression"
        },
        "parameters": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Parameter"
          }
        },
        "rules": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Rule"
          }
        },
        "underlyingMetrics": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Metric"
          }
        }
      }
    },
    "Expression": {
      "type": "object",
      "required": [ "expression", "expressionLanguage" ],
      "properties": {
        "id": {
          "type": "string"
        },
        "expression": {
          "type": "string"
        },
        "expressionLanguage": {
          "type": "string"
        },
        "note": {
          "type": "string"
        },
        "unit": {
          "type": "string"
        },
        "subExpressions": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Expression"
          }
        }
      }
    },
    "Parameter": {
      "type": "object",
      "required": [ "id", "parameterStatement", "unit" ],
      "properties": {
        "id": {
          "type": "string"
        },
        "parameterStatement": {
          "type": "string"
        },
        "unit": {
          "type": "string"
        },
        "note": {
          "type": "string"
        }
      }
    },
    "Rule": {
      "type": "object",
      "required": [ "id", "ruleStatement", "ruleLanguage" ],
      "properties": {
        "id": {
          "type": "string"
        },
        "ruleStatement": {
          "type": "string"
        },
        "ruleLanguage": {
          "type": "string"
        },
        "note": {
          "type": "string"
        }
      }
    },
    "ViolationTriggerRule": {
      "type": "object",
      "properties": {
        "breachesCount": {
          "type": "number"
        },
        "violationInterval": {
          "type": "number"
        }
      }
    },
    "Remedy": {
      "type": "object",
      "required": [ "type", "unit", "value", "validity" ],
      "properties": {
        "type": {
          "type": "string"
        },
        "unit": {
          "type": "string"
        },
        "value": {
          "type": "number"
        },
        "validity": {
          "type": "string"
        }
      }
    },
    "SchemaElement": {
      "type": "object",
      "properties": {
        "microservices": {
          "type": "array",
          "items": {
            "type": "string",
            "format": "uuid"
          },
          "minItems": 1,
          "uniqueItems": true
        },
        "csId": {
          "type": "string"
        },
        "index": {
          "type": "number"
        }
      }
    },
    "Monitoring": {
      "type": "object",
      "properties": {
        "status": {
          "type": "boolean"
        },
        "urls": {
          "type": "array",
          "items": {
            "type": "string",
            "format": "uri"
          }
        }
      }
    },
    "Endpoint": {
      "type": "object",
      "required": [ "protocol", "port" ],
      "properties": {
        "protocol": {
          "type": "string"
        },
        "port":  {
          "type": "integer",
          "minimum": 0
        },
        "skipRule": {
          "type": "integer"
        },
        "containerNameOverride": {
          "type": "string"
        }
      }
    }
  }
}
```