Deliverable D4.6

Final multi-cloud application deployment and adaptation

Editor(s):	Paolo Barone (HPE)
Responsible Partner:	HPE
Status-Version:	Final – V1.0
Date:	31/05/2019
Distribution level (CO, PU):	PU

Project Number:	GA 726755
Project Title:	DECIDE

Title of Deliverable:	Final multi-cloud application deployment and adaptation
Due Date of Delivery to the EC:	31/05/2019

Workpackage responsible for the Deliverable:	WP4 - Continuous deployment and operation
Editor(s):	Paolo Barone (HPE)
Contributor(s):	Paolo Barone (HPE)
Reviewer(s):	Simon Dutkowski (FhG)
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP5, WP6

Abstract:	This final version of the deliverable is the continuation of deliverable D4.5 [1]. It finalizes the methods and the tools for implementing the automatic deployment and adaptation of multi-cloud applications and provides mapping between the requirements identified in the architecture for M30 (D4.3 [2]) and the related use cases implementation in the ADAPT DO component.
Keyword List:	Microservice, REST, container, virtual machine, orchestration system, adaptation, architecture, installation, packaging, usage, private cloud, hybrid cloud.
Licensing information:	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 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	17/05/2019	First draft version	Paolo Barone (HPE)
V0.2	22/05/2019	Updated version	Paolo Barone (HPE)
v0.3	24/05/2019	Version submitted to reviewer	Paolo Barone (HPE)
V0.4	27/05/2019	Version containing reviewer comments	Simon Dutkowski (Fraunhofer)
V0.5	27/05/2019	Version addressing reviewer comments	Paolo Barone (HPE)
V0.6	27/05/2019	Version submitted to Project Coordinator	Paolo Barone (HPE)
V1.0	27/05/2019	Ready for submission	Leire Orue-Echevarria (TECNALIA)

Table of Contents

List of Figures

List of Tables

Terms and abbreviations

ACSmI	Advanced Cloud Service meta-Intermediator
ADAPT DO	ADAPT Deployment Orchestrator
API	Application Programming Interface
AWS	Amazon Web Services
CA	Certification Authority
CLI	Command Line Interface
CSP	Cloud Service Provider
DevOps	Development and Operations
EC	European Commission
GA	Grant Agreement
GB	Giga Bytes
GUI	Graphic User Interface
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KR	Key Result
MM	Monitoring Manager
REST	Representational State Transfer
SSH	Secure SHell
TLS	Transport Layer Security
UI	User Interface
URL	Unified Resource Locator
UC	Use Case
VH	Violation Handler
VM	Virtual Machine
WP	Work Package

Executive Summary

This deliverable, related to the final multi-cloud application deployment and adaptation, is to be considered an extension of the D4.5 [1], which documented the intermediate implementation of the ADAPT Deployment Orchestrator (ADAPT DO) at M24. We follow the same approach adopted for D4.5 [1]: considering that the relationship with the architecture defined in D4.1 [3] and D4.2 [4] (as well as the internal design, the execution behavior and the dynamic generation of configuration scripts for the automated deployment) were already in a consolidated state at M24, to avoid huge repetitions between different versions of the document we will not report here the concepts already documented in D4.4 [5] and D4.5 [1], as they would be identical. Instead*, this document focuses on the relationship between the requirements and use cases expected by M30 for ADAPT DO and documented in the deliverable D4.3* [2], the document describing the final architecture for the ADAPT component.

Just to recap the main concepts, only in this executive summary we recall from D4.4 [5] and D4.5 [1] that the ADAPT DO is a component within ADAPT that is in charge of orchestrating the deployment lifecycle (deployment, undeployment, user confirmation, redeployment) for the represented user application and its components. It takes as main input the *Application Description* file, that is the main document shared between the DECIDE components and that each component uses/enriches with specific data during the DECIDE workflow execution.

As the Application Description is the main data exchange mechanism between DECIDE components, it is fundamental to maintain coherence and correctness in its format and contents. Therefore, a proper schema for the contents of the file has been defined (and is constantly updated) within the scope of WP3. To make sure that ADAPT DO is using a valid Application Description as input, and to grant the modifications or updates made by ADAPT DO to the file are compliant with the schema a validation mechanism has been added to the ADAPT DO.

A dedicated area in the DECIDE DevOps framework (which is the GUI part of the DECIDE framework) provides a wizard which drives the end user in setting up all the input data required for the application deployment. With respect to M24, where only deployment progress status was shown to the user, for the final integrated release of the DECIDE platform new tabs have been designed in WP4. They must enable end users to insert relevant data required for the deployment and running of the applications. The design for such tabs is documented in the remainder of this document. This extended dashboard provides 4 steps that allow the end user to fill in configuration data, to upload files required for identity and access verification, and to get a set of textual and visual information about the progress and status of the deployment activities. The design of such GUI is implemented as part of the integration tasks in WP2, responsible for the DevOps framework.

ADAPT DO interacts with the ACSmI component, to start and release cloud resources when needed. In the project lifecycle the set of Cloud Service Providers (CSPs) supported and the type of cloud deployment scenarios allowed have been constantly extended. While for the M12 release it was possible to leverage only two public providers (AWS and Cloudsigma), since the M24 release we can leverage also one additional public CSP (Arsys) and, more importantly, also Openstack private clouds. This is important because it allows ADAPT DO to demonstrate not only public cloud scenarios, but also hybrid cloud or private cloud scenarios, which are much closer to the cloud paradigms considered at enterprise-level.

As defined by the related Amendment, following the mid-term review and in accordance with the Reviewers, the documentation for the “Helpers” components, initially described in a dedicated deliverable D4.10 [6], have been included in D4.5 [1]. Such components have not undergone major changes, therefore the dedicated section 3.4 of D4.5 [1] must be considered the reference documentation.

Regarding the implementation and the packaging aspects, ADAPT DO has not changed: it is still implemented as a microservice, packaged into a container image and running as a container, exposing a REST API to communicate with external components. It was implemented by combining, customizing and even extending a set of existing open source tools/APIs: Flask[1], Werkzeug[2], Jinja 2[3], Flask-RESTplus[4] and Terraform[5].

Being packaged as a Docker image, there are no specific steps to install ADAPT DO. The only requirement is having access to a Linux-based system running a Docker daemon. The image can be found on an unofficial private repository, set up for DECIDE, or extracted by a .tar archive released with the software, or even built easily from the code. All the three options are described in this document.

Regarding the usage of the component, once started, ADAPT DO provides endpoints for interacting with other DECIDE components. For evaluation purposes, it is possible to feed said endpoints manually, but since this is a complex operation requiring specific knowledge, a set of CLI tools has also been released, to help end users who want to experience the ADAPT DO capabilities as a standalone tool.

Preliminary activities for Use Cases evaluation have been performed as joint activity between WP4 and WP6, leveraging the CLI tools, and the results will be reported in the dedicated WP deliverables.

To successfully test the prototype, a user must have credentials for the Git repository where the Application Description is hosted and for the cloud broker in which resources will be provisioned.

[1] http://flask.pocoo.org/
[2] http://werkzeug.pocoo.org/
[3] http://jinja.pocoo.org/
[4] https://flask-restplus.readthedocs.io/en/stable/
[5] https://www.terraform.io/

1 Introduction

1.1 About this deliverable

This deliverable focuses on the implementation at M30 of the Deployment Orchestrator part of the DECIDE ADAPT component (ADAPT DO), which takes care of the multi-cloud application deployment and adaptation functionalities. The final architecture of the whole DECIDE ADAPT component, which is taken as the reference for the implementation of ADAPT DO, has been described in the deliverables D4.1 (initial) [3], D4.2 (intermediate) [4] and D4.3 (final) [2] along with the relevant requirements mapping.

This document is related to the DECIDE Task T4.2 in WP4, whose main outcome is a software deliverable; therefore, this is a report accompanying the released software code at M30. In particular, it describes how the requirements and use cases expected by M30 for ADAPT DO (and documented in the deliverable D4.3 [2], the document describing the final architecture for the ADAPT component) were fulfilled.

1.2 Notes for the reader

To avoid repetitions between the deliverables related to T4.2 (D4.4 [5], D4.5 [1] and the current D4.6), this document will *not* describe (unless changed since the previous versions):

- the ADAPT DO architecture
- the implementation details of the related software components
- the use case and component diagrams
- the sequence of interactions between components and actors

Their design and implementation details were already in a consolidated state at M12 and M24 and were documented accordingly in the corresponding deliverables D4.4 [5], and D4.5 [1] for the related extensions.

It will rather focus on the mapping with the requirements for M30 and must therefore be considered as a continuation of D4.5 [1].

As a consequence, to have a full understanding about ADAPT DO, the related architecture, the details about the implementation and the technologies used, D4.4 [5] and D4.5 [1] are pre-requisites for D4.6 and all documents must be considered.

The exception to the above approach is Section 3, describing a set of CLI utilities, and Section 4, which is an operating manual describing how to get, set up and launch the ADAPT DO component. We believe it is worth keeping a self-contained section for them, even if with some repetitions from D4.4 [5] and D4.5 [1], to support end users who need a reference to follow step-by-step instructions.

In case some contents in D4.6 invalidates and fully replaces specific contents of previous D4.5 [1], we will highlight it clearly in the document.

1.3 Document structure

After the Introduction of the current Section 1, Section 2 of the document describes how the requirements, defined in the architecture and described in D4.3 [2], for ADAPT DO at M30 are mapped to use case implementations.

Section 3 describes a set of CLI tools which allow to run ADAPT DO functionalities as a standalone tool, which is needed in a complex framework like DECIDE during intermediate project phases, where not

all the components are fully integrated. These tools were used to perform a preliminary Use Case evaluation phase in collaboration with WP6.

Since D4.5 [1] is a software deliverable, it also provides details in Section 4 about the released software: how it's structured, how it can be installed and used.

2 Requirements mapping and implementation for M30

This section summarizes the requirements identified for ADAPT DO as reported in D4.3 [2], the document related to the ADAPT final architecture.

First, we list for every requirement the expected deadline and the mapping with the actual status. Then, we report in dedicated sections how we addressed the requirement expected for M30 with the related mapping to use case definitions in D4.3 [2].

2.1 Requirements mapping for ADAPT DO

Table 1 below is an excerpt from Table 1 in D4.3 [2], from which we extracted the requirements specifically related to ADAPT DO.

Table 1. Requirements mapping for ADAPT DO (excerpt from D4.3)

Req. ID	Description	Component	Deadline	Status
WP4-MR1	DECIDE ADAPT will support the semi-automatic adaptation and dynamic re-deployment of (parts of) multi-cloud applications when certain conditions are not met, by changing the configuration and topology of services at operational time based on continuous monitoring of both the conditions of the application and the CSPs where the application is deployed on	DO, MM, VH	M24	Implemented. Testing integration
WP4-MR4	DECIDE ADAPT will support automated dynamic deployment of service components	DO	M12	Satisfied
WP4-MR5	DECIDE ADAPT will generate scripts for automating both resource provisioning and deployment for multi-cloud native applications	DO	M12	Satisfied
WP4-MR6	**ADAPT will support manual checking of the deployment configuration and scripts**	**DO**	**M30**	**Wizard / dashboard being integrated**
WP4-MR7	ADAPT will maintain the current deployment configuration situation; other tools will maintain the history of the previous deployment configurations, so that they can be checked in the re-deployment phase	DO	M12	Satisfied
WP4-MR8	In case the technological risk for the application has been defined as low, the multi-cloud application will be redeployed automatically, following a new deployment configuration [provided by triggering OPTIMUS].	VH, DO	M24	Implemented. Testing integration

Req. ID	Description	Component	Deadline	Status
WP4-MR11	ADAPT functionalities (deployment, monitoring and adaptation) rely on information about the multi-cloud application obtained from the Application Descriptor	DO, MM, VH	M12 - M24	Satisfied. Application Description still being extended
WP4-MR12	ADAPT will support applications based on composition of stateless (possibly micro) services	DO	M12	Satisfied
WP4-MR13	ADAPT will support composable applications where each composition unit is a containerized service	DO	M12	Satisfied
WP4-MR15	Users will perceive relevant improvements in the business continuity since as soon as there is a problem (i.e. lack of resource due to a peak of requests) the software is automatically re-adapted and re-deployed	DO	M24	Implemented. Testing integration
WP4-MR16	DECIDE ADAPT will support the continuous deployment of the developed apps	DO	M24	Satisfied

It can be seen that the only requirement expected for M30 is **WP4-MR6,** the one listed in bold characters, while all the other ones expected for previous phases were already satisfied.

Next sections describe how ADAPT DO was extended to satisfy the **WP4-MR6** requirement

2.2 Use Cases for ADAPT DO M30 requirements

Requirement **WP4-MR6** states that *"ADAPT will support manual checking of the deployment configuration and scripts".*

To satisfy such requirement, it is necessary to provide visual input and visual reporting components to ADAPT DO. The DECIDE framework is based on the integration of several sub-components, orchestrated by the DevOps framework, and the integrated GUI of the platform is provided by the web interface of the DevOps framework itself. Therefore, to satisfy the above-mentioned requirement, it is necessary to provide features on both the ADAPT DO side and the DevOps framework GUI side.

In this document we report how ADAPT DO provides the back-end features required for the triggering and checking of the deployment status, and the specifications provided to WP2 for the development of visual components to be accessed by the end users. Such visual components are being developed and integrated within WP2 activities.

The requirement has been detailed in D4.3 [2] in form of Use Cases Diagram, reported for sake of convenience also in **Figure *1***. All the use cases are addressed in next section.

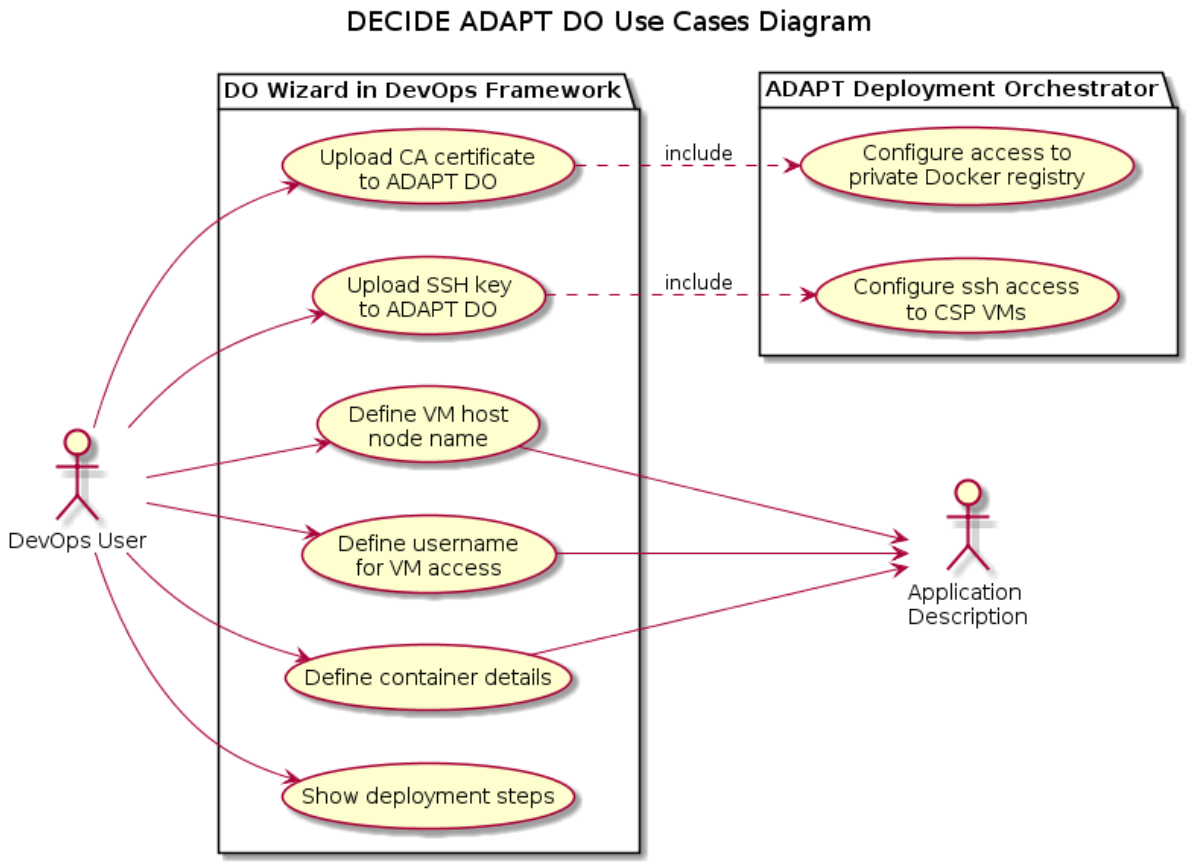

Figure 1. ADAPT DO Use Cases Diagram

2.3 Mapping Use Cases to implementation

In this section we map each use case required for satisfying requirement **WP4-MR6** to the related implementation items.

2.3.1 UCDO301 – Configure access to private Docker registry

UCDO301 description from D4.3:

"*A private Docker registry is a repository from which ADAPT can download container images which are not public. If the private registry uses a self-signed certificate to prove its identity when establishing a TLS session, every client must trust this certificate. Therefore, to download container images from such a private registry to a node, that node must trust the registry certificate as a trusted Certification Authority (CA). A dedicated REST API call will allow to upload the registry's certificate file (ca.crt) to ADAPT DO, so that it can install it as a trusted CA on every node which should run containers downloaded from that private registry.*"

Implementation:

ADAPT DO implements a REST endpoint (cf. **Figure *2***) which allows uploading a certificate file into the ADAPT DO runtime. Such certificate will be properly transferred and stored into a dedicated folder. Upon a deployment request, after starting up the set of VMs specified by the Application Descriptor, such certificate will be copied by ADAPT DO on the VM(s) in a specific folder, expected by the Docker

runtime (typically /etc/docker/certs.d/[registryIp:port]. This allows the Docker runtime to access the remote repository.

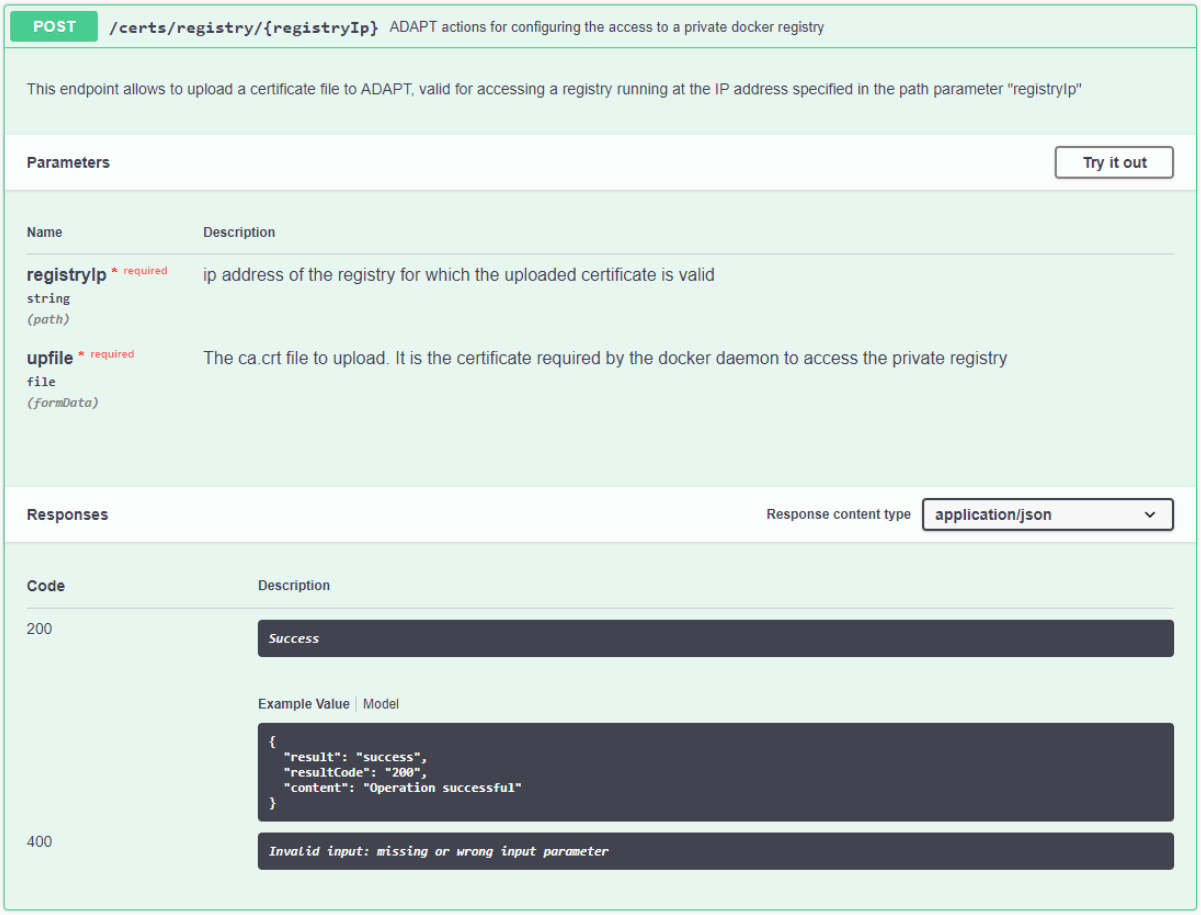

Figure 2. UCDO301 implementation

2.3.2 UCDO302 – Configure ssh access to CSP VMs

UCDO302 description from D4.3 [2]**:**

"To allow ADAPT to configure resources created by a Cloud Service Provider (CSP), it is necessary that it can access such resources via ssh authentication. To enable this, ADAPT must have access to the private SSH key related to the profile of the user for the CSP. A dedicated REST API call will allow to upload to ADAPT such a private SSH key."

Implementation:

ADAPT DO implements a REST endpoint (cf. **Figure *3***) which allows uploading a file representing an ssh key into the ADAPT DO runtime. Such file will be properly transferred and stored into a dedicated folder. Upon a deployment request, after starting up the set of VMs specified by the Application Descriptor, ADAPT DO will use this key to password-less connect to the VM started on a CSP and perform automated configuration and installation steps. The user for the connection is specified in the Application Descriptor.

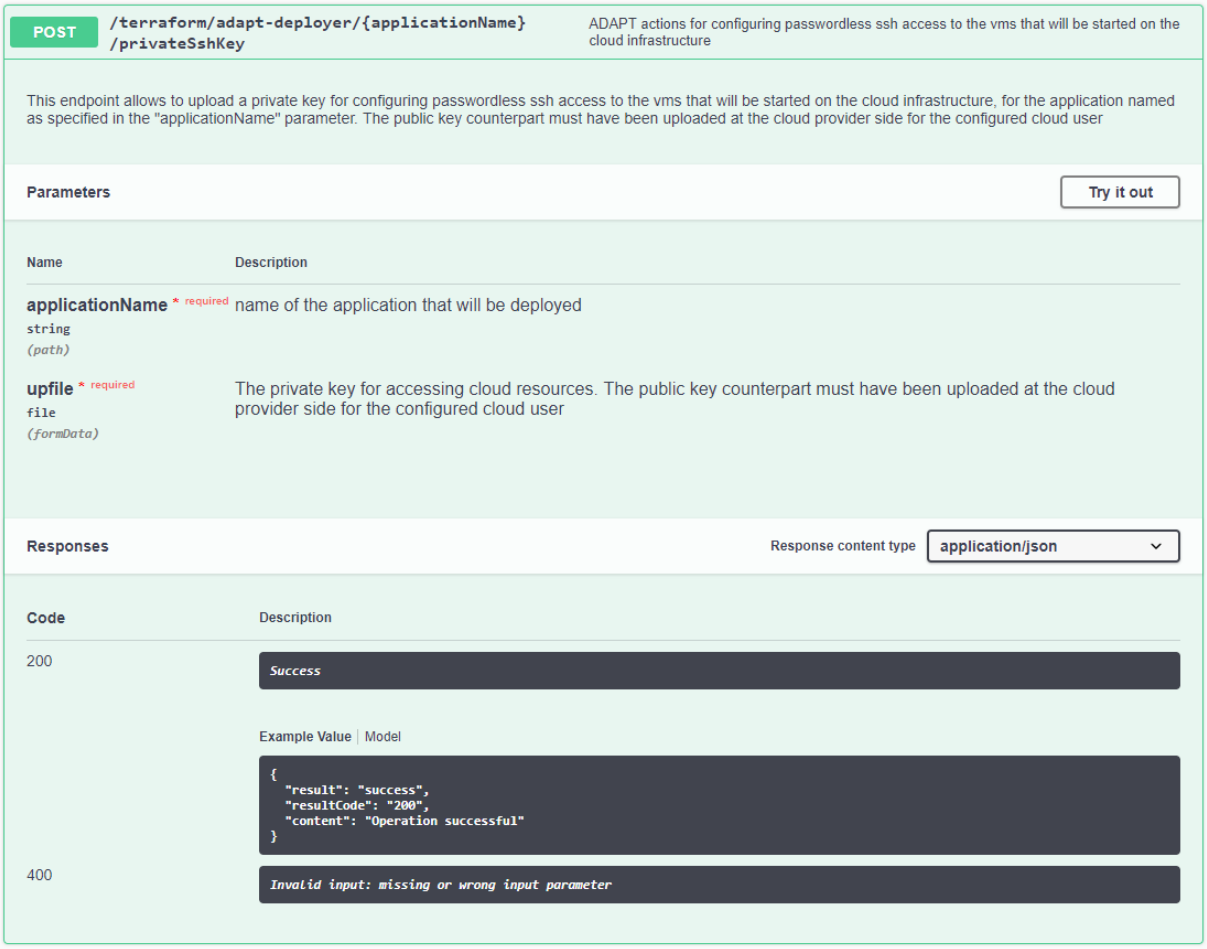

Figure 3. UCDO302 implementation

2.3.3 UCWI301 – Upload CA certificate to ADAPT DO

UCWI301 description from D4.3 [2]**:**

"The DO Wizard user interface within DevOps Framework, in one of its configuration steps, will allow to select a private registry (previously inserted from user profile management) from a combo box and to upload its public certificate to ADAPT DO, using use case UCDO301."

Implementation:

This use case is satisfied by the DevOps framework implementation in WP2, while we report here the specification provided to WP2 for the expected UI components. It is strictly connected to *UCDO301,* as it allows a user to trigger that REST API endpoint to upload a certificate for accessing a private registry.

Figure *4* highlights the section of the UI specification where the selection and upload of a certificate for a specific private registry happens. This is in a Wizard, which drives the user in setting the required/allowed information related to the application deployment following four steps. Step 1 is dedicated to the setting of general information, pre-requisite for the infrastructure provisioning and application deployment steps.

One pre-requisite is that one or more certificates have been configured in the user-profile area of an end user via a dedicated section of the DevOps framework GUI and stored in a vault component.

An end user will then be able to select from a drop down list a registry for which he wants to upload a certificate to ADAPT DO. After that, it is possible to submit the upload certificate request by pressing the proper button.

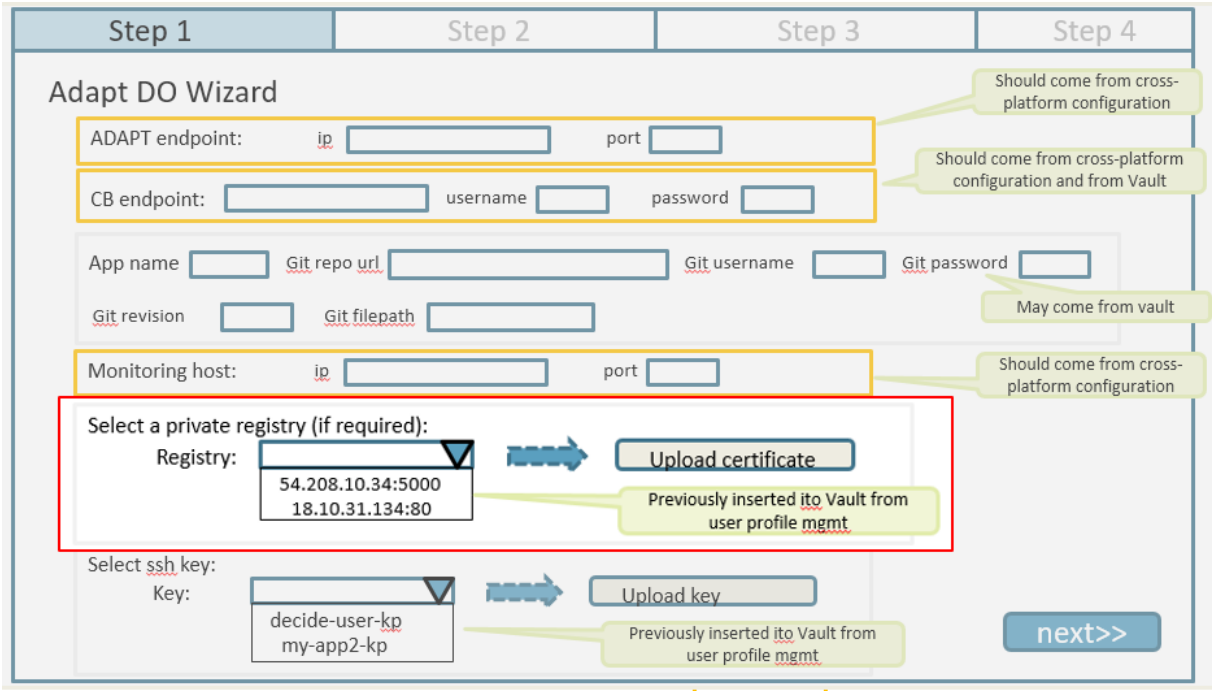

Figure 4. UI specification for UCWI301

2.3.4 UCWI302 – Upload SSH key to ADAPT DO

UCWI302 description from D4.3 [2]**:**

"The DO Wizard user interface within DevOps Framework, in one of its configuration steps, will allow to upload to ADAPT DO the private SSH keys related to the profile of the user for the CSPs that should be accessed during the user's application deployment. The actual upload will be done calling use case UCDO302."

Implementation:

As for the previous use case, also this one is satisfied by the DevOps framework implementation in WP2, while we report here the specification provided to WP2 for the expected UI components. It is strictly connected to *UCDO302,* as it allows a user to trigger that REST API endpoint to upload a file containing a private ssh key for accessing a VM started on a CSP password-less.

Figure *5* highlights the section of the UI specification, in Step 1 of the Wizard, where the selection and upload of a ssh key owned by a user happens.

The pre-requisite is that one or more ssh keys have been configured in the user-profile area of an end user via a dedicated section of the DevOps framework GUI and stored in a vault component.

An end user will then be able to select from a drop down list a ssh key that he wants to upload to ADAPT DO. After that, it is possible to submit the upload request by pressing the proper button.

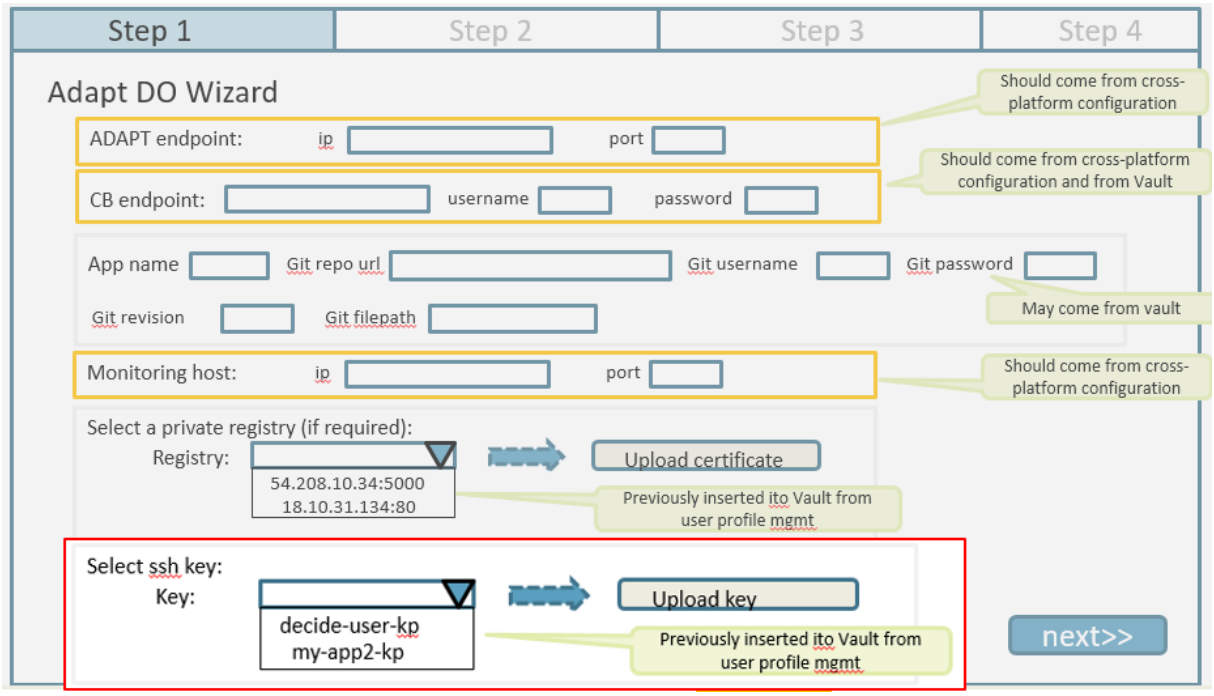

Figure 5. UI specification for UCWI302

2.3.5 UCWI303 – Define VM host node name

UCWI303 description from D4.3 [2]**:**

"The DO Wizard user interface within DevOps Framework, in one of its configuration steps, will allow the user to give a meaningful name to each of the resources (currently Virtual Machines) provided by ACSmI contracting for deploying the user's application. Each name specified by the user will be written in the Application Description as the value of the key "dockerHostNodeName" for the related VM resource."

Implementation:

This use case is satisfied by the DevOps framework implementation in WP2, while we report here the specification provided to WP2 for the expected UI components.

In Step 2 of the Wizard (cf. **Figure *6***), a set of form fields allows the end user specifying information related to the infrastructure items to be provisioned on the CSPs, namely:

- A name for each VM to be started
- A username, required to allow ADAPT DO to connect password-less (via ssh key) to the VM to automatically configure it
- The set of ports to be opened for remote access. Such ports will be either used for administrative services (e.g. ssh access, registry services like consul, etc.), or by the containers to expose ports

The fields filled in the forms will be then added to the Application Description file, in the section dedicated to the virtual machines' definition (cf. the bottom-right area of the figure below).

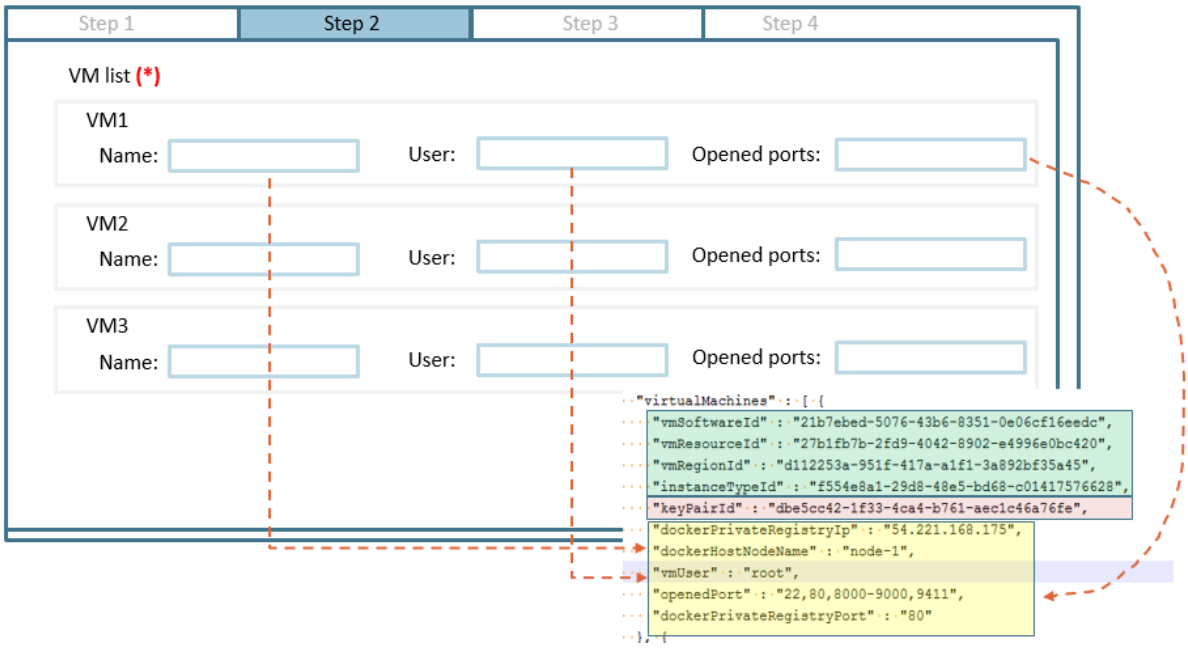

Figure 6. UI specification for UCWI303

2.3.6 UCWI304 – Define username for VM access

UCWI304 description from D4.3 [2]**:**

"When ADAPT DO accesses the VM resources to configure them and deploy the application, it must do that using credentials with the right privileges. The DO Wizard user interface within DevOps Framework, in one of its configuration steps, will allow the user to specify the username (usually root for Linux) that DO should use when accessing VMs. Each username specified by the user will be written in the Application Description as the value of the key "vmUser" for the related VM resource".

Implementation:

This use case is addressed together with UCWI303 and documented in the previous section 2.3.5 and **Figure *6***.

2.3.7 UCWI305 – Define container details

UCWI305 description from D4.3 [2]**:**

"ADAPT DO needs to know some details about the containers to be deployed. At least the container name, its image name:tag and the repository from which to download it must be specified; additionally also the (optional) mapping between network ports on the container and the host can be specified. An optional reverse proxy with a configuration based on a Key/Value store could also be added to (at most) one container to allow dynamical routing of API calls even when containers are redeployed on different nodes. The DO Wizard user interface within DevOps Framework, in one of its configuration steps, will allow the user to specify the details listed above and will write them in the Application Description using the keys "containerName", "imageName", "imageTag", "dockerPrivateRegistryIp", "dockerPrivateRegistryPort", "portMapping" (array of pairs "hostPort" "containerPort")".

Implementation:

The requirement is satisfied by the DevOps framework implementation in WP2, while we report here the specification provided to WP2 for the expected UI components.

In Step 3 of the Wizard (cf. **Figure *7***), a set of form fields allows the end user specifying information related to the containers to be launched on the VMs, namely:

- The Container Name
- The image name and tag
- The private registry ip (or address name) and port
- The port mapping required by the container (set of ports local to containers to be mapped to corresponding ports of the VM)
- A flag for specifying the usage of a reverse proxy in front of the set of containers to allow unique entry point to the application and enable automatic load balancing and low-latency failover (advanced feature)

The fields filled in the forms will be then added to the Application Description file, in the section dedicated to the containers definition (cf. the bottom-right area of **Figure *7***).

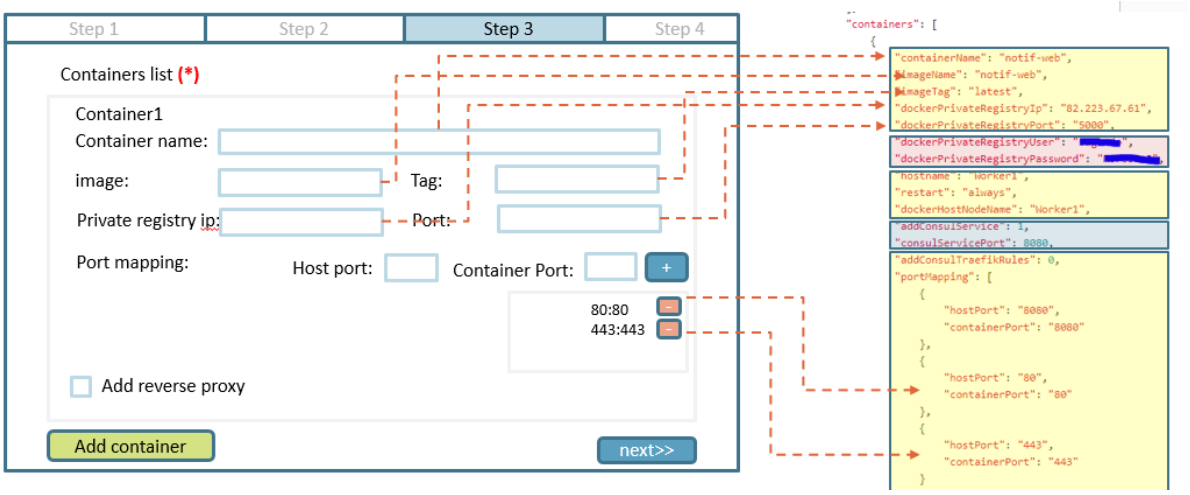

Figure 7. UI specification for UCWI305

2.3.8 UCWI306 – Show deployment steps

UCWI306 description from D4.3 [2]**:**

"The deployment process is composed of three steps (init, plan, apply) for provisioning and configuring the needed infrastructure, and then the same steps applied to the application, for deploying and starting its containers. The process may take some time and the user should be informed about its progress and the result of each phase. The DO Wizard user interface within DevOps Framework, will allow the user to see the progress of the deployment process and the status of each of the above-mentioned steps".

Implementation:

This requirement was indeed already implemented for M24 and documented in D4.5 [1]. ADAPT DO provides a visual Dashboard in the DevOps framework to inform the end user about the status of the deployment actions. This Dashboard is part of Step 4 of the Wizard specified in the previous sections.

We summarize in the following, just for the sake of completeness, how the deployment status is reported to the user.

Figure *8* shows a set of buttons, together with textual and visual status information. Buttons allow to trigger manually all the steps required by ADAPT DO to carry on the infrastructure provisioning (e.g.

the startup of VMs) and the microservices deployment, according to the workflow defined by the ADAPT DO Execution Engine (cf. D4.4 [5] Section 2.2). The default behaviour of the DECIDE platform consists of automated steps activation, orchestrated by the DevOps framework. Anyway, the option of having buttons for manual triggering is a valuable alternative in case a user wants to test only a subset of the DECIDE framework functionalities.

For each operation:

- an icon shows whether the operation was successful (green tick mark), running (orange spinner), failed (red cross icon, not shown in **Figure *8***) or still to be triggered (off status icon)
- textual information shows the unique id of the operation (if triggered), the type of operation, the "environment" (meaning if it is an infrastructural component such as a VM or a microservice), the status (off, running, success, failed)

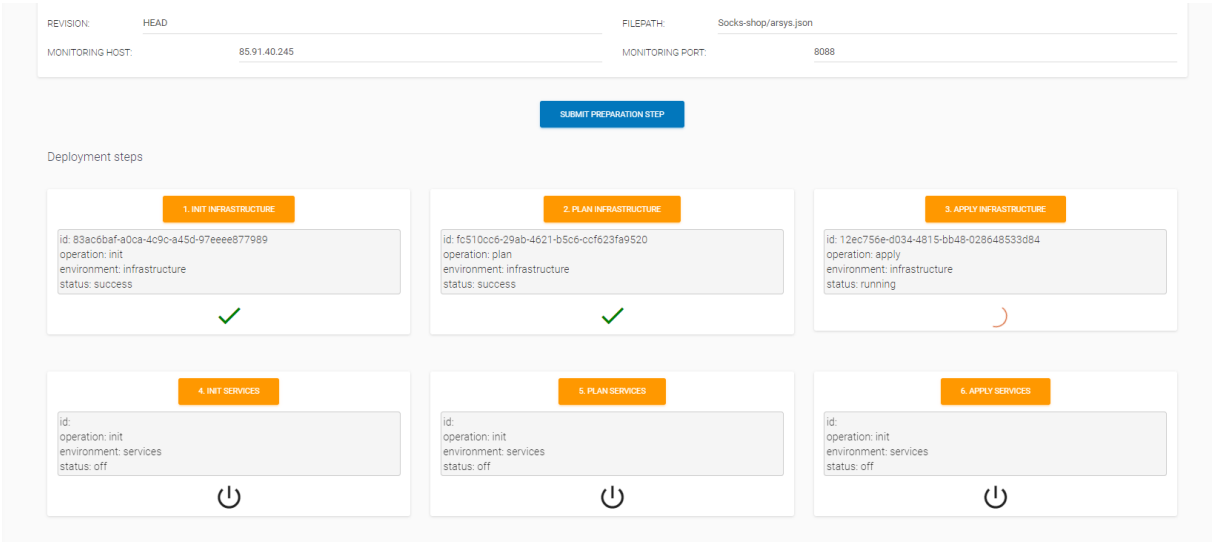

Figure 8. ADAPT DO Dashboard: buttons to trigger API calls

Typically, the most time-consuming operations are the ones that trigger infrastructure provisioning ("apply infrastructure" button), as this step implies requesting cloud providers to startup resources (e.g. VMs), which takes minutes, and the subsequent connection to such VMs to perform automated updates, software installation and system and software configurations. For this reason, ADAPT DO provides visual and textual items describing the status of such operations, as shown in **Figure *9*** and **Figure *10***.

Figure *9* describes the intermediate status of VM provisioning for the deployment of an application based on 2 nodes. The progress status circle bar gives an idea of how far we are from the completion of the tasks for each VM. The associated text in each box provides details related to the status: the public IP address of each node assigned by the cloud provider, the current operation, which step out of a maximum number of steps is being performed.

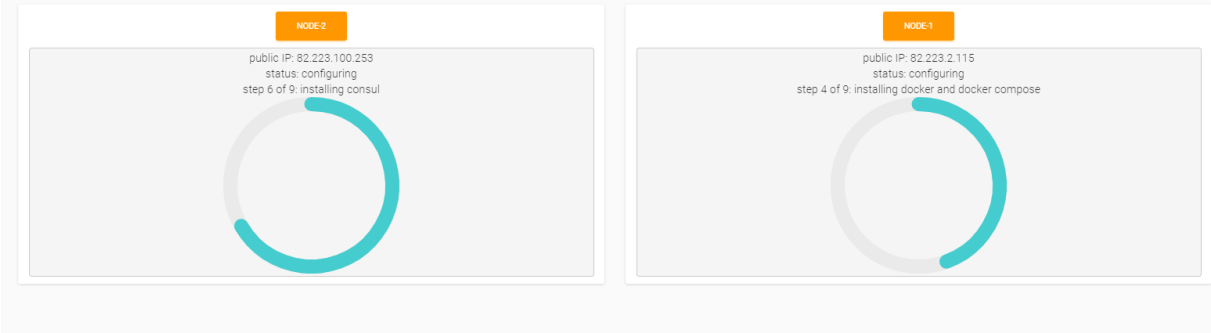

Figure 9. ADAPT DO Dashboard: progress status for infrastructure deployment

Figure *10*, finally, shows how the progress status visual elements are after the successful completion of the infrastructure deployment actions.

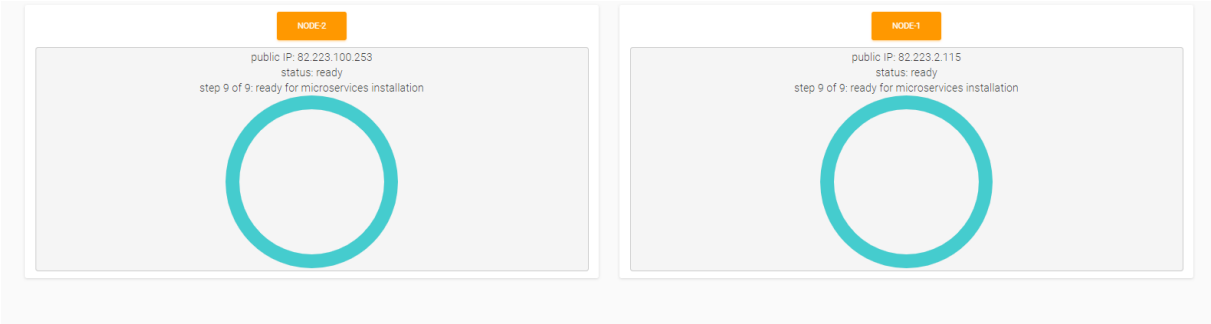

Figure 10. ADAPT DO Dashboard: completion of infrastructure deployment actions

The underlying implementation techniques enabling the status reporting in the Dashboard have already been described in D4.5 [1], Sections 2.1.2 and 2.2, and will not be repeated here.

3 Utilities for standalone mode

As documented in the previous version of this deliverable (D4.5 [1]), testing the whole DECIDE workflow, from the initial generation of the Application Description file to the final deployment of the application microservices on different clouds, is a complex process and require that the whole set of DECIDE components in the middle are fully integrated and functional. This cannot be accomplished at initial and intermediate stages of the project. To address this and to allow an easier intermediate evaluation of specific features of the platform, some tools are required to test a subset of KRs from the use case perspective (WP6).

To enable the evaluation of ADAPT DO features, and to prevent the partners involved in such activity from the need of performing manually (invoking the REST API) or from the Dashboard all the sub steps required by the ADAPT DO process, we provided since M24 a set of command-line tools for the Linux environment that allow, after a minimal configuration of a set of input parameters, to trigger the whole preparation and deployment process via simple commands.

As mentioned in the introduction, even if this was already documented in D4.5 [1], we repeat in this section the description of such tools and of their usage, in the same way as we do for the information required to run ADAPT DO. This is considered as an operating manual describing how to perform easily evaluation of the ADAPT DO functionalities. We think it is worth keeping a self-contained section for that, even if with some repetitions from D4.4 [5] and D4.5 [1], to support end users who need to follow instructions step-by-step.

3.1 Use Cases validation

The following CLI Tools have been extensively used in the timeframe M24-M30 to perform preliminary Use Case evaluations, from the ADAPT DO standpoint, of ARSYS and AIMES owned use cases. The usage of the CLI Tools allowed for a quick configuration and execution of the Use Cases, demonstrating the effectiveness of the deployment capability of ADAPT DO and the power of the automation steps performed behind the scenes. The results of such evaluations are documented in the dedicated deliverables in WP6.

3.2 CLI Tools

The tools consist of a set of scripts and in two configuration files, as shown in **Figure *11***, and requires that the "jq" tool (a tool that allows filtering of streams of json data) is installed on the system running the scripts. The "jq" tool can be easily installed via the operating system package management systems. In the case of Ubuntu, the install command is `sudo apt-get install jq`

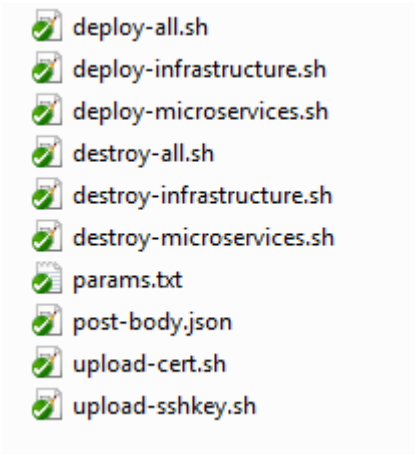

Figure 11. Script tools to facilitate intermediate use case validation

The scripts can be used at different levels: a user can run the "deploy-all.sh" script to trigger with a single command all the operations performed by ADAPT DO, namely the preparation, initialization, planning and execution of both the infrastructure and the microservices. As an alternative, a user can run separately and in sequence the "deploy-infrastructure.sh" and the "deploy-microservices.sh" scripts.

Before running any script, two files must be properly configured: the "params.txt" file, containing a set of variables that are used throughout the scripts, and a json file (in this example named "post-body.json"), containing the parameters that will be POSTed to the ADAPT DO REST API.

Figure *12* shows the parameters to be configured:

- JQ_HOME: the path to the home of the jq tool, usually /usr/bin
- ADAPT_IP: the ip address of the ADAPT DO
- ADAPT_PORT: the port where ADAPT DO is listening
- APPLICATION_NAME: the name of the application as written in the Application Descriptor
- PRIVATE_SSH_KEY_PATH: the path to the private key of the user as registered in ACSmI. This is required because during VM startup ADAPT DO must connect via ssh to the VM to install and configure software
- PREPARATION_JSON: the path to the json file containing the parameters to be configured and that will be POSTed to ADAPT DO
- DOCKER_REGISTRY_IP: the address of a private Docker registry, in case the application does not use the public Docker Hub
- DOCKER_REGISTRY_CA_CRT: the path to the certification authority certificate for the private registry
- APPLY: true to apply the operations, false to just simulate and log the calls

```
export JQ_HOME=/usr/bin
export ADAPT_IP=178.22.69.102
export ADAPT_PORT=8473
export APPLICATION_NAME=REDCap
export PRIVATE_SSH_KEY_PATH=./decide-user-kp
export PREPARATION_JSON=./post-body.json
export DOCKER_REGISTRY_IP=54.221.168.175
export DOCKER_REGISTRY_CA_CRT=./ca.crt
export APPLY=true
```

Figure 12. Parameters to be configured before scripts execution

The second file to be configured is the one specified in the "PREPARATION_JSON" parameter, in this example named "post-body.json". A sample configuration is depicted in **Figure *13***, where:

- Cloudbroker endpoint is the endpoint of the ACSmI cloud broker (ACSmI contracting)
- Cloudbroker_username is the username to be used to access the ACSmI platform (ACSmI contracting)
- Cloudbroker_password is the password to be used to access the ACSmI platform (ACSmI contracting)
- Repository_user is the username to be used to access the Git repository containing the Application Description file
- Repository_pwd is the password to be used to access the Git repository containing the Application Description file
- Repository_URL is the URL of the Git repository containing the Application Description file
- Revision: is the Git revision (can be left blank, HEAD will be used in that case)
- Filepath: the path to the Application Description file
- Monitoring_host: the address of the monitoring server

- Monitoring_port: the port of the monitoring server

```
{
  "cloudbroker_endpoint":"https://decide-prototype.cloudbroker.com",
  "cloudbroker_username":"decide@decide.eu",
  "cloudbroker_password":"[redacted]",
  "repository_user":"decide-user",
  "repository_pwd":"[redacted]",
  "repository_url": "https://git.code.tecnalia.com/decide/adapt-do-demo.git",
  "revision": "HEAD",
  "filepath": "Socks-shop/socks-shop.json",
  "monitoring_host": "89.45.56.98",
  "monitoring_port": "80"
}
```

Figure 13. Sample configuration for a post-body.json file

After configuring the two files above described, it is possible to run any of the scripts listed in **Figure *11***.

In particular, the launching of the "deploy-infrastructure.sh" script, will trigger:

- The upload to ADAPT DO of the ssh key
- The upload to ADAPT DO of the certificate for the private Docker registry (if needed)
- The invocation of the ADAPT DO REST API for preparing Terraform configuration files, based on the Application Description contents
- The invocation of the ADAPT REST API for
  - Initializing the Terraform state for the infrastructure
  - Creating a deployment plan
  - Applying the plan, hence starting up and configuring the VMs

Accordingly, the "deploy-microservices.sh" will trigger:

- The invocation of the ADAPT DO REST API for preparing Terraform configuration files, based on the Application Description contents
- The invocation of the ADAPT REST API for
  - Initializing the Terraform state for the microservices
  - Creating a deployment plan for the microservices
  - Applying the plan, hence downloading container images from the target VMs and starting them according to the parameters defined in the Application Descriptor

As mentioned, a "deploy-all.sh" script is available, which executes automatically the two scripts above described.

All the scripts have a corresponding "destroy-xxx.sh" counterpart, which allows to trigger undeployments and Terraform "destroy" operations on the services and infrastructure, in order to cleanup and shutdown VMs.

3.3 Ongoing extensions: metrics reporting

During preliminary Use Case validations, the need arose to take measurements related to the various automated steps of the application deployment. Such metrics are important for evaluation purpose, for comparisons to other tools and for end user awareness of how the underlying resources react to the DECIDE framework actions.

Therefore, we plan to add metric collection functionalities to the ADAPT DO framework, in particular to measure:

- The time taken for provisioning and configuring the virtual machines (which means CSP provisioning plus the time taken to update the VM operating system, to install the required runtimes, to configure security, to register the node to a consul k/v store)
- The time taken for deploying the application microservices on the provisioned infrastructure
- The time taken for re-deploying the application when a re-deployment action is triggered
- The time taken to undeploy the application
- The time taken to dismiss the CSP resources

ADAPT DO is a stateless component, therefore the data measured at each phase will be passed to the DevOps framework, which will store them in a centrally-managed component.

These extensions will be documented in the last version of the integrated platform documentation in WP2.

4 Updated Delivery and Usage

4.1 Package information

The ADAPT Deployment Orchestrator is packaged as a Docker image, to be launched as a container on any system running a Docker instance.

The image is currently available on an "unofficial" private Docker registry that we have set up for DECIDE, for development and testing purposes. Credentials are needed to login to the registry and to pull images. There are plans to set up publicly available, official repositories in the next iteration of the project for accessing code, images and all the needed material for running DECIDE components.

4.2 Installation instructions

Being packaged as a Docker image, the ADAPT Deployment Orchestrator does not require specific installation steps.

The only pre-requisite is to have access to a Linux-based system running a Docker daemon.

Currently, there are three options available to get the ADAPT Deployment Orchestrator image:

1. Get the image from the private Docker registry, in case you have been assigned credentials.
2. Get the image from a tar archive, available for download in case you have been authorized.
3. Build the image directly from the ADAPT project code.

Requests for credentials, download URLs and access to systems can be submitted by filling the form at the following URL: https://www.decide-h2020.eu/contact

4.2.1 Using the image from the private Docker registry

For users who have access to the private Docker registry where the ADAPT Deployment Orchestrator image is stored, the operations to perform to start it are:

- Log into the registry (only for first-time access) with the dedicated Docker command:

```
shell> docker login [registryIp]:[registryPort] -u registryUser -p registryPassword
```

- Execute the Docker 'run' command on the image, mapping a proper port which is open on the host and defining the –d (daemon) flag:

```
shell> docker run -p 8473:80 -d --name adapt [registry_ip]:[registry_port]/adapt:m30
```

- Verify that the container is up and running with the 'ps' command:

```
shell> docker ps
REPOSITORY                    TAG         IMAGE ID          CREATED          SIZE
aaa.bbb.ccc.ddd:nnnn/adapt    m30         4537a4171c5a      12 days ago       1.69GB
```

4.2.2 Extracting the image from a tar.gz archive

Images that are saved as tar archives, can be loaded on a local machine with the Docker 'image load' command.

- Download the 'adapt.tar.gz' file from the URL received from your request submission
- Unzip the archive with the 'gunzip' tool:

```
shell> gunzip adapt.tar.gz
```

- Load the image from the Docker shell with the following command:

```
shell> docker image load -i /home/ubuntu/adapt.tar

2c40c66f7667: Loading layer [==================================================>]
129.3MB/129.3MB

654f45ecb7e3: Loading layer [==================================================>]
45.45MB/45.45MB

f3ed6cb59ab0: Loading layer [==================================================>]
126.8MB/126.8MB

5616a6292c16: Loading layer [==================================================>]
326.7MB/326.7MB

97108d083e01: Loading layer [==================================================>]
8.043MB/8.043MB

815acdffadff: Loading layer [==================================================>]
62.18MB/62.18MB

325b9d6f2920: Loading layer [==================================================>]
4.608kB/4.608kB

2548e7db2a94: Loading layer [==================================================>]
5.591MB/5.591MB

9a9ce7dcd474: Loading layer [==================================================>]
8.599MB/8.599MB

df08e2c3d6fe: Loading layer [==================================================>]
5.209MB/5.209MB

3c0d8f1e556d: Loading layer [==================================================>]
3.584kB/3.584kB

87e2b99e95df: Loading layer [==================================================>]
3.584kB/3.584kB

5babbba9a986: Loading layer [==================================================>]
3.072kB/3.072kB

433a67f63093: Loading layer [==================================================>]
3.584kB/3.584kB

394c0a98982c: Loading layer [==================================================>]
3.072kB/3.072kB

461de7fb06ff: Loading layer [==================================================>]
5.346MB/5.346MB

b61bb40af46f: Loading layer [==================================================>]
3.584kB/3.584kB

40dc546a568a: Loading layer [==================================================>]
2.048kB/2.048kB

357d65e53b78: Loading layer [==================================================>]
2.048kB/2.048kB

66ddad2c15b4: Loading layer [==================================================>]
3.584kB/3.584kB

31b3b1f0ab7b: Loading layer [==================================================>]
5.5MB/5.5MB

bba5e66e1bc9: Loading layer [==================================================>]
3.072kB/3.072kB

0889a339caa2: Loading layer [==================================================>]
3.072kB/3.072kB

475e51c5e84e: Loading layer [==================================================>]
3.584kB/3.584kB

297a10341f47: Loading layer [==================================================>]
67.77MB/67.77MB
```

```
3cb698cec5c8: Loading layer [==================================================>]
45.06kB/45.06kB

be6e62470c3b: Loading layer [==================================================>]
17.58MB/17.58MB

b8983b6f72e0: Loading layer [==================================================>]
10.25MB/10.25MB

Loaded image: aaa.bbb.ccc.ddd:nnnn/adapt:m30
```

- Verify the image is available from the set of local Docker images:

```
shell> docker images

REPOSITORY                      TAG          IMAGE ID            CREATED             SIZE

aaa.bbb.ccc.ddd:nnnn/adapt    m30          4537a4171c5a        12 days ago         1.69GB
```

- Run the image:

```
shell> docker run -p 8473:80 -d --name adapt aaa.bbb.ccc.ddd:nnnn/adapt:m30
```

- Verify that the container is up and running with the 'ps' command:

```
shell> docker ps

REPOSITORY                      TAG    IMAGE ID        CREATED       SIZE

aaa.bbb.ccc.ddd:nnnn/adapt    m30    4537a4171c5a     12 days ago   1.69GB
```

4.2.3 Building the image from code

If you have access to the code repository, the ADAPT Deployment Orchestrator Docker image can be easily built from the Dockerfile available in the 'adapt-do' project repository available at the following address:

https://git.code.tecnalia.com/decide/adapt-do

You can get the ADAPT DO code either directly via the Git CLI commands or via the project Gitlab web UI.

4.2.3.1 Getting the code

Option 1 (via Git CLI):

- After configuring properly your credentials in the Git CLI, run the command:

```
shell> git clone --branch m30 git@git.code.tecnalia.com:decide/adapt-do.git
```

This will create a sub-folder "adapt-do" folder and download the code there

Option 2 (via the Gitlab Web UI):

- Click on the repository URL and, after logging in, select the tag "m30" from the drop-down menu containing the available project branches and tags, as shown in **Figure *14***.

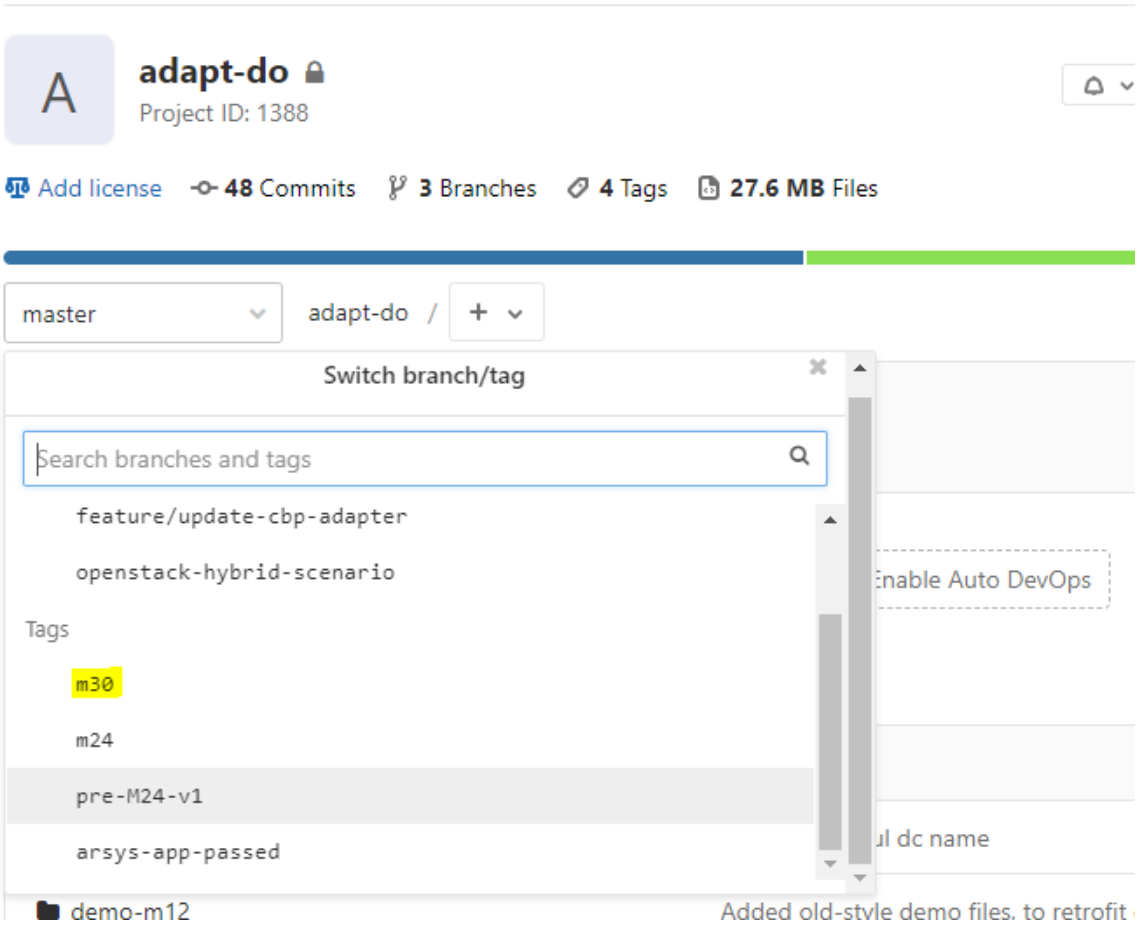

Figure 14. Selecting the M30 tag for getting the ADAPT DO source code released at M30

- Download the project code from Gitlab as zip or archive file clicking on the "download" icon, as shown in **Figure *15***

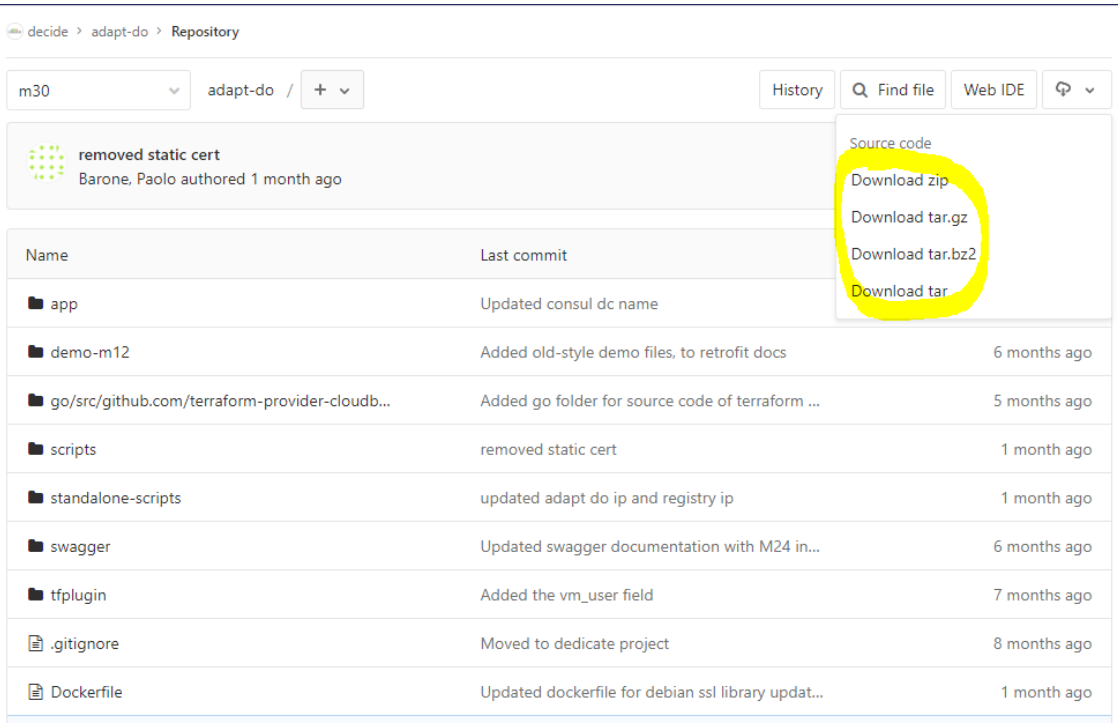

Figure 15. Downloading the source code

- Extract the archive into a directory of your choice (e.g. '~/tmp) and go into the directory 'adapt-do/'

4.2.3.2 Building the code

- Enter the "adapt-do" folder and run the command:

```
shell> sudo docker build -t adapt:m30
```

The above command builds the image 'adapt' with tag 'm30.

- Verify the image is available from the set of local Docker images:

```
shell> docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
adapt                       m30                 4537a4171c5a        12 days ago         1.69GB
```

- Run the image:

```
shell> docker run -p 8473:80 -d --name adapt adapt:m30
```

- Verify that the container is up and running with the 'ps' command:

```
shell> docker ps
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
adapt                       m30                 4537a4171c5a        12 days ago         1.69GB
```

4.3 User Manual

Once started, the ADAPT Deployment Orchestrator provides the REST API endpoints (documented in D4.4 [5] - Section 2.2.2, updated in D4.5 [1] - Section 3 and in Section 2 of the current document) for interactions with the other DECIDE components.

It is possible to test the component independent of the rest of the system, for evaluation purposes, by feeding the endpoints with proper input parameters.

Please consider that ADAPT Deployment Orchestrator is a component which is meant to be used by automatic tools; reproducing the behaviour via human interactions may result difficult due to the amount of manual interactions and configuration steps. As reported in Section 3, we provided also some CLI tools to facilitate this type of activity.

The pre-requisites for successful testing are:

- Having credentials to a git repository where an Application Description document will be fetched by ADAPT DO.
- Having credentials and permissions to access and start resources on the cloud broker environment configured. Such credentials comprise:
    - Username and password, to interact to the broker API and start resources.
    - SSH keypairs, to connect to the resources created. These are usually created and set up in the user profile creation steps, when registering to a cloud provider service. *In addition, it is also necessary to get the internal ID of the keypairs, which can be extracted automatically by automated tools but requires advanced steps for manual tests*. We have set up a test user with well-known ids and credentials in order to facilitate the verification from human end users.
- Having credentials to access the DECIDE private Docker image registry.

The steps for the test are:

1. Provisioning of an ADAPT Deployment Orchestrator instance specific to the test application.
2. Creation of Terraform configuration files for the infrastructure and for the services environments.
3. Initialization and planning of the infrastructure.
4. Provisioning of the infrastructure.
5. Initialization and planning of the services.
6. Provisioning of the services.
7. Verification of the application.

We suggest to use the CLI tools documented in Section 3, as they allow to get most of the required steps done automatically.

If you want to have a detailed insight of what is happening behind the scene step-by-step, then you can follow the steps documented in the remainder of this Chapter.

4.3.1 Provisioning of an ADAPT Deployment Orchestrator instance

You can choose one of the options described in Section 4.2 to get a Docker image for ADAPT DO and follow the directions described to start it.

4.3.2 Creation of Terraform configuration files for the infrastructure and for the services environments

In the DECIDE workflow, the information contained in the Application Description is passed to ADAPT via a Git repository, specific to the application that must be deployed. The file is stored and updated on that repository, and the Application Controller invokes ADAPT by POSTing a JSON data structure containing the needed information to access the repository and the specific revision of the file. To reproduce this mechanism manually, you have to specify your repository data and push a configuration file there, according to the following directions. In the 'adapt-do/demo-m12' folder, open the file 'app-descriptor.json' (which contains the description of the cloud resources we want to create: two virtual machines and a set of containers for running the Socks Shop application) and replace:

- all the fields marked with 'TO_BE_FILLED' with proper credentials;
- The fields marked with 'TO_BE_FILLED_WITH_ADAPT_IP' with the IP address of the ADAPT instance generated in the previous step.

Now, you have to push this file into your Git repository, at a well-specified path, and get the revision number of the file. This information is required in the next step. Here follow some directions on how to do it. Let's assume that your Git repository is named "my-app-repo", and that you have cloned it locally via the Git "clone" command as in the following:

```
shell> git clone git@git.code.myrepositories.com:decide/my-app-repo.git
```

You have now to copy the modified 'app-descriptor.json' in your project folder, then commit and push it to the remote repository:

```
shell> git commit -m "updated app descriptor" app-descriptor.json

shell> git push
```

Now, you have to get the revision of the file:

```
shell> git rev-parse HEAD

06f926e7bc8a6bd40703874dd9c05ed71e6ca49a
```

The returned hexadecimal code is the revision number, needed in the next step together with the information related to your Git repository.

- In the 'adapt-do/demo-m12' folder, open the file 'preparation-post-data.json' and replace:
  - all the fields marked with 'TO_BE_FILLED_XXX' with proper data.
- Using the 'curl' command line tool, POST the json data to the REST endpoint for the creation of configuration files for the infrastructure and for the services:

```
shell> curl -H "Content-Type: application/json" -X POST --data @preparation-post-data.json
http://[ADAPT_IP]:[ADAPT_PORT]/terraform/all
```

- Verify that a folder named 'My-Example-App' is created on the container:

```
shell> sudo docker exec -it adapt ls
```

- Verify the following subfolders structure is there:

```
shell> sudo docker exec -it adapt ls -R My-Example-App

My-Example-App:

infrastructure  services

My-Example-App/infrastructure:
```

```
My-Example-App-vm-node-1.tf
My-Example-App-vm-node-2.tf
My-Example-App-adapt/services:
My-Example-App-container-carts-db.tf
My-Example-App-container-carts.tf
My-Example-App-container-catalogue-db.tf
My-Example-App-container-catalogue.tf
My-Example-App-container-front-end.tf
My-Example-App-container-orders-db.tf
My-Example-App-container-orders.tf
My-Example-App-container-payment.tf
My-Example-App-container-queue-master.tf
My-Example-App-container-rabbitmq.tf
My-Example-App-container-shipping.tf
My-Example-App-container-traefik-private.tf
My-Example-App-container-user-db.tf
My-Example-App-container-user.tf
My-Example-App-container-zipkin.tf
My-Example-App-network-node-1-carts-network.tf
My-Example-App-network-node-1-user-network.tf
My-Example-App-network-node-2-catalogue-network.tf
My-Example-App-network-node-2-orders-network.tf
My-Example-App-network-node-2-shipping-network.tf
My-Example-App-services-common.tf
```

The files with '.tf' extension contain the configurations for Terraform.

4.3.3 Initialization and planning of the infrastructure

- Initialize the infrastructure environment:

```
shell> curl -H "Content-Type: application/json" -X POST
http://[ADAPT_IP]:[ADAPT_PORT]/terraform/init/My-Example-App/infrastructure
```

You can poll the URLs returned by the command to verify the status and logs of the request.

- Create a deployment plan for the infrastructure:

```
shell> curl -H "Content-Type: application/json" -X POST
http://[ADAPT_IP]:[ADAPT_PORT]/terraform/plan/My-Example-App/infrastructure
```

You can poll the URLs returned by the command to verify the status and logs of the request.

4.3.4 Provisioning of the infrastructure

- Deploy the infrastructure:

```
shell> curl -H "Content-Type: application/json" -X POST
http://[ADAPT_IP]:[ADAPT_PORT]/terraform/apply/My-Example-App/infrastructure
```

You can poll the URLs returned by the command to verify the status and logs of the request.

4.3.5 Initialization and planning of the services

- Initialize the services environment:

```
shell> curl -H "Content-Type: application/json" -X POST
http://[ADAPT_IP]:[ADAPT_PORT]/terraform/init/My-Example-App/services
```

You can poll the URLs returned by the command to verify the status and logs of the request.

- Create a deployment plan for the services:

```
shell> curl -H "Content-Type: application/json" -X POST
http://[ADAPT_IP]:[ADAPT_PORT]/terraform/plan/My-Example-App/services
```

You can poll the URLs returned by the command to verify the status and logs of the request.

4.3.6 Provisioning of the services

- Deploy the service:

```
shell> curl -H "Content-Type: application/json" -X POST
http://[ADAPT_IP]:[ADAPT_PORT]/terraform/apply/My-Example-App/services
```

You can poll the URLs returned by the command to verify the status and logs of the request.

4.3.7 Verification of the application

If the above steps succeeded, you would be able to access the Socks Shop application at the URL:

- http://[node-2-ip]:80

Where [node-2-ip] is the address of the virtual machine 2 started by the 'infrastructure apply' operation.

4.4 Licensing information

The plan is to release the ADAPT DO component, developed by HPE, as open source software. HPE is following an internal process with reviews and decisions at corporate level to decide and approve the license under which to release the developed software. This process is approaching to the final approval, but it is not completed at the time of writing; therefore the licensing information for the released software is not yet officially available. Anyway, the most likely licensing will be based on the Apache v2 model.

In any case, credentials can be provided under request, and download URLs and access to systems can be requested by filling the form at the following URL: https://www.decide-h2020.eu/contact.

5 Conclusions

This deliverable, written as continuation/update of deliverable D4.4 [5] and D4.5 [1], is a report accompanying the software package deliverable, which is released together with this document on the dedicated area of the official content and document sharing system. It reports the status of the implementation for the requirements expected for ADAPT DO at M30. The outcome is the ADAPT DO release M30. In particular, it considers the specification of the requirements identified for ADAPT DO in the D4.3 [2] document (related to the ADAPT architecture definition) and expected by M30, and describes how the related use cases are satisfied by the features implemented.

The M30 prototype is responsible for the final deployment and adaptation functionalities of DECIDE and collects all the functionalities specified by the requirements during the whole project lifetime.

The document is not self-contained, it is rather the natural extension of previous deliverables D4.4 [5] and D4.5 [1], therefore the reading of all of them is needed for an exhaustive comprehension. In the following we recap briefly the outcomes of all of them.

The initial deliverable D4.4 [5] has described the first ADAPT prototype that was implemented during the first year of the project. The initial prototype was focused on the basic deployment and adaptation functionalities, carried out by the Deployment Orchestrator component. The main functionalities of this component were listed, along with an explanation of how they fit within the whole DECIDE workflow. The component has been thoroughly described, including the exposed APIs and tools used. This first deliverable was the most important, as it documented the main architectural choices and the rationale behind the tools and languages selected to implement ADAPT DO. The main goal was to create it as a stateless microservice providing a REST API and leveraging the Infrastructure as Code paradigm to provision infrastructures dynamically. Furthermore, the deliverable provided information about the packaging of the component, the requirements and dependencies and the steps needed to test it. A description of the SockShop application, the test application that would have been used to validate DECIDE, was also given.

The intermediate deliverable D4.5 [1], written as a continuation of D4.4 [5], described the updates and extensions implemented during the period M12-M24 to the ADAPT DO prototype released in M12. Taking as the basis the deliverable D4.4, and the refined ADAPT architecture documented in D4.2 [4], it reported the extensions and the updates developed till M24, toward the integration with the other DECIDE components. The outcome was the ADAPT DO release M24. It was used as the basis for the current document.

6 References

[1] DECIDE, “D4.5 Intermediate multi-cloud application deployment and adaptation”.

[2] DECIDE, “D4.3 Final DECIDE ADAPT Architecture”.

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

[4] DECIDE, “D4.2 Intermediate DECIDE ADAPT Architecture”.

[5] DECIDE, “D4.4 Initial multi-cloud application deployment and adaptation”.

[6] DECIDE, “D4.10 Initial multi-cloud application helpers”.