Deliverable D4.5

Intermediate multi-cloud application deployment and adaptation

Editor(s):	Paolo Barone (HPE)
Responsible Partner:	HPE
Status-Version:	Final – V1.0
Date:	28/11/2018
Distribution level (CO, PU):	CO

Project Number:	GA 726755
Project Title:	DECIDE

Title of Deliverable:	Intermediate multi-cloud application deployment and adaptation
Due Date of Delivery to the EC:	30/11/2018

Workpackage responsible for the Deliverable:	WP4 - Continuous deployment and operation
Editor(s):	Paolo Barone (HPE)
Contributor(s):	Paolo Barone (HPE)
Reviewer(s):	Andrey Sereda (CB), Nicola Fantini (CB), Vitalii Zakharenko (CB)
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP5, WP6

Abstract:	This intermediate version of the deliverable is the continuation of deliverable D4.4. It updates the methods and the tools for implementing the automatic deployment and adaptation of multi-cloud applications and provides updates and extensions to the technical description of the workflow manager implemented for supporting those activities. As defined by the related Amendment, this document also includes the updates to the "Helpers" developed as part of ADAPT Deployment Orchestrator, formerly documented in the dedicated deliverable D4.10.
Keyword List:	Microservice, REST, container, virtual machine, orchestration system, adaptation, architecture, installation, packaging, usage, Openstack, 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	19/10/2018	First draft version	Paolo Barone (HPE)
v0.2	25/11/2018	Version submitted to reviewer	Paolo Barone (HPE)
V0.3	27/11/2018	Version containing reviewer comments	Andrey Sereda (CB), Nicola Fantini (CB), Vitalii Zakharenko (CB)
V0.4	28/11/2018	Version addressing reviewer comments	Paolo Barone (HPE)
v0-5	28/11/2018	Final version	Paolo Barone (HPE)
V1.0	28/11/2018	Ready for submission	Leire Orue-Echevarria (TECNALIA)

Table of Contents

List of Figures

Terms and abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
CSP	Cloud Service Provider
DevOps	Development and Operations
DoW	Description of Work
EC	European Commission
GB	Giga Bytes
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KR	Key Result
MCSLA	Multi-Cloud Service Level Agreement
NCA	Native Cloud Application
NFR	Non Functional Requirement
OS	Operating System
Protobuf	Protocol Buffers (Google's data interchange format)
QA	Quality Assurance
RAM	Random-Access Memory
REST	Representational State Transfer
SLA	Service Level Agreement
SLO	Service Level Objective
SQO	Service Quality Objective
SSH	Secure SHell
ToC	Table of Contents
UI	User Interface
URI	Unified Resource Identifier
URL	Unified Resource Locator
UC	Use Case
UUID	Universally Unique IDentifier
VM	Virtual Machine
WP	Work Package

Executive Summary

This deliverable, related to the intermediate multi-cloud application deployment and adaptation, is to be considered an extension of the D4.4, which documented the initial implementation of the ADAPT Deployment Orchestrator (ADAPT DO). Since the relationship with the architecture defined in D4.1 was already in a consolidated state at M12, as well as the internal design, the execution behavior and the dynamic generation of configuration scripts for the automated deployment, to avoid huge repetitions between different versions of the document we will not report here the concepts already documented in D4.4, as they would be identical. Instead, this document reports the extensions added on top of the software released for M12 and the updates made to that release.

Just to recap the main concepts, only in this executive summary we recall from D4.4 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, in the M24 release a validation mechanism has been added to the ADAPT DO.

In the M24 release, a dedicated tab and related web page has been added to the DevOps framework (which is the GUI part of the DECIDE framework). ADAPT DO provides now a Dashboard to give to the end user a set of textual and visual information about the progress and status of the deployment activities.

ADAPT DO interacts with the ACSmI component, to start and release cloud resources when needed. In the current release a lot of improvements have been added with respect to the number of Cloud Service Providers (CSPs) supported and the type of cloud deployment scenarios allowed. While for the M12 release it was possible to leverage only two public providers (AWS and Cloudsigma), for the M24 release we can now 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, this document also includes the updates to the “Helpers” developed as part of ADAPT DO, formerly documented in the dedicated deliverable D4.10.

Regarding the implementation and the packaging aspects, ADAPT DO has not changed since M12: 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, Werkzeug, Jinja 2, Flask-RESTplus and Terraform.

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 of 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 for the M24 version, to help end users who want to experience the ADAPT DO capabilities as a standalone tool. 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 Introduction

1.1 About this deliverable

This deliverable focuses on the implementation at M24 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 intermediate 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) and D4.2 (intermediate) 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 M24. In particular, it describes what has been implemented in the second year of the project, what are the new features introduced in the period from M12 to M24 and which updates or extensions have been added with respect to the M12 version, already documented in the deliverable D4.4. In addition, this document also integrates updates to the "Helpers", formerly documented in a dedicated document (D4.10), now consolidated here as a result of an Amendment.

1.2 Notes for the reader

To avoid repetitions between the deliverables related to T4.2 (D4.4 and the current D4.5), this document will *not* describe (unless changed since the M12 version):

- 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 were documented accordingly in the corresponding deliverable D4.4.

It will rather focus on updates to the interfaces and extensions to capabilities and must then be considered as a continuation of D4.4.

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 is a pre-requisite for D4.5 and both document must be considered.

The only exception to the above approach is Section 4, which is an operational manual describing how to get, set up and launch the ADAPT DO component. We think it is worth keeping a self-contained section for that, even if with some repetitions from D4.4, to support end users who need to follow instructions step-by-step.

In case some contents in D4.5 invalidates and fully replaces specific contents of D4.4, 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 the new features introduced in the timeframe M12-M24, such as the ADAPT DO Dashboard, the deployment progress status reporting mechanism, CLI tools for using ADAPT DO as a standalone component, input data validation and the support for a Hybrid/Private cloud scenario.

Section 3 explains what extensions were added to already existing functionalities implemented by M12, related to the REST API, the support for status data storage, the input/output data, the

refinement of “helpers” which allow connecting to cloud providers, the setup of a real private cloud infrastructure for demonstrating the Hybrid/Private scenario.

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

2 New features

2.1 ADAPT DO dashboard in the DevOps framework

ADAPT DO is a "backend" component: the end user does not interact directly with ADAPT DO, only the other DECIDE software components do it programmatically, by invoking the provided REST API and by exchanging the Application Description file, filled in with proper data. ADAPT DO, positioned at the end in the DECIDE workflow, applies infrastructure provisioning and deployment actions based on the deployment topology specified by the other DECIDE components. For this reason, ADAPT DO does not need any GUI for the management of specific actions. Anyway, since the infrastructure provisioning and deployment actions are long-lasting operations, mainly due to the time required to start up VMs on the Cloud Service Provider (CSP) side, to install software and updates, and to download container images, ADAPT DO provides a visual Dashboard in the DevOps framework to inform the end user about the status of the deployment actions.

In addition to the status visualization, in order to facilitate testing and demonstration activities, the current (intermediate) version of the Dashboard provides also some buttons and forms allowing to trigger manually the operations required (which in a real use case would be performed automatically behind the scenes via API calls). In the final version of the Dashboard, we plan to remove such buttons.

2.1.1 ADAPT DO dashboard items

The following pictures show the visual items provided by the ADAPT DO Dashboard in the M24 version.

Figure *1* depicts the positioning of the ADAPT DO tab in the DevOps framework (left side), and forms which allow to insert manually all the parameters required to trigger API calls to ADAPT DO. Such data are then collected as a json body and get POSTed to the proper ADAPT DO endpoints to trigger actions. As mentioned, these forms will not be needed in the final version and will be removed or made optional, but they are very helpful in the intermediate versions of the DECIDE platform for testing, integration and demonstration purposes.

The forms are placeholders for generating exactly the body required by the REST endpoints documented in D4.4, containing the following required information:

- IP address and port of the ADAPT DO endpoint
- Name of the application to be deployed
- ACSmI endpoint and credentials, to trigger cloud service provisioning actions
- GIT url, credentials, path and revision of the Application Description json file containing the deployment topology and all the information required to deploy the application
- IP and port of the monitoring host, implementing the ADAPT Monitoring Manager functionality (cf. D4.7)

The button "submit preparation step" allows to submit the above described data to the ADAPT DO Preparation Engine (cf. D4.4, Section 2.2), which retrieves the Application Description from the Git repository and processes it, translating the information contained into Terraform configuration files that will enable the automated deployment.

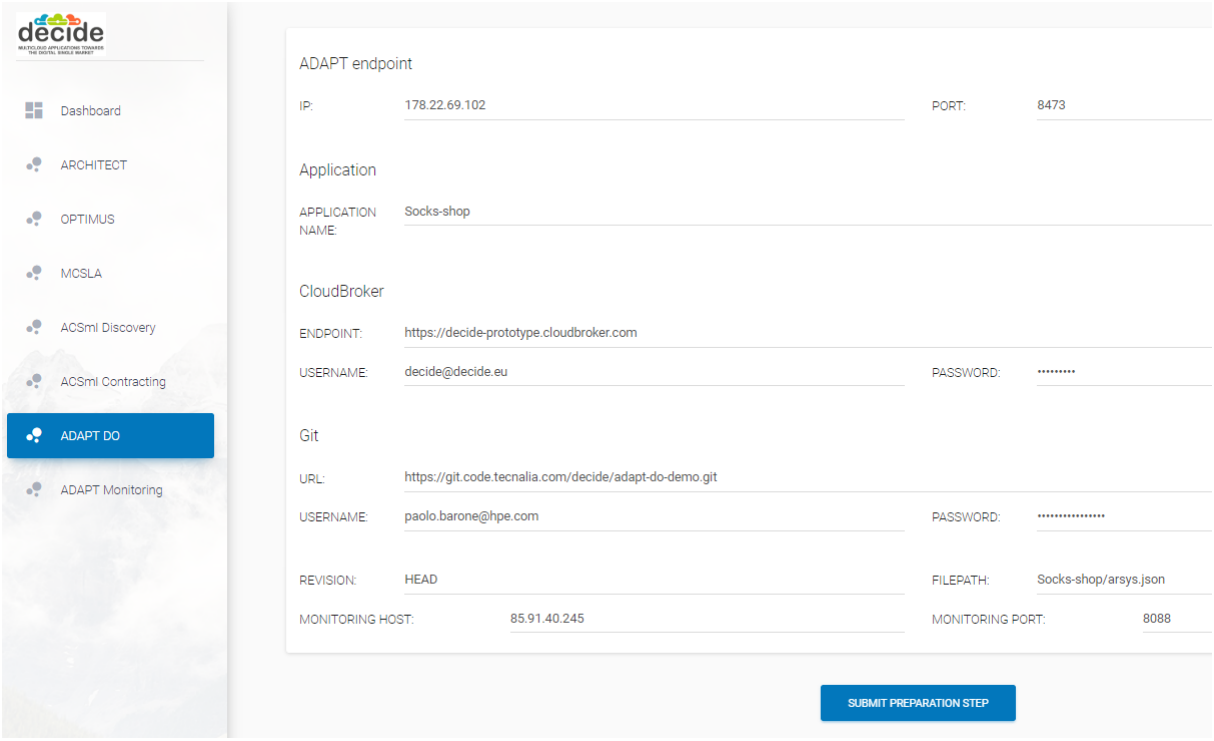

Figure 1. ADAPT DO page in the DevOps framework: forms to set input parameters

Figure *2* shows a set of buttons, together with textual and visual status information, which allows 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 Section 2.2). For each operation:

- an icon shows if the operation was successful (green tick mark), if it is running (orange spinner), if it failed (red cross icon, not shown in **Figure *2***) or if it must still 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)

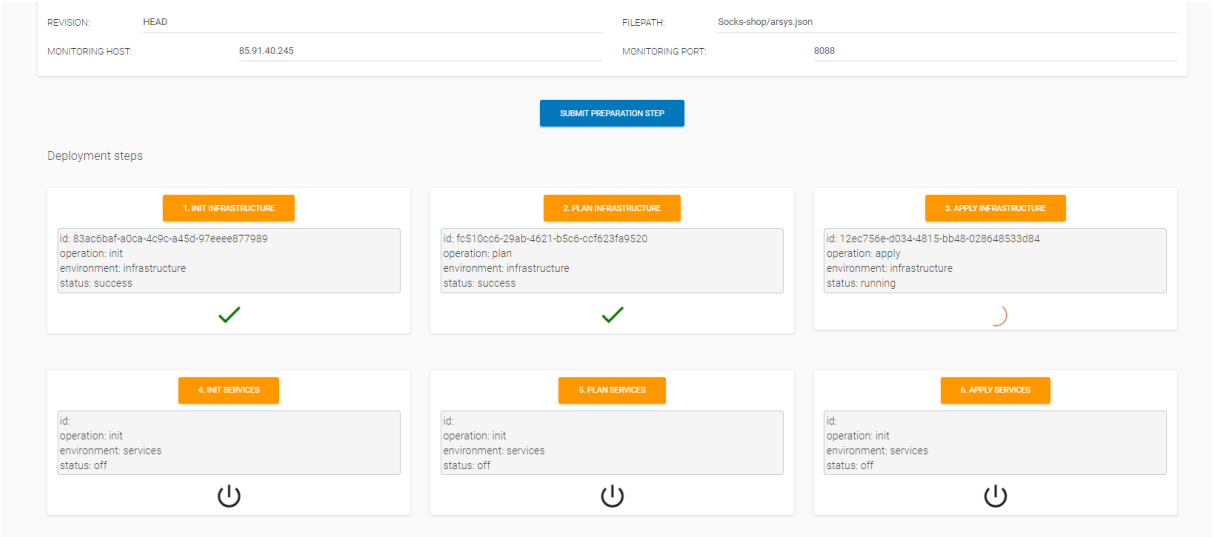

Figure 2. ADAPT DO Dashboard: buttons to trigger API calls

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 3 and Figure 4.

Figure 3 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.

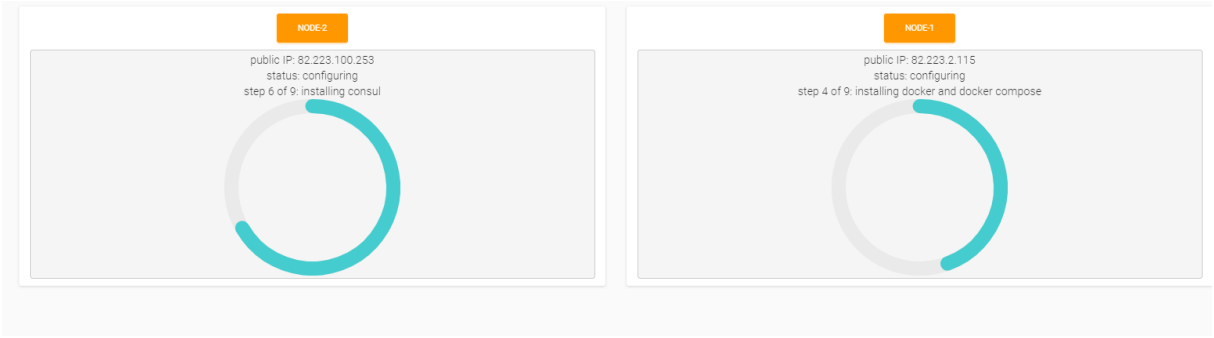

Figure 3. ADAPT DO Dashboard: progress status for infrastructure deployment

Figure 4, finally, shows how the progress status visual elements are after the successful completion of the infrastructure deployment actions.

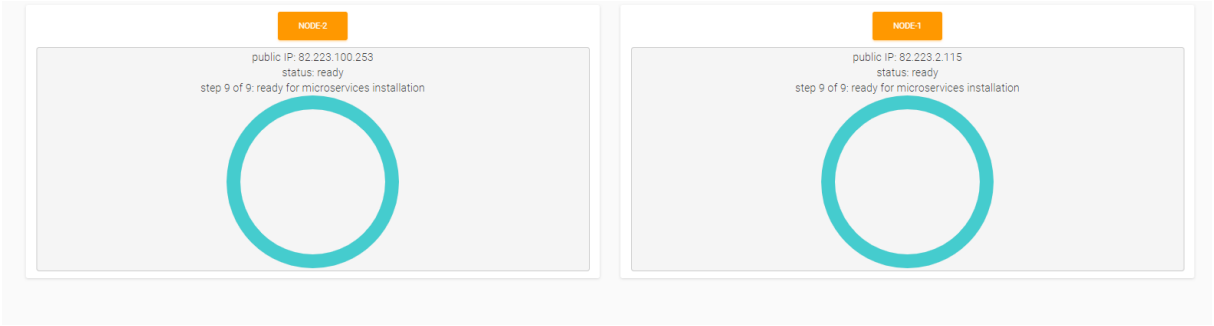

Figure 4. ADAPT DO Dashboard: completion of infrastructure deployment actions

2.1.2 ADAPT DO dashboard implementation

The ADAPT DO Dashboard code itself is indeed part of the DevOps framework; accordingly, it was developed in the AngularJS language. We document it here as it was implemented as an ADAPT DO feature within the WP4 scope and because it required extensions in the ADAPT DO code, in its REST API and in the deployment templates to be functional. Also, at this intermediate stage it allows to test the ADAPT DO functionalities in a standalone mode, without specific requirements from the components preceding ADAPT DO in the workflow.

The key enablers for the visualization of deployment status information are:

- The extensions of the software installation scripts which are copied and executed on the VMs upon their startup by ADAPT DO. Such extensions allow to detect and report the infrastructure provisioning status form the VMs to ADAPT DO

- The saving of deployment progress status data in a DB managed by ADAPT DO and running on the same machine
- The extension of the ADAPT DO REST API, with endpoints which allow:
    - the starting VMs to POST progress status data to the API
    - The ADAPT DO dashboard (or any component that may need it) to GET progress status data and, in general, the deployment status

Such enablers are described later in this document, in dedicated sections (cf. 2.2, 3.1, 3.2).

The ADAPT DO Dashboard leverages the AngularJS features related to asynchronous operations on GUI items, where a single GUI item can be associated to the information retrieved by a remote API call (in this case the calls to the ADAPT DO REST API endpoints allowing the retrieval of progress status), independent from the other GUI items in the Web GUI. This allows to display simultaneously multiple elements, each one showing and updating the status of a specific remote element (e.g. of a VM), without affecting the rest of the page.

2.2 Deployment progress status reporting

As mentioned in Section 2.1.2, the reporting of the deployment status is enabled by extending the installation scripts (cf. D4.4 Section 2.2.2.3) that are copied to the VMs and executed by ADAPT DO during the deployment time. Such scripts, which take care of the setup (updates, software installation, configuration) of the VM runtime environment, are orchestrated by a master script named “init-vm.sh”, that takes as input a set of parameters, among which the ADAPT DO public ip and port:

```
#$1: decide-vm name
#$2: decide-vm external_ip_address
#$3: decide-vm internal_ip_address
#$4: wan consul-join-ip
#$5: adapt host public ip and port
#Optional arguments:
#$6: Private Docker registry ip
#$7: Private Docker registry port
```

Figure 5. Input parameters driving VMs initialization

Using that information, the script can POST data on the extended ADPAT DO REST API to register the status of the deployment progress. The snippet below gives an example of this mechanism: the “curl” instruction performs a POST on the ADAPT DO endpoint “vm/[vm_ip_address]/status”, declaring that step 3 out of 9 is being performed, corresponding to the task of “security configuration”.

```
echo "Configuring security..." >> /tmp/scripts/install.log

echo "About to curl for step 3: security"
curl --max-time 4 -X POST -H 'Content-Type: application/json;charset=UTF-8' -d '{"vmName": "'$1'", "vmPublicIp": "'$2'","status":
"configuring", "step": 3, "steps": 9, "desc": "configuring security"}' $5/vm/$2/status || RESULT=$?

if [ $RESULT -ne 0 ]; then
    echo "Could not log step status"
fi
```

Figure 6. Example of a POST from the VM initialization script, for updating the deployment progress *status*

The POSTed data are then stored by ADAPT DO in a private database and made available to the Dashboard (or any other client interested) via dedicated REST endpoints. This allows the visualization of the data shown in Figure 3 and Figure 4.

This mechanism is applied for all the sub-steps of the deployment, until the VM is ready for hosting the application microservices.

Currently this feature lacks an authorization mechanism to verify that update status requests are legal and coming from the expected IP addresses. This will be addressed during the next project timeframe (M24-M36).

2.3 CLI Tools for standalone mode to support Use Cases validation

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 (M12) and intermediate (M24) stage of the project but is expected to be for the next integrated version of the DevOps framework in M27. Therefore some tools are required to allow the testing of 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 substeps required by the ADAPT DO process, we provided 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.

The tools consist in a set of scripts and in two configuration files, as shown in **Figure *7***, 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`

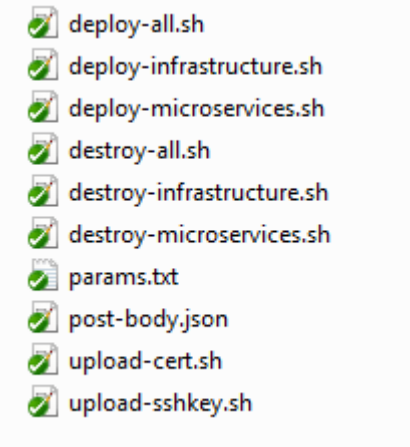

Figure 7. Script tools to facilitate intermediate use case validation

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 8 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 Description
- 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 8. 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 9**Figure *9***, where:

- Cloudbroker endpoint is the endpoint of the ACSmI cloud broker
- Cloudbroker_username is the username to be used to access the ACSmI platform
- Cloudbroker_password is the password to be used to access the ACSmI platform
- 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 9. 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 7..

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 "deploly-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 Description

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.

2.4 Input data validation

The primary data exchange mechanism between the DECIDE components is the Application Description file. The file is generated at the beginning of the DECIDE workflow, and enriched step by step by the components that participate in it. In order to have consistency throughout the whole workflow, a data model with a related schema is provided (within WP3 activities), defining what are the allowed fields and their format, what is their cardinality, if they are required or optional and in general the overall structure that will be transformed into the Application Description json file.

It is then fundamental for the DECIDE components which use the Application Description file to perform proper validation whenever it is parsed or updated, so to avoid breaking consistency and generating errors at some point in the workflow execution. ADAPT DO was updated accordingly, with the introduction of a mechanism to validate the Application Description json file against a schema.

Form the technical point of view, since the core of ADAPT DO was developed in Python via the Flask framework (cf. D4.4), the validation is enabled leveraging the Python "jsonschema" library, used with the "Draft7Validator". Whenever the Application Description is retrieved by the Git repository, and just before pushing an updated Application Description to the Git repository, it is checked for validation against the official release of the data model schema.

The official release of the schema is released within WP3 activities, in the Git repository of the AppController project at:

https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/master/AppController

2.5 Hybrid cloud scenario enablement

The initial release of ADAPT DO for M12 had the main purpose of demonstrating the capability of deploying a microservice application on different clouds, taking care also of the VMs provisioning and the preparation of the required runtime environment for the microservices execution. The deployment addressed a public multi-cloud scenario, where the available CSPs in the DECIDE platform were the public cloud providers AWS and Cloudsigma.

Making one step forward, in addition to the extensions in the deployment capabilities, during the timeframe M12-M24, we started addressing a more complex scenario related to hybrid cloud, where the microservices of an application would be deployed on a multi cloud scenario where some services are running in a public cloud while other (e.g. more sensitive ones) in a private cloud.

Therefore, to demonstrate the above scenario we were missing the following:

- Having a private cloud infrastructure available
- Having the support from ACSmI to access such private cloud
- Having the proper extensions in the ADAPT DO code and configuration scripts to support the specific needs (in particular in terms of access and security) of a private cloud

Hence, a set of tasks were carried on in parallel and in strict collaboration between the teams working at ADAPT DO in WP4 and at ACSmI in WP5 in order to fulfil the above requirements.

More in detail:

- A private cloud infrastructure (at Labs level) was set up in HPE, in an environment almost “isolated” from the public network, with only a few ports opened and with very strict firewall rules, as it happens in real enterprise-level private cloud scenarios
- ACSmI was extended to support Openstack clouds and in particular to support this specific scenario, with very limited access to ports
- ADAPT DO was updated to support the interactions with such kind of cloud

At the time of writing (M24), ACSmI is fully integrated with Openstack and can manage the private cloud resources on the HPE infrastructure; the integration between ADAPT DO and the ACSmI interfaces for triggering VMs provisioning is under testing and expected to be completed by M25. Final results will be documented in the final version (D4.6) of the deliverables related to task T4.2.

3 Extensions

3.1 Extensions in the REST API

To enable the reporting and retrieval of deployment status data for visualization in the Dashboard or, in general, for any component that may need it, the new endpoints shown in Figure 10 have been implemented as extensions to the ADAPT DO REST API.

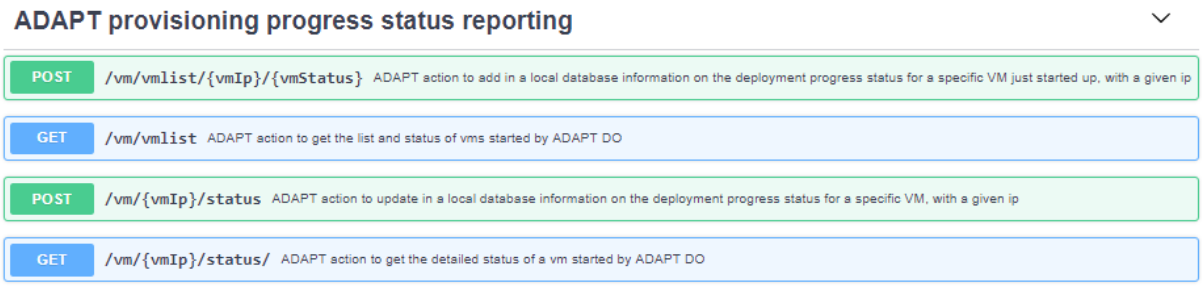

Figure 10. New ADAPT DO endpoints for deployment status data

In the following we describe such endpoints.

Figure 11 shows the endpoint for POSTing to ADAPT DO information related to a new VM started on a CSP. As mentioned earlier in the document, the process of VM startup takes a meaningful amount of time (minutes). This endpoint is invoked when the VM is fully started up and running using the scripts the are copied on the VM at deployment time by ADAPT DO. This allows to know that the VM has successfully started and is being "prepared" for hosting DECIDE applications microservices. The POSTed data are the parameters defined in the endpoint path: {vmIp} and {vmStatus}. The {vmIp} is an information which is known only after the VM has been stared up, because it is assigned (usually as a floating ip address) by the CSP. The {vmStatus} can be one of the following statuses, corresponding to the operations performed by ADAPT DO during the deployment:

- "configuring": the VM is up and running, has a public ip assigned, and is going to be configured for hosting the DECIDE applications microservices
- "ready": the VM has been configured and is ready for hosting microservices

The POST operation returns code 200 for success, and 404 for invalid input.

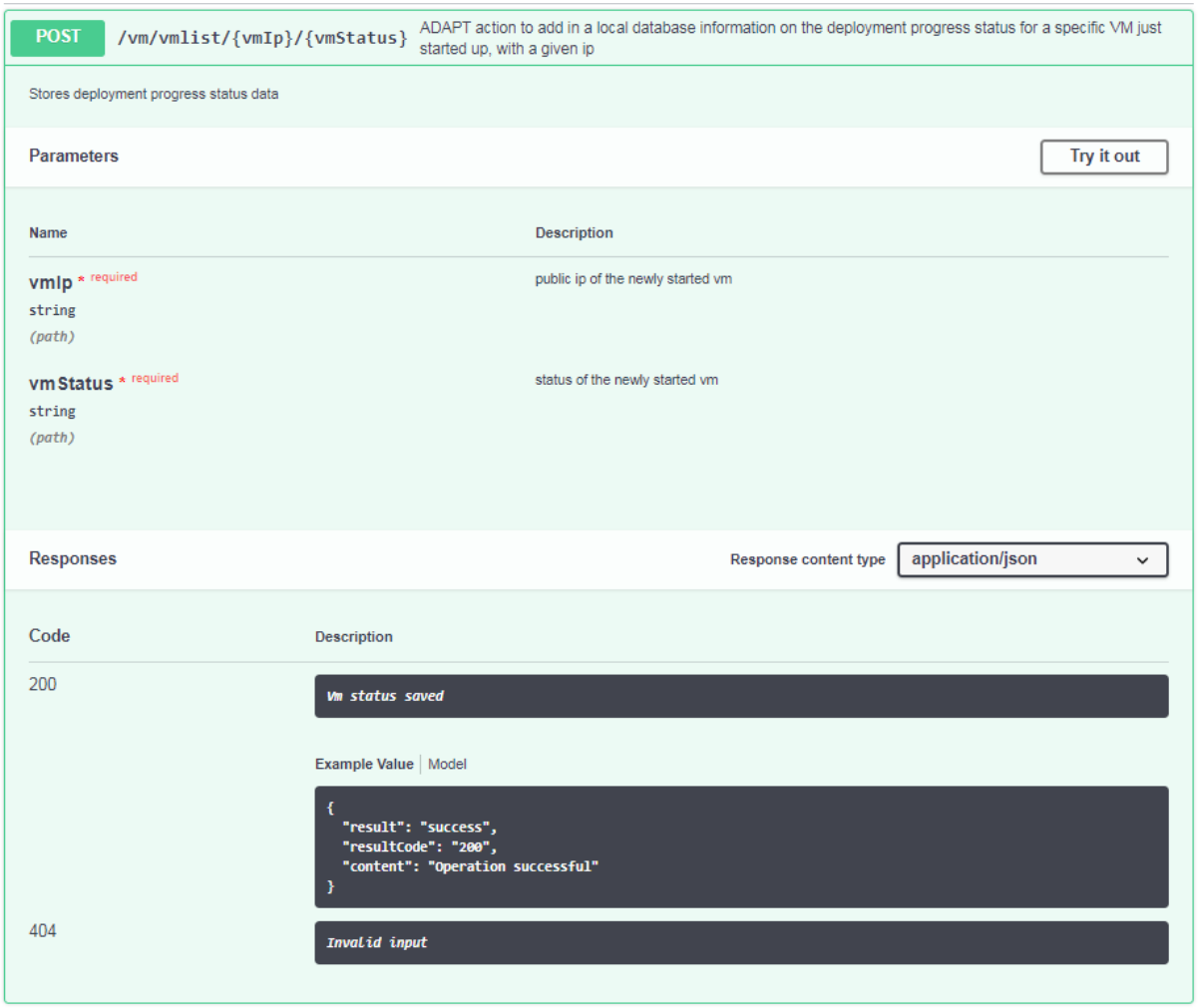

Figure 11. Endpoint for adding status information for a new VM started on a CSP by ADAPT DO

Figure 12 is the GET counterpart of the previous endpoint. It allows to get the list of VMs started by ADAPT DO for a specific deployment topology. Having the possibility of querying about the list of VMs started by ADAPT DO is important for the Dashboard, as it is the mechanism which allows to update the GUI dynamically whenever a new VM gets ready on the CSP side, independent of the other ones started on other CSPs.

The GET operation returns a list of ips, each one identifying a VM and with the associated status (“ready” or “configuring”). In case of error, it returns a 400 “not found” error.

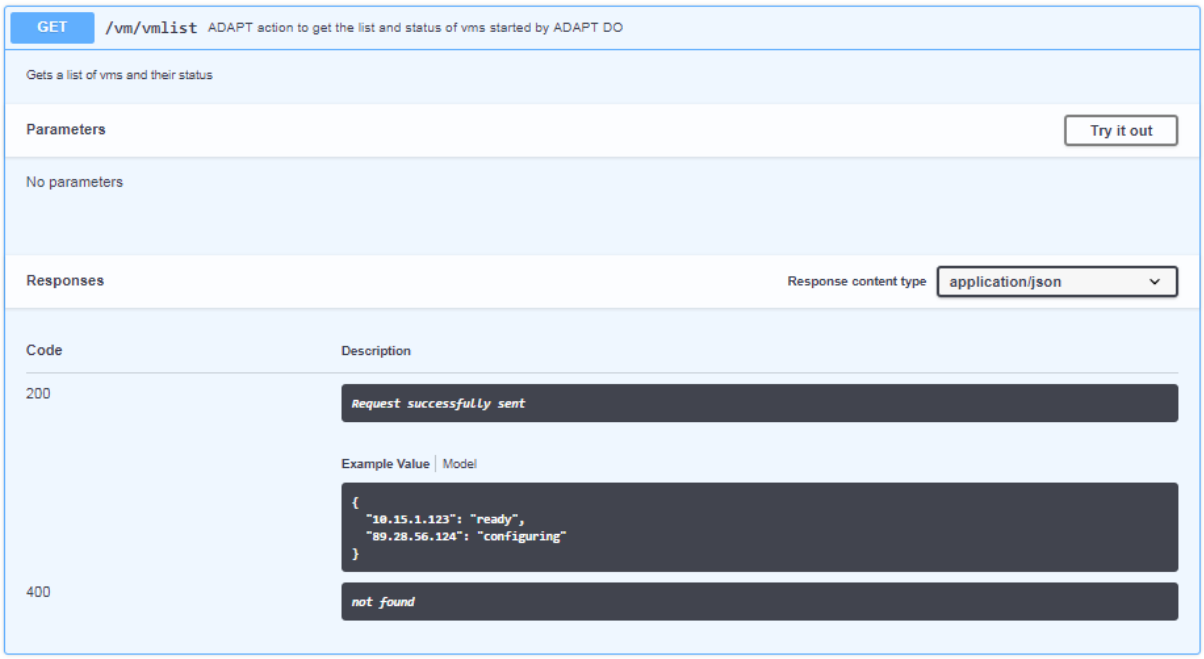

Figure 12. Endpoint for getting the list of VMs started on a CSP by ADAPT DO

Figure 13 shows the endpoint which is used by the VM configuration scripts to update the status detail of a specific VM started by ADAPT DO on a CSP and which is under the process of configuration.

The endpoint requires the {vmIp} parameter on the path, and a body containing the following information:

- "vmPublicIp": Ip of the virtual machine
- "desc": textual description of the status. Can be one of the following:
  - "clean before update"
  - "update packages"
  - "configuring security"
  - "installing docker and docker compose"
  - "configuring docker as a service"
  - "installing consul"
  - "joining consul wan cluster"
  - "configuring access to private registry"
- "status": can be one of the following:
  - "configuring"
  - "ready"
- "step": the number of the current step being performed
- "steps": the total number of steps expected
- "vmName": the name of the VM as defined in the deployment topology

The POST operation returns code 200 for success, and 404 for invalid input.

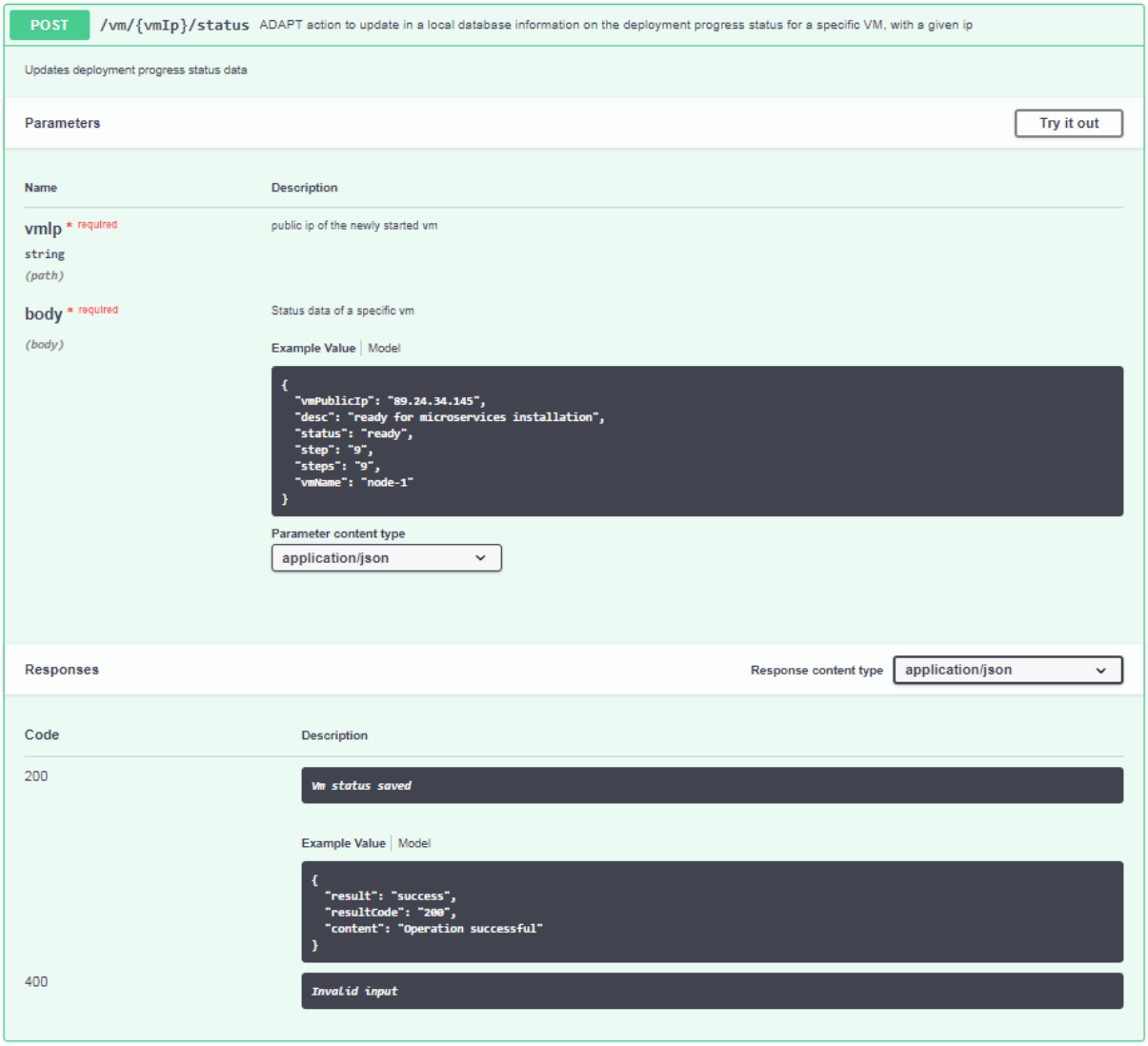

Figure 13. Endpoint for updating status information for a specific VM started on a CSP by ADAPT DO

Figure 14**Figure *14*** describe the GET counterpart to the previous endpoint,and allows to GET detailed information about the deployment status of a specific VM.

Accordingly, it requires as parameter the {vmIp} in the request path, and the response contains the following information:

- "vmPublicIp": Ip of the virtual machine
- "desc": textual description of the status. Can be one of the following:
  - "clean before update"
  - "update packages"
  - "configuring security"
  - "installing docker and docker compose"
  - "configuring docker as a service"
  - "installing consul"
  - "joining consul wan cluster"
  - "configuring access to private registry"
- "status": can be one of the following:
  - "configuring"
  - "ready"

- "step": the number of the current step being performed
- "steps": the total number of steps expected
- "vmName": the name of the VM as defined in the deployment topology

In case of error, it returns a 400 "not found" error.

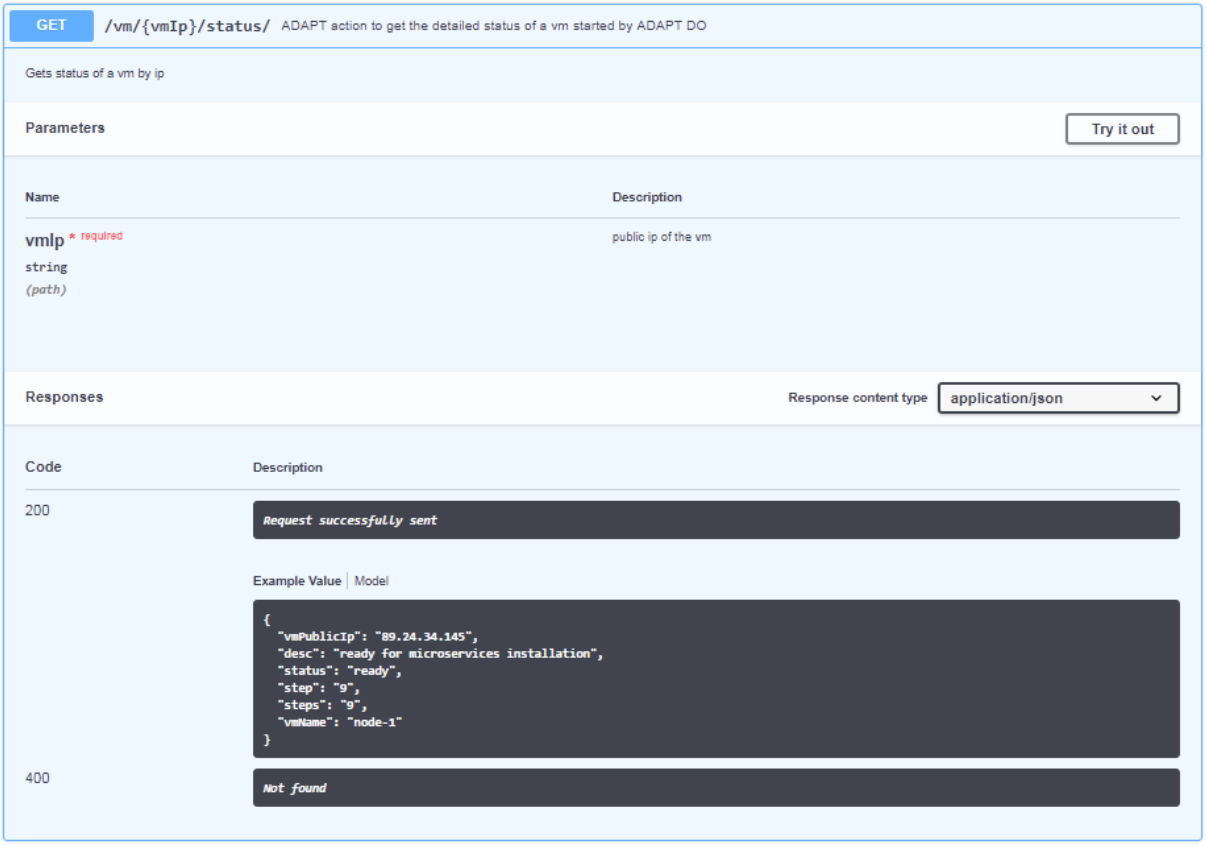

Figure 14. Endpoint for getting status information for a specific VM started on a CSP by ADAPT DO

3.2 Extensions for status data storage

The intermediate version of ADAPT DO extends the way status data related to VMs is stored. In the initial version status data was managed via a file-based mechanism, as the set of information that required to be stored was very limited. With the need for tracking the deployment status, the whole management of data related to the deployment (and in general to all ADAPT DO operations) has been replaced with a database-based mechanism. We adopted MongoDB, a lightweight, noSQL, document database which is particularly suitable to map software objects into database documents represented in the JSON format. Accordingly, also the software code has been updated in order to access the mongodb in place of the filesystem.

3.3 Extensions in input/output data

With respect to the initial version of ADAPT DO, a few relevant changes have been introduced in:

- the input data provided to ADAPT DO via the REST APIs and via the Application Description
- the output produced by ADAPT DO and saved into the Application Description

3.3.1 Extensions in input data

3.3.1.1 Input in REST API parameters

To enable the monitoring functionality, which is triggered by ADAPT DO after the microservices of the application have been deployed and started, ADAPT DO needs to know the endpoint to invoke on the Monitoring Manager. Therefore, the body of the POST operation used to start the ADAPT DO operations has been extended with two additional required parameters: “monitoring_host” and “monitoring_port”, as depicted in **Figure *15***.

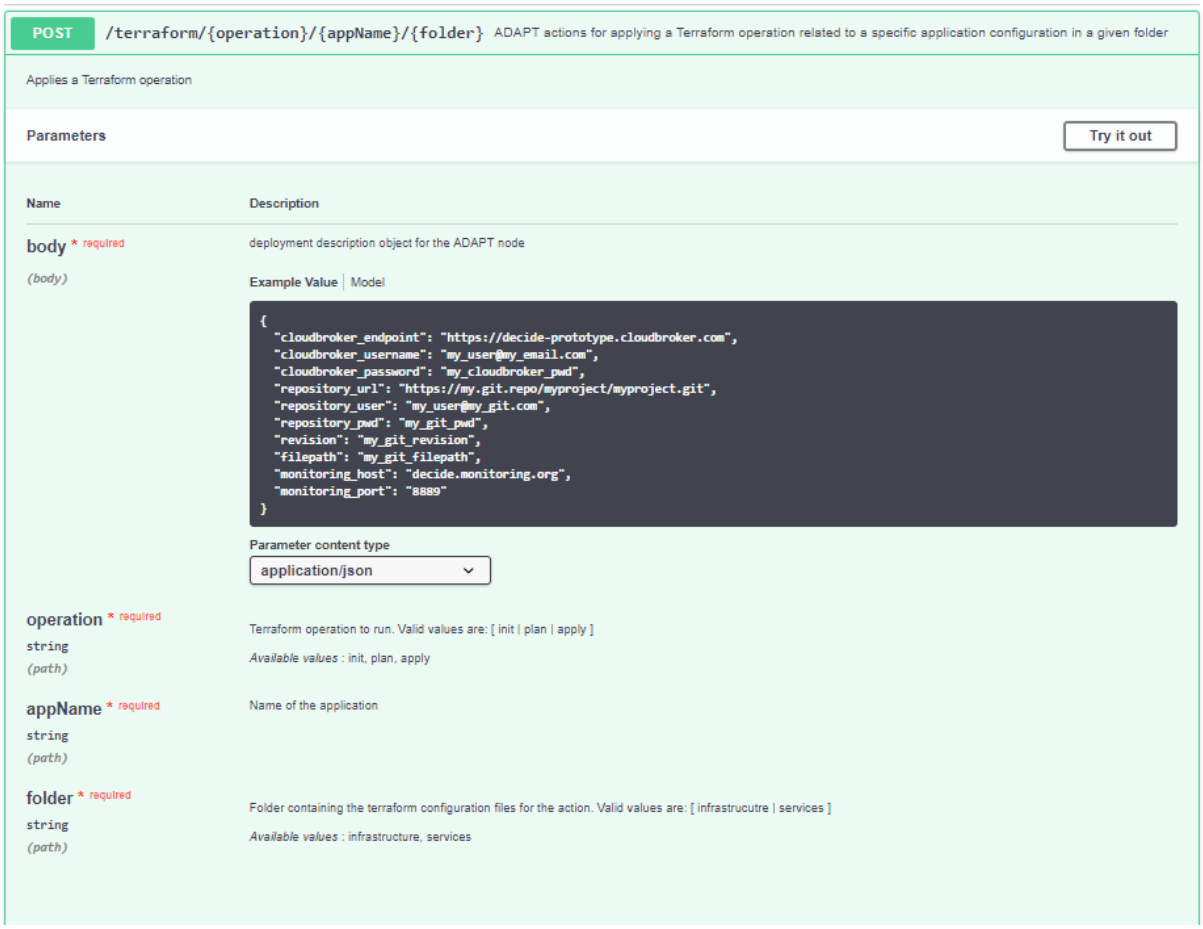

Figure 15. Extension in the body of the data POSTed for triggering ADAPT DO operations

3.3.1.2 Input in Application Description

In the initial version of ADAPT DO, the virtual machine images available on the CSP that could be managed by ACSmI had been prepared with a default user. Therefore, ADAPT DO was using that user (in an “hardcoded” style) to connect to the VMs that were started on the cloud infrastructures. With the addition of new Cloud Providers to the set of CSPs managed by ACSmI, additional images were introduced, which were configured with different users. Therefore, ADAPT DO was extended to make parametric the management of the VM user. This addition had several implications in the source code, not only in ADAPT DO code but also in the custom Terraform plugin (“helper”) developed in the project to extend the Terraform capabilities to the management of the ACSmI interface (cf. D4.10).

This required that the input data must now be configured by the end user for every VM to be started up, and passed to ADAPT DO in the Application Description, specifically in the “virtualMachine” section, as in the example of Figure 16.

```
"virtualMachines": [
    {
        "vmSoftwareId": "21b7ebed-5076-43b6-8351-0e06cf16eedc",
        "vmResourceId": "27b1fb7b-2fd9-4042-8902-e4996e0bc420",
        "vmRegionId": "d112253a-951f-417a-a1f1-3a892bf35a45",
        "instanceTypeId": "f554e8a1-29d8-48e5-bd68-c01417576628",
        "vmUser": "root",
        "keyPairId": "dbe5cc42-1f33-4ca4-b761-aec1c46a76fe",
        "openedPort": "22,80,8000-9000,9411",
        "dockerPrivateRegistryIp": "54.221.168.175",
        "dockerPrivateRegistryPort": "8200",
        "dockerHostNodeName": "node-1",
        "dockerHostPublicIp": "82.223.100.123"
    },
```

Figure 16. Additional input field for ADAPT DO in the Application Description

3.3.2 Extensions in output data

ADAPT DO is the component responsible to interact with ACSmI for starting up new VMs on the CSPs. The public ip address of a VM, which allows to connect to the VM services, is not known in advance, but is typically assigned automatically by a CSP upon the VM startup. Therefore, being ADAPT DO the component which manages the deployment of the VMs and that checks the starting status, it is the only one that can detect such information and pass it to the other DECIDE components. Since the main data exchange mechanism between DECIDE components is the Application Description file, ADAPT DO updates such file and pushes if to the Git repository as soon as the VMs are fully started up and configured on the CSPs. Here the summary of the update flow:

- When triggered via a POST request for starting an infrastructure ADAPT DO gets the Application Description file
- ADAPT DO applies all the operations required to startup and configure VMs on the CSPs (via ACSmI and direct ssh access to the VMs)
- When VMs have been provisioned and configured, ADAPT DO updates the Application Description file by filling in the VM ip in the VirtualMachines section of the Application Description, under the “dockerHostPublicIp” field (cf. Figure 17)
- ADAPT DO pushes the Application Description file to the Git repository, so that all the other DECIDE components can use that information

```
    {
        "vmSoftwareId": "21b7ebed-5076-43b6-8351-0e06cf16eedc",
        "vmResourceId": "27b1fb7b-2fd9-4042-8902-e4996e0bc420",
        "vmRegionId": "d112253a-951f-417a-a1f1-3a892bf35a45",
        "instanceTypeId": "f554e8a1-29d8-48e5-bd68-c01417576628",
        "vmUser": "root",
        "keyPairId": "dbe5cc42-1f33-4ca4-b761-aec1c46a76fe",
        "openedPort": "22,80,8000-9000,9411",
        "dockerPrivateRegistryIp": "54.221.168.175",
        "dockerPrivateRegistryPort": "8200",
        "dockerHostNodeName": "node-2",
        "dockerHostPublicIp": "82.223.2.216"
    }
```

Figure 17. New field "dockerHostPublicIp" returned by ADAPT DO in the Application Description

In addition to the ips of the VMs, ADAPT manages also a unique identifier which allows to uniquely associate an Application Description to runtime instances of VMs. This information is required in particular by the monitoring (ADAPT MM), as it allows to map monitoring instances to a specific deployment. Such identifier is generated by ADAPT DO itself, and returned, in the same way of the VMs ips, to the other DECIDE components by updating and pushing to the Git repository the Application Description file, as in the snippet of:

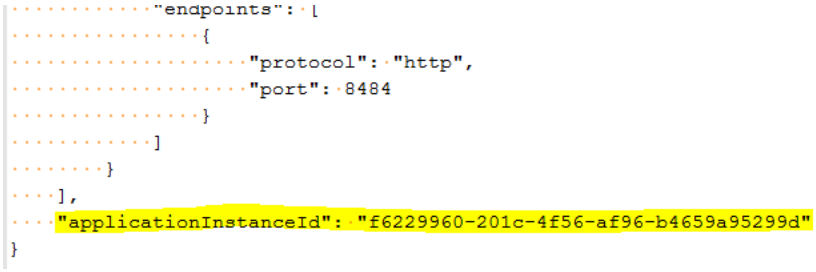

Figure 18. THe new output field applicationInstanceId populated by ADAPT DO

3.4 Extensions on helpers

The Helpers are modules tightly linked with the main DECIDE ADAPT components, Deployment Orchestrator and Monitoring Manager, defined in task T4.4. The initial documentation of the Helpers in M12 was provided in the dedicated deliverable D4.10. After the mid-term review, the Consortium and the Reviewers agreed on a plan to consolidate some deliverables, merging a few very focused documents into deliverables strictly connected. This would reduce the overall effort for the Reviewers to access the information on the delivered components, and for the Consortium to manage the editing, reviewing and releasing process of the documents. As part of this plan, an amendment was proposed and accepted for merging deliverable D4.11 into D4.5 and, in general, to document T4.4 in the same deliverable documenting T4.2 for the future versions of the deliverables.

In accordance to the amendment, we describe here the updates to the Helpers during the timeframe M12-M24. Following the same approach of the rest of the document, we will not rewrite contents from D4.10 but just report the additions or modifications. Therefore, to have a full understanding of the Helpers the reading of D4.10 is a prerequisite.

3.4.1 Terraform Cloudbroker plugin and Preparation scripts

Regarding the Terraform Cloudbroker provider plugin, documented in Section 3.1 of D4.10, the following additions were implemented:

- The capabilities of the plugin have been extended to support firewall rules, which allow important security improvements as they make possible to filter the access to specific ports and protocols. As an example, they make possible to access a specific port of a VM to a given ip or to a set of ips belonging to a defined subnet. This is very important because ADAPT DO can automatically define at VM startup the firewall rules by allowing communication on specific ports only to the VMs belonging to the deployment topology defined by the DECIDE workflow execution. This feature will be fully exploited by ADAPT DO in the final release that will be documented in the final version of the deliverable related to T4.2 (D4.6)
- The plugin has also been extended to support the management of a specific user for the virtual machines started on the CSP side (cf. Section 3.3.1.2). This was needed as during the plugin execution there are a set of operations for configuring VMs that must be carried on and that require ssh access to the VMs, and this requires knowing the default vm user defined in the image used in a specific cloud infrastructure, that may be arbitrary.

Regarding the Preparation Scripts documented in Section 3.2 of D4.10, they underwent a general refactoring to make them more efficient, to improve the error management logic, to login automatically to private Docker registries at service deployment time, to handle certificates generation in a parametric way. In addition, they were extended to support the deployment status reporting. This last feature has been already documented in Section 2.2.

3.4.2 Extensions in the set of supported cloud providers: private cloud

In addition to the updates mentioned in Section 3.4.1, a set of additions were made to enable the demonstration of an important scenario: the private cloud scenario.

The set of supported cloud providers has been extended in the timeframe M12-M24: to the set of CSPs supported in M12, namely AWS and Cloudsigma, the support for an additional public CSP Arsys (https://www.arsys.net/) was introduced. In addition, as we wanted to address also a multi-cloud deployment involving a Hybrid cloud scenario (cf. Section 2.5), the support for Openstack-based cloud platforms was added, being Openstack one of the main platforms used for private clouds. ADAPT DO relies on ACSmI for connecting to the CSPs, therefore these extensions are indeed not strictly implemented by ADAPT DO, rather by ACSmI, and documented in the corresponding deliverables. Anyway, regarding the Openstack cloud platforms, which are the primary ones used by organizations that want to set up a private cloud, ADAPT DO is being extended to optionally support direct connection to the Openstack API without intermediation of ACSmI. The rationale behind this extension is that the connection with ACSmI requires necessarily that an organization enables specific "communication paths" to the ACSmI platform, by opening some ingress traffic on specific ports to let ACSmI invoke the local Openstack APIs. In some scenarios this option is not applicable, as some organizations do not allow ingress traffic to their network for security policies. To satisfy this private cloud scenario, we are developing an adaptor that can connect directly to a local Openstack cloud so that, if the ADAPT DO is deployed itself within the private network of an organization, it can still perform the deployment within that network.

This feature will be used by adding a new flag in the "VirtualMachines" section of the Application Description, which allows ADAPT DO to discriminate whether instantiating VMs through ACSmI or via direct access to an Openstack API. This selection would trigger the creation of dedicated Terraform configuration scripts for Openstack via pre-defined parametric templates, in the same way as the ones that are currently created for the VMs started by ACSmI.

In accordance with the documentation provided in D4.4, Section 2.2.2.2, we provide in **Figure *19*** an excerpt of the Openstack Terraform plugin configuration template, that allows to create dynamic configurations at runtime via ADAPT DO for instantiating VMs on a private Openstack cloud. The template contains information specific to the Openstack API, such as the tenant_name, the API auth_url, and all the other items required to specify type of VM, size, network, key_pairs, etc.

Currently (M24) we are finalizing the development of the adaptor, and the results of the usage will be documented in the final version of this deliverable (D4.6).

```
provider "openstack" {
  user_name   = "${var.user_name}"
  tenant_name = "${var.tenant_name}"
  password    = "${var.password}"
  auth_url    = "${var.auth_url}"
  region      = "${var.region}"
}

resource "openstack_compute_instance_v2" "decide-vm" {
  name            = "${var.node_name}"
  image_id        = "${var.image_id}"
  flavor_id       = "${var.flavor_id}"
  key_pair        = "${var.key_pair}"
  security_groups = "${var.security_groups}"

  block_device {
    uuid                  = "6ad2bbf2-041c-4b75-98e1-e9935f7e6a01"
    source_type           = "image"
    destination_type      = "local"
    boot_index            = 0
    delete_on_termination = true
  }

  block_device {
    source_type           = "blank"
    destination_type      = "volume"
    volume_size           = 1
    boot_index            = 1
    delete_on_termination = true
  }

  network {
    name = "${var.network_name}"
  }
```

Figure 19. Excerpt of the configuration template for the Openstack Terraform plugin

3.4.3 Setup of a real private cloud in HPE

As introduced in Section 2.5 and 3.4.2, to enable both the hybrid cloud scenario and the private cloud scenario we need a private cloud environment that would allow us to test the developed components and validate the related use cases.

Therefore, as no partners in the project could provide a private cloud environment, we have set up in HPE an Openstack infrastructure in a private Lab, full-featured from the point of view of the functionalities provided, but "scaled-down" with respect to features like high-availability, data-replication, etc. These limitations, though, do not affect in any way the validity of the demonstrations, because all the relevant functional aspects are provided (VM provisioning, private virtual network provisioning, volume management, security groups handling) and, from the point of view of network security, the infrastructure is under all aspects compliant with the setup of a real enterprise cloud. It is indeed "isolated" from the public network via enterprise-level firewall configuration policies, and network traffic between the internal network and the internet is allowed only on ports 80 and 443.

Since the Openstack management API requires access to several endpoints managed by different ports, to enable the hybrid cloud scenario (with ACSmI instantiating VMs on our private Openstack infrastructure) we had to set up a complex set of reverse proxy rules so that the ACSmI interfaces could reach the Openstack management API on endpoints exposed on the allowed ports.

We also had to expose a public endpoint with a public Domain Name and have it validated by a Certification Authority in order to make the whole mechanism work.

Currently the Openstack platform can be reached at the public url:

https://decideos.mycloud.bz/horizon

Our current Openstack deployment is running the main set of Openstack services such as nova-compute, neutron, cinder, keystone, and horizon for the Dashboard. As for the hardware, it runs on a set of 7 HP BL 460 G6 blade servers installed in an enclosure HP c3000 and connected to a private

network. The private network is, in turn, connected to the internet through a HPE Company Managed network infrastructure with very strict policies, whose port opening rules are managed at worldwide level.

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:m24
```

- 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    m24         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:m24
```

- 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    m24          4537a4171c5a        12 days ago         1.69GB
```

- Run the image:

```
shell> docker run -p 8473:80 -d --name adapt aaa.bbb.ccc.ddd:nnnn/adapt:m24
```

- 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    m24    4537a4171c5a     12 days ago   1.69GB
```

4.2.3 Building the image from code

The code has been provided as part of the zip file delivered to the EC.

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

The repository can be easily accessed via the project Gitlab web GUI. To build the ADAPT DO image from code for the release atM24, follow the steps below:

- Click on the repository url and, after logging in, select the tag "m24" from the drop-down menu containing the available project branches and tags, as shown in **Figure *20***

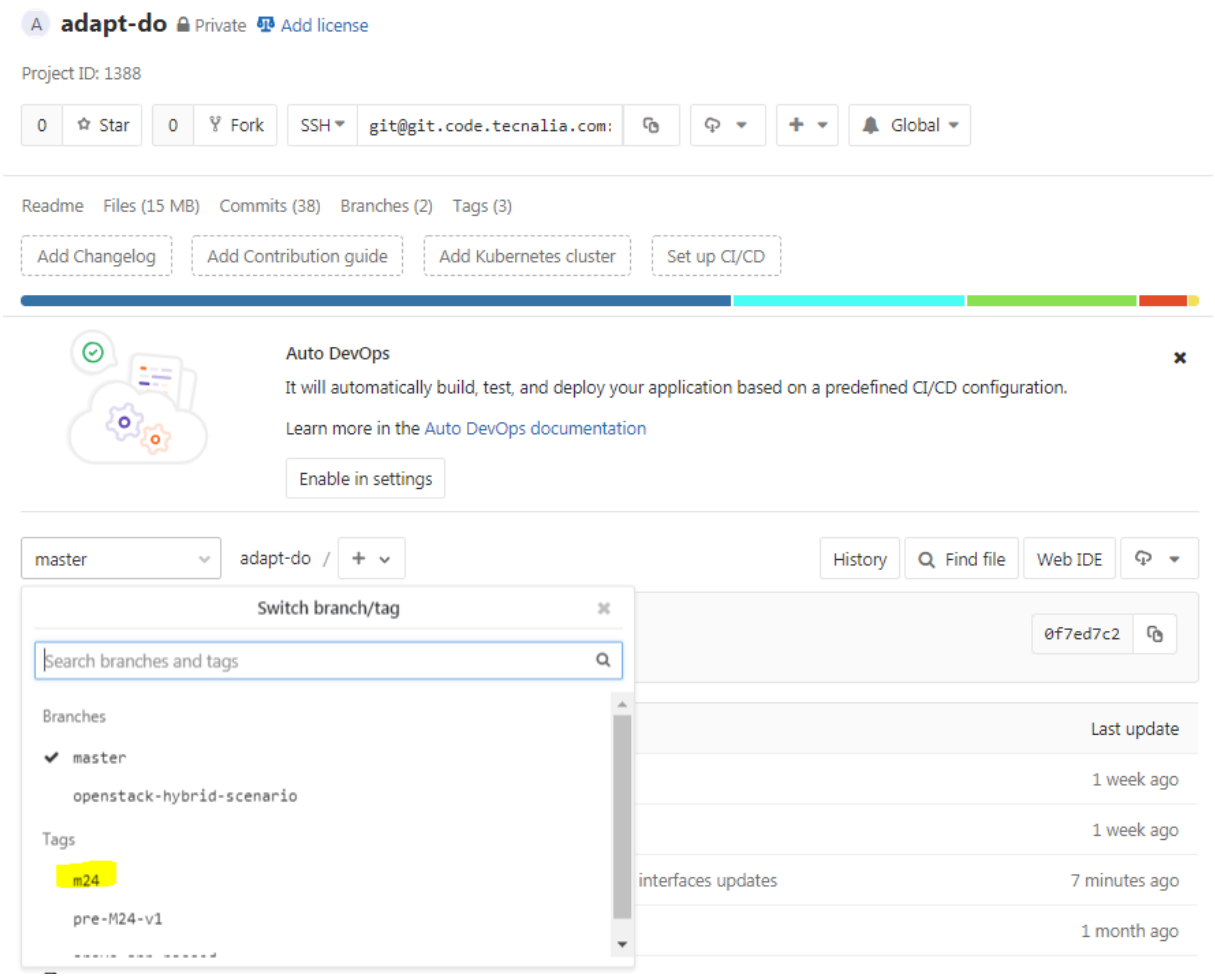

Figure 20. Selecting the M24 tag for getting the ADAPT DO source code released at M24

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

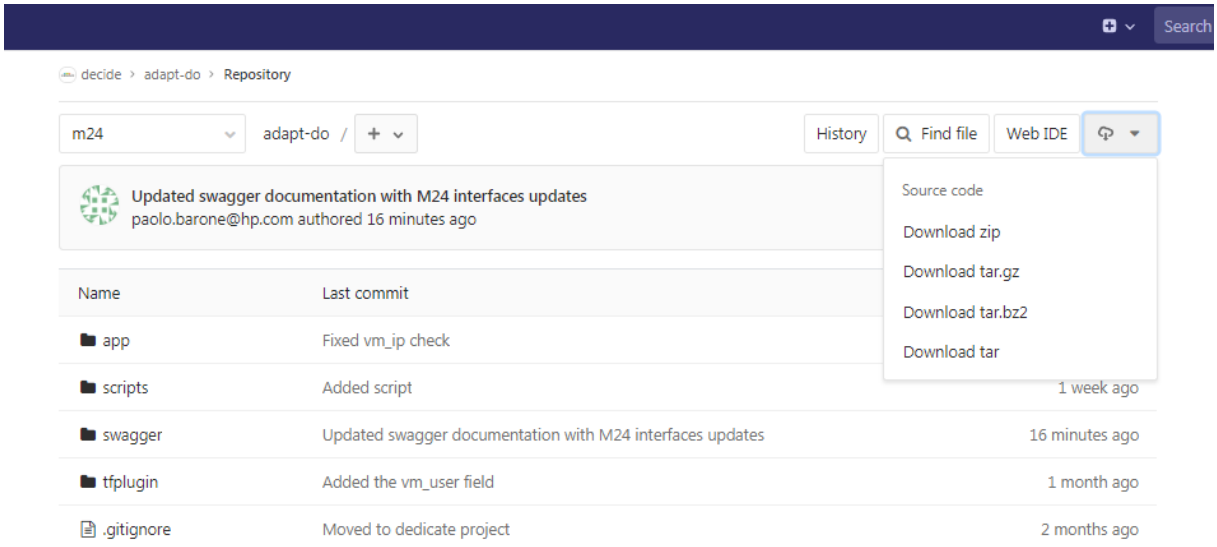

Figure 21. Downloading the source code

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

- Run the command:

```
shell> sudo docker build -t adapt:m24 .
```

The above command builds the image 'adapt' with tag 'm24.

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

```
shell> docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
adapt                       m24                 4537a4171c5a        12 days ago         1.69GB
```

- Run the image:

```
shell> docker run -p 8473:80 -d --name adapt adapt:m12
```

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

```
shell> docker ps
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
adapt                       m24                 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, Section 2.2.2 and updated in Section 3.1 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 2.3, 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 using the CLI tools documented in Section 2.3, 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 follows 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 contains 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 has to follow an internal process with reviews and decisions at corporate level to decide and approve the license under which to release the developed software. Unfortunately, this process takes time and it is not yet completed at the time of writing, therefore the licensing information for the released software is not yet available. However, credentials can be provided under request, 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, has described the updates and extensions implemented during the period M12-M24 to the ADAPT DO prototype released in M12. The outcome is the ADAPT DO release M24. The prototype is responsible of the deployment and adaptation functionalities of DECIDE.

Taking as the basis the deliverable D4.4, which documented the main functionalities and their relationship with the ADAPT architecture documented in D4.1 and with the rest of the DECIDE components, this deliverable documents the extensions and the updates developed till M24.

Updates about the packaging of the component, about the requirements to install it and on the steps a user would have to follow to test it have been added as well.

This deliverable 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.