Deliverable D4.10

Initial multi-cloud application helpers

Editor(s):	Lorenzo Blasi
Responsible Partner:	HPE
Status-Version:	Final - v1.0
Date:	30/11/2017
Distribution level (CO, PU):	PU

Project Number:	GA 726755
Project Title:	DECIDE

Title of Deliverable:	Initial multi-cloud application helpers
Due Date of Delivery to the EC:	30/11/2017

Workpackage responsible for the Deliverable:	WP4 - Continuous deployment and operation
Editor(s):	HPE
Contributor(s):	HPE
Reviewer(s):	Leire Orue-Echevarria (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP5

Abstract:	The deliverable reports the most relevant technical details of the helpers specifically developed for applications or parts of applications. This is the initial version of the deliverable.
Keyword List:	Helpers, plugin, scripts, Terraform, provisioning
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This deliverable reflects only the author's views and views and the Commission is not responsible for any use that may be made of the information contained therein

Document Description

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	30/10/2017	First draft version	HPE
V0.2	8/11/2017	Added technical descriptions	HPE
V0.3	10/11/2017	Added Exec Summary and Conclusions	HPE
V0.4	13/11/2017	Finalized all the sections	HPE
V0.5	23/11/2017	Updated after internal review	HPE
V1.0	23/11/2017	Ready for submission	TECNALIA

Table of Contents

Table of Contents 4
List of Figures 4
List of Tables 4
Terms and abbreviations 5
Executive Summary 6
1 Introduction 7
1.1 About this deliverable 7
1.2 Suggested reading path 7
1.3 Document structure 7
2 Functional description 8
2.1 Fitting into overall DECIDE ADAPT Architecture 9
3 Technical description 11
3.1 The Terraform ACSmI provider plugin 11
3.2 The Preparation scripts 14
4 Conclusions 16

List of Figures

FIGURE 1. DEPLOYMENT ORCHESTRATOR COMPONENT IN THE ADAPT ARCHITECTURE 10
FIGURE 2. THE ACSMI PROVIDER PLUGIN IN THE DEPLOYMENT ORCHESTRATOR ARCHITECTURE 10
FIGURE 3.- RELATIONSHIP BETWEEN TERRAFORM LOGICAL COMPONENTS AND THE PLUGIN 13

List of Tables

TABLE 1. MAPPING BETWEEN REQUIREMENTS AND FUNCTIONALITIES, WITH DETAILED COVERAGE 9

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

The main objective of DECIDE project is to provide a novel software framework to design, develop, and dynamically deploy multi-cloud applications. ADAPT is the tool of the DECIDE framework which aims to allow continuously deploy and dynamically self-adapt multi-cloud applications.

This deliverable is a result of DECIDE Work Package 4 ("continuous deployment and operation") and reports the work done in the first year (Y1) in Task T4.4 (Multi-cloud application helpers) in order to implement the Helpers.

The Helpers are modules tightly linked with the main DECIDE ADAPT components, Deployment Orchestrator and Monitoring Manager. Those two components are described in detail in their specific deliverables, i.e. respectively D4.4 [1] (Initial multi-cloud application deployment and adaptation) and D4.7 [2] (Initial multi-cloud application monitoring). The overall ADAPT architecture and design is described in deliverable D4.1. The suggested reading order for Y1 WP4 deliverables is the following: D4.1, [3] D4.4 [1], D4.7 [2], D4.10 (this deliverable).

This document starts with a functional description (section 2) of the Helpers developed in year 1 (Y1), enriched with a mapping between the functionalities and ADAPT requirements.

DECIDE ADAPT, as already described in D4.4, uses the open source Terraform[1] tool to apply provisioning and deployment actions. The two Helpers developed in Y1 are described in more detail in section 0; they support the functionalities of Terraform and its integration with the rest of the DECIDE framework: the Terraform Plugin allows ADAPT to interface with ACSml's implementation and the set of Preparation scripts to enable setting up the runtime environment for the execution of the multi-cloud application containers.

In the next year further Helpers will be developed, such as monitoring probes to collect application-specific metrics and the optional reverse-proxy component to maintain microservices communication even in case of migration. These new Helpers will be described in the WP4 deliverables planned for Y2.

[1] https://www.terraform.io

1 Introduction

1.1 About this deliverable

This deliverable reports the work done in Task T4.4 (Multi-cloud application helpers) in the first year (Y1) of the DECIDE project's progress. The Helpers, as indicated in deliverable D4.1 (Initial DECIDE ADAPT Architecture), are *ADAPT modules planned for implementing the low-level logics for the deployment steps, the retrieval of monitoring data, the actions for adapting applications and implementing what is required for interfacing different cloud platforms*.

1.2 Suggested reading path

The architecture of DECIDE ADAPT is described in deliverable D4.1 [3], which is the first document to read for understanding ADAPT's requirements, functionalities and architecture.

The Helpers are modules tightly linked with the main DECIDE ADAPT components, Deployment Orchestrator and Monitoring Manager. The detailed description of those two components is included in their specific deliverables, i.e. respectively D4.4 (Initial multi-cloud application deployment and adaptation) [1] and D4.7 (Initial multi-cloud application monitoring) [2]. This deliverable will not duplicate concepts and details already expressed in D4.4 [1] and D4.7 [2], therefore the suggestion to the interested readers is to read those two documents in advance, to better understand what is described in this document.

The suggested reading order for Y1 WP4 deliverables is therefore: D4.1 [3], D4.4 [1], D4.7 [2], D4.10.

1.3 Document structure

This document is composed of two main sections. The first, section 2, describes what the Helpers are and how they fit in the DECIDE ADAPT architecture. The second, section 0, provides technical details about the first implementation. The implemented software for the Helpers in Y1 is released as part of the Deployment Orchestrator deliverable, D4.4 [1], therefore this document does not include packaging and installation information.

2 Functional description

The Helpers, as indicated in deliverable D4.1 (Initial DECIDE ADAPT Architecture) [3], are "*ADAPT modules planned for implementing the low-level logic for the deployment steps, the retrieval of monitoring data, the actions for adapting applications and implementing what is required for interfacing different cloud platforms*".

Several low-level modules of DECIDE ADAPT will satisfy the definition above and can be considered as Helpers. Possible examples are:

- Any monitoring probe implemented to collect application-specific metrics, or possibly a template of such a probe available for the application developer to customize.
- The set of "Preparation scripts" responsible for installing pre-requisite software (such as Docker, Consul, etc.) in each VM of the multi-cloud application infrastructure (likely managed by Terraform "provisioners" defined in the Terraform configuration for the VM resources).
- The "Terraform plugin" implemented to interface with ACSmI implementation to start and release infrastructure resources.
- The optional reverse-proxy component that can be used by applications to define automatically the routing between microservices endpoints and the containers which provide them

This deliverable describes the Helpers implemented in Y1: the Terraform Plugin to interface with ACSmI implementation and the set of Preparation scripts enabling the runtime environment for the multi-cloud application components.

The Terraform plugin component has been indicated as "ACSmI provider plugin" in deliverable D4.1 [3](sect. 4.1, Deployment components), and described as follows: *the ACSmI provider plugin allows to use ACSmI as an IaaS provider for provisioning the infrastructure resources defined in the deployment plan. This plugin can be considered a Helper since it allows to interface the underlying cloud platforms*.

As indicated in Terraform online documentation[2], "*a provider in Terraform is responsible for the lifecycle of a resource: create, read, update, delete. An example of a provider is AWS, which can manage resources of type* `aws_instance`, `aws_eip`, `aws_elb`, etc."

The ACSmI Provider Plugin can manage resources of type `cloudbroker_instance`, which are Virtual Machines of the specific Cloud Provider indicated in the Application Description, and allows to create, read, update, and delete them.

The main functionalities planned so far for the ACSmI provider plugin are the following.

F1. Request to ACSmI the creation of an infrastructure resource, given its type, the target Cloud Provider, and various initialization parameters read from the Application Description
F2. Request to ACSmI to read the information available about a given existing infrastructure resource
F3. Request to ACSmI to update the configuration of a given existing infrastructure resource
F4. Request to ACSmI to delete / release a given existing infrastructure resource

The Preparation scripts cannot be identified with a specific ADAPT architectural component; they are part of the deployment scripts needed for the automation of resource provisioning indicated in requirement WP4-REQ12. The main functionalities of the Preparation scripts are the following

F5. Generate credentials to access the Virtual Machines (VMs)
F6. Install and configure the container runtime (Docker) on those VMs

[2] https://www.terraform.io/docs/plugins/provider.html

F7. Install and configure the support for VMs' and services' health checks (Consul)

The listed functionalities are implemented using an incremental approach over multiple releases, also depending on the functionalities available in the ACSmI API. In this first release the following functionalities are implemented: **F1**, **F2**, **F4**, **F5**, **F6**, **F7**.

The following **Table *1*** details the relationship between requirements indicated in deliverable D4.1 and the implemented functionalities, with a description of the coverage for each functionality.

Table 1. Mapping between requirements and functionalities, with detailed coverage

Functionality	Req. ID	Coverage
F1	WP4-REQ31[3]	The prototype ACSmI Provider Plugin supports requesting to ACSmI the creation of VM resources from any of the supported Cloud Providers (currently AWS and CloudSigma). Parameters for the request are read from the Application Description.
F2	WP4-REQ31	The prototype reads the IP address assigned to the created VM.
F3	WP4-REQ31	None.
F4	WP4-REQ31	The prototype supports requesting to ACSmI the release of a VM resource given its ID.
F5	WP4-REQ12[4], WP4-REQ31	The prototype generates certificates and access keys for each created Virtual Machine
F6	WP4-REQ12, WP4-REQ31, WP4-REQ36[5]	The prototype installs and configures Docker on each created Virtual Machine, also setting up the credentials needed to access the image repository
F7	WP4-REQ12, WP4-REQ31	The prototype installs Consul and configures it to join a WAN cluster, where ADAPT is the master node

2.1 Fitting into overall DECIDE ADAPT Architecture

The ADAPT Deployment Orchestrator, as already indicated in deliverable D4.4 [1], is part of the ADAPT architecture as shown in **Figure *1***. The "iObtainRelease resources" interface exported from ACSmI and used by the Deployment Orchestrator is exactly the API invoked through the ACSmI Provider Plugin to create or delete Virtual Machines from a specific Cloud Provider.

[3] WP4-REQ31: Helper modules implement the logics for the deployment steps, the retrieval of the monitoring actions, the actions for adapting applications and implementing the actions required for interfacing different cloud platforms.

[4] WP4-REQ12: DECIDE [..] will provide [..] automation of the provisioning resources and deployment scripts for multi-cloud native applications

[5] WP4-REQ36: ADAPT will support modular applications where each composition unit is a containerized service

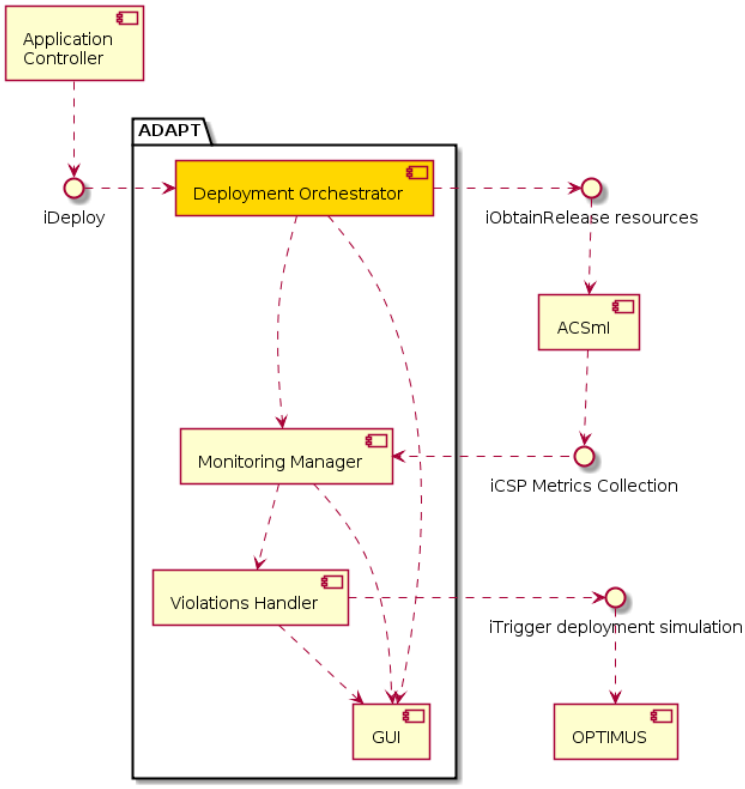

Figure 1. Deployment Orchestrator component in the ADAPT architecture

The ACSmI Provider Plugin is shown in **¡Error! No se encuentra el origen de la referencia.** **Figure *2*** as part of the ADAPT Deployment Orchestrator component.

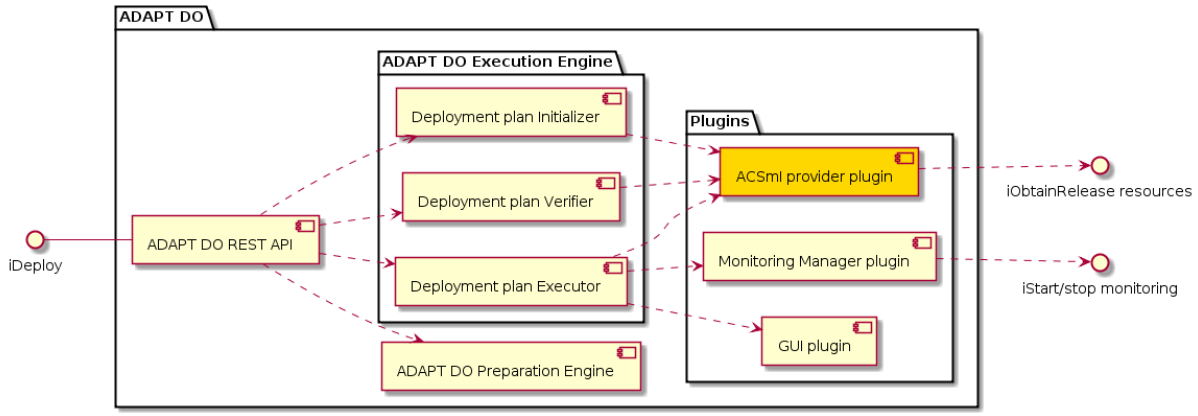

Figure 2. The ACSmI Provider Plugin in the Deployment Orchestrator architecture

The Preparation scripts are part of the ADAPT Deployment Orchestrator package, and are used by the Preparation Engine and by the Execution Engine when creating and applying the deployment actions respectively.

3 Technical description

This section describes the technical details of the implemented software.

3.1 The Terraform ACSmI provider plugin

ADAPT Deployment Orchestrator uses the Terraform tool to apply provisioning and deployment actions. This mechanism is widely documented in deliverable D4.4 [1] and will therefore not be described here.

The Terraform interaction with external systems (cloud environments, configuration management tools, databases, etc.) is based on a plugin mechanism. Terraform provides libraries and schemas for defining interfaces, data structures, configuration keys and callbacks, which allow to abstract many of the complexities and ensure consistency between providers. With such libraries at hand, a developer can extend the Terraform capabilities by implementing new plugins for connecting to private clouds and for managing custom tools and resources.

In DECIDE, we rely on the ACSmI component as the central manager for the selection of cloud resources. The ACSmI component acts as a cloud broker, and the provisioning of cloud resources is mediated by a REST API which is then the single access point to the clouds.

To allow the ADAPT Deployment Orchestrator to interact via Terraform with such API, we developed a custom Terraform plugin, extending the Terraform functionalities with new capabilities.

The implementation follows the directions specified by the dedicated section of the Terraform documentation[6], which we briefly summarize in the following.

Terraform and the related plugins are implemented in the Go Language[7], a modern, compiled programming language optimized for modern multicore and networked machines. Pre-requisite for developing a Terraform plugin is to install the Go Language programming environment[8]. In order to compile, the Terraform source code is also needed locally. It is available for download from the Terraform Github public area[9].

A Terraform plugin must import the Terraform schemas and core libraries (cf. snippet code below) and implement a set of interfaces for providing access to resources and for their manipulation.

```
package main

import (

        "github.com/hashicorp/terraform/helper/schema"

        "github.com/hashicorp/terraform/plugin"

        "github.com/hashicorp/terraform/terraform"

[..]

)
```

Our implementation provides a mapping between Terraform and the ACSmI component for the creation, manipulation and deletion of a virtual machine on a cloud provider.

The plugin implements first the definition of the data structures specific to the ACSmI virtual machines, extending the Terraform schemas with plugin-specific resource definitions. The interfaces for virtual

[6] https://www.terraform.io/guides/writing-custom-terraform-providers.html
[7] https://golang.org/
[8] https://golang.org/doc/install
[9] https://github.com/hashicorp/terraform

machine management are then implemented by mapping the Terraform operations and the equivalent REST calls required on the ACSmI API. The operations consist of:

- Creation of a virtual machine, based on parameters defining the size, OS image, architecture, opened ports, etc. in the format understood by the ACSmI API.
- Reading of the virtual machine properties at runtime, such as the id or the IP addresses assigned by the provider after creation.
- Deletion of the virtual machine.

The implemented code must be provided into a 'main.go' file, or in files imported by a main.go file, which acts as the entry point for the plugin execution. The code is then compiled via the Go 'build' tool, which generates as output a binary file.

To adhere to the Terraform plugin specification, the plugin binary file must be named after the following convention:

```
terraform-provider-'name'
```

Since the ACSmI functionality which we are using in this case is that of a cloud broker, the plugin binary has been named 'cloudbroker' and it is built as:

```
terraform-provider-cloudbroker
```

This binary file must then be made available to the Terraform installation of any ADAPT Deployment Orchestrator. As specified in deliverable D4.4 [1], the ADAPT Deployment Orchestrator is packaged into a Docker image. For this reason, the plugin binary is copied in the PATH of the Terraform binaries at image creation, as defined in the corresponding Dockerfile:

```
FROM tiangolo/uwsgi-nginx-flask:python3.6

RUN apt-get update && apt-get install unzip && wget https://releases.hashic
orp.com/terraform/0.10.7/terraform_0.10.7_linux_amd64.zip?_ga=2.121414664.1
02068769.1507033863-2054770415.1501495729 -O temp.zip && unzip temp.zip -d
/usr/local/bin && rm temp.zip && mkdir -p /app/repo && mkdir -p /home/ubunt
u/terraform/certs && mkdir -p /home/ubuntu/terraform/scripts && mkdir /home
/ubuntu/terraform/keypairs && pip install flask-restplus

COPY app/ /app/

COPY tfplugin/terraform-provider-cloudbroker /usr/local/bin

COPY scripts /home/ubuntu/terraform/scripts


ENV STATIC_INDEX 1
```

The Dockerfile above shows the steps for the creation of the ADAPT Deployment Orchestrator image. Starting from a pre-packaged image providing the flask framework deployed on a nginx http engine, a

set of installation steps are performed to update the package libraries and to install Terraform and other tools on the system.

After that, with the 'COPY' instructions, a set of files and folders are copied into the image. The highlighted line (in bold and in a bigger font) shows the copy of the terraform plugin binary into the '`/usr/local/bin`' folder of the image, which is in the execution PATH of the system. Terraform can then find the plugin when a resource configuration specifies it as provider.

The diagram in **Figure** shows the relationship between Terraform, the ACSmI plugin and ACSmI itself. The Terraform ACSmI plugin implements the interfaces defined by the schemas provided by the Terraform core distribution. If the plugin is available in the execution path of the Terraform core binaries, the Terraform commands (init, plan, apply) can manage resources configured for using the "cloudbroker" provider implemented by the ACSmI plugin.

The 'init' command is responsible for verifying first that the plugin exists and is available. If the check successful, the 'plan' and 'apply' commands can be executed.

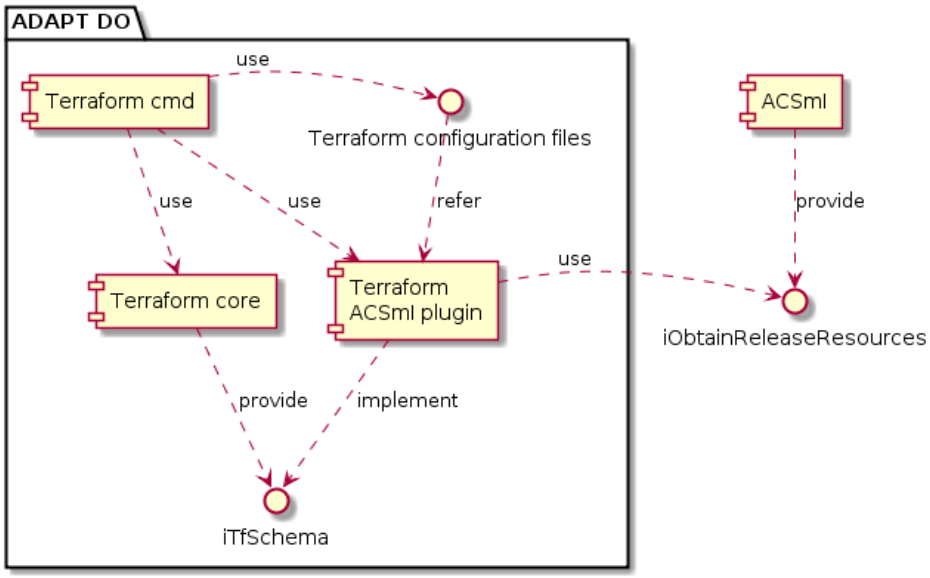

Figure 3.- Relationship between Terraform logical components and the plugin

A sample resource configuration involving the plugin is shown in the following excerpt, taken from deliverable D4.4 [1]:

```
provider "cloudbroker" {
  username    = "${var.cloudbroker_username}"
  password    = "${var.cloudbroker_password}"
  endpoint    = "${var.cloudbroker_endpoint}"
  timeout     = 60
  max_retries = 5
}

resource "cloudbroker_instance" "my-virtual-machine" {
  software_id = "${var.vm_software_id}"
  resource_id = "${var.vm_resource_id}"
```

```
  region_id = "${var.vm_region_id}"
  instance_type_id = "${var.instance_type_id}"
  isolated = "false"
  key_pair_id = "${var.key_pair_id}"
  disable_autostop = "true"
  opened_port = "${var.opened_port}"
  name = "${var.node_name}"
}
```

The configuration defines a 'cloudbroker' provider; Terraform init, while parsing the configuration file, checks for the existence of a 'terraform-provider-cloudbroker' in the execution path.

When running the Terraform 'plan' or 'apply' command, Terraform uses the plugin to create a 'cloudbroker_instance' resource, using the methods of the plugin which implement the 'create' functionality. This triggers a REST call toward the ACSmI which POSTs a request for the creation of a new virtual machine, with the specified parameters.

When running a Terraform 'destroy' command, the corresponding 'delete' method of the plugin is invoked, which in turn triggers the REST call toward the provider for stopping and removing the virtual machine.

3.2 The Preparation scripts

The Preparation scripts are a collection of utilities that are needed to prepare the runtime environment for the multi-cloud application components. They comprise both the installation of tools required to run specific commands and the configuration steps enabling access to resources.

In the DECIDE scenario, we are targeting multi-cloud applications obtained by composition of services provided by components packaged into containers. In particular, we are adopting Docker containerization as the base technology for the applications. Therefore, the virtual machines which define the application infrastructure are based on Linux (the reference environment for Docker) and the scripts are text files written in a scripting language that can be executed by the Linux command line interpreter.

The preparation scripts are handled in two phases by the ADAPT Deployment Orchestrator:

- During the dynamic generation of the Terraform configuration files related to the creation of the virtual machines, by the ADAPT Deployment Orchestrator Preparation Engine;
- During the provisioning of the virtual machines, by the ADAPT Deployment Orchestrator Execution Engine.

In the first case, the Preparation Engine includes references to such script files into the templates defining the virtual machine resources, more specifically into the 'file', 'local-exec' and 'remote-exec' provisioners sections.

In the second case, The Execution Engine runs the scripts according to the configurations defined in the provisioners sections (cf. deliverable D4.4, section "*ADAPT Deployment Orchestrator Execution Engine"* for details [1]). A 'file provisioner' takes care of copying the scripts from the ADAPT Deployment Orchestrator to the virtual machine. A 'remote-exec' provisioner executes the scripts remotely, after connecting to the virtual machine via SSH. A 'local-exec' provisioner copies back from the virtual machine to the ADAPT Deployment Orchestrator a set of files generated during the remote script execution, typically certificates that are required to connect remotely to the Docker daemon socket.

More specifically, the scripts executed on the virtual machines perform the following actions:

- Generate self-signed certificates and access keys on the virtual machine, based on the public IP assigned by the cloud provider.
- Install Docker on the virtual machine and start it as a daemon, configuring it to be accessible remotely via the certificates created at the previous step.
- Configure Docker to start as a daemon and restart it.
- Install Consul to allow basic health check on the virtual machines and services.
- Make the virtual machine join a Consul WAN cluster, where ADAPT is the master node.
- Configure the virtual machine with proper filepath and certificate to access a private Docker registry, required in case the application needs to pull Docker images from a private repository.

4 Conclusions

This deliverable reports about the Helpers, which are modules tightly linked with the main DECIDE ADAPT components. In Y1 two Helpers have been developed: the Terraform Plugin to interface with ACSmI component and the set of Preparation scripts enabling the setup of the runtime environment for the multi-cloud application components.

In the next year further Helpers will be developed, such as monitoring probes to collect application-specific metrics and the optional reverse-proxy component to maintain microservices communication even in case of migration.

References

[1] DECIDE Consortium, “D4.4. Initial multi-cloud application deployment and adaptation,” 2017.

[2] DECIDE Consortium, “D4.7 Initial multi-cloud application monitoring,” 2017.

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