Deliverable D4.1

Initial DECIDE ADAPT Architecture

Editor(s):	Lorenzo Blasi
Responsible Partner:	Hewlett Packard Italiana / HPE
Status-Version:	Final – v2.0
Date:	28/11/2017
Distribution level (CO, PU):	PU

Project Number:	GA 731533
Project Title:	DECIDE

Title of Deliverable:	Initial DECIDE ADAPT Architecture
Due Date of Delivery to the EC:	30/11/2017

Workpackage responsible for the Deliverable:	WP4 - Continuous deployment and operation
Editor(s):	Hewlett Packard Italiana s.r.l. (HPE)
Contributor(s):	ARSYS, EXPERIS, FhG, HPE, TECNALIA
Reviewer(s):	AIMES, TECNALIA
Approved by:	All Partners
Recommended/mandatory readers:	WP2, WP3, WP5, WP6

Abstract:	This deliverable describes the initial version of the architecture of DECIDE ADAPT tool providing a comprehensive overview of the system using different architectural views to represent different aspect of the system (e.g. Use Case View, Logical View, Process View, Deployment View, and Implementation View).
Keyword List:	Cloud-native, microservice, REST, stateless, container, virtual machine and orchestration system
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 document 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	28/07/2017	Defined document ToC	HPE
V0.2	01/08/2017	Agreed on responsibility for each section	HPE
V0.3	22/08/2017	Contribution on Application Description	FhG
V0.4	05/09/2017	Added Use Cases Diagram and descriptions; updated ToC with Experis contribution	HPE
V0.5	06/09/2017	Include merged version of Application Description and MCSLA	HPE
V0.6	07/09/2017	Added content for section on Improving business continuity	HPE
V0.7	08/09/2017	Integrated first Tecnalia contribution	TECNALIA, HPE
V0.8	08/09/2017	Modifications during WP4 call	HPE
V0.9	12/09/2017	Added sections for requirements analysis and requirements mapping; added UCUI03 to the UC diagram; added text for use cases UCUI01 and UCOPTIMUS01.	HPE
V0.10	18/09/2017	Syncronized Application Description fields with Architect; added logical architecture diagram and description	HPE
V0.11	21/09/2017	Integration of Tecnalia and Experis contributions	HPE, TECNALIA, Experis
V0.12	26/09/2017	Added: Use Cases UCDO06 and UCDO07, deployment components architecture, Terraform and other implementation technologies	HPE
V0.13	28/09/2017	Added content for section 7 on ADAPT deployment alternatives; integrated ARSYS contribution	HPE
V0.14	29/09/2017	Integration of ARSYS Glossary contribution; updated section 3.3 for new Use Cases; updated application description; integrated Experis update	HPE, ARSYS, Experis
V0.15	02/10/2017	Added Executive Summary; added Introduction and Conclusion	HPE, Experis
V0.16	9/10/2017	Add further TECNALIA contribution, removed partners’ names from titles	HPE, TECNALIA

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
V0.17	16/10/2017	Updated after reviewer's comments : also moved the Application Description definition to the Appendix	HPE
V0.18	18/10/2017	Added interface iObtainRelease resources in section 5.3	TECNALIA
V0.19	19/10/2017	Changed ADAPT architecture figure in section 4	HPE
V0.20	20/10/17	Comments updated during conference call	HPE
V0.21	23/10/17	Removed handled/obsolete comments	HPE
V0.22	24/10/17	Updated App Description: removed unused fields, renamed some fields using camelCase notation, added Used By column, added other fields needed by ADAPT Deployment	HPE
V0.23	25/10/17	Integrated updates from TECNALIA; obsolete comments removed; minor text and formatting changes	HPE, TECNALIA
V0.24	25/10/17	Updated component diagrams for consistency	HPE
V0.25	31/10/17	Integrated ARSYS contribution on section 2.4.2.3; small fixes to text; updated monitoring component diagrams; added missing interfaces	ARSYS, HPE, TECNALIA, EXPERIS
V0.26	10/11/17	Updated ADAPT deployment components	HPE
V0.27	13/11/17	Updated Helpers’ role in ADAPT logical architecture; final editing	HPE
V1.0	13/11/17	Version ready for release	HPE
V1.1	21/11/17	Updated according to TECNALIA review comments	HPE
V2.0	22/11/2017	Ready for submission	TECNALIA

Table of Contents

List of Figures

List of Tables

Terms and abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
CSP	Cloud Service Provider
DevOps	Development and Operations
DoW/DoA	Description of Work/Action
EC	European Commission
GB	Giga Bytes
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KR	Key Result
MCSLA	Multi-Cloud Service Level Agreement
NFR	Non Functional Requirement
CNA	Cloud Native Application
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
UC	Use Case
UI	User Interface
UML	Unified Modeling Language
URI	Unified Resource Identifier
URL	Unified Resource Locator
UUID	Universally Unique IDentifier
VM	Virtual Machine
WP	Work Package

Executive Summary

Multi-cloud applications are applications that can dynamically distribute their components over heterogeneous cloud resources. The main objective of DECIDE project is to provide a novel software framework to design, develop, and dynamically deploy multi-cloud applications. In this framework the ADAPT tool aims to allow continuous deployment and dynamical self-adaptation for multi-cloud applications and to support their re-deployment using the best combination of cloud services to satisfy the required Non Functional Properties (NFP) and Service Level Agreement (SLA).

This deliverable is the first result of DECIDE Work Package 4 (“continuous deployment and operation”) and reports the initial ADAPT architecture and design. The architecture of the overall DECIDE framework can be found in the DECIDE deliverable D2.4.

This document starts with a detailed listing of requirements (sect. 2.1) collaboratively collected from DECIDE Partners at the beginning of the project. The requirements are then analyzed and categorized (sect. 2.3) by separating those applying to the whole DECIDE framework or describing the ADAPT tool at a high level, from the requirements explicitely referring to specific functionalities .

To better understand ADAPT requirements, a section is also provided (sect. 2.4) to explain the main concepts about the ADAPT target: *native cloud applications* possibly developed according to the *micro-services architecture* and deployed on multiple clouds using *containers technology*.

Starting from the requirements, and in the context of the overall DECIDE workflow (sect. 3.1) developed in Work Package 2, WP4 Partners identified and detailed the main functionalities of the ADAPT tool (sect.3.2). To represent those functionalities and their relationships we used the UML use case graphical syntax, followed by a textual description for each identified use case. For better traceability, each use case has a unique identifier and is mapped to its related requirements (sect.3.3). The main functionalities of ADAPT are: deploying the application, monitoring it and adapting this application to any external or internal variability, typically by redeploying it.

Redeploying an application involves stopping and undeploying the current configuration and deploying the new one. Adapting an application by redeploying it on every SLA violation may impact business continuity. Techniques to avoid such an impact have been studied and are reported in the deliverable (sect. 3.4). Some of the reported techniques apply to the application or to the way developers create it (e.g. use a microservices architecture possibly composed of stateless services), and have been therefore proposed as fundamental techniques for the whole DECIDE framework.

The three main functionalities of ADAPT, deployment, monitoring and adaptation, are reflected in the three main components of its architecture: Deployment Orchestrator, Monitoring Manager, and Violations Handler. This document reports the high level ADAPT architecture and details the components and subcomponents that are planned to be developed in the whole project lifecycle (sect. 4), along with their respective interfaces (sect. 5).

Most of the information needed to perform the ADAPT functionalities are provided by an object which is also central to the whole DECIDE workflow: the Application Description. Its definition has evolved since the first data model shown in the DECIDE deliverable D2.1, and it is still evolving in parallel with the design and implementation iterations. This deliverable reports the current definition of all Application Description’s fields and their types (introduced in sect. 5.1 and listed in the Appendix)..

The implementation of the ADAPT components is not starting from scratch. In parallel with the collection of requirements and the definition of the architecture, DECIDE Partners evaluated some new open source technologies for deployment and monitoring, aiming at reusing them for a faster and less effort-intensive development of the ADAPT tool. Some of the evaluated technologies, such as

Terraform and Nagios plus others, which can be considered good candidates as starting points for the implementation, have been briefly described in this deliverable (sect. 6).

Another major topic to be analyzed beyond implementation is how to deploy the ADAPT tool. The deployment decision results from both technical and business considerations: we will have to analyze technical aspects to understand what is feasible and then weight the possible solutions from a business standpoint. This analysis work is just started and will progress in the next year, in collaboration with WP7 for business analysis. This deliverable reports the basic idea, which project Partners currently agree on: implementing ADAPT as a containerized microservices application (sect. 7.1), and deploying it either with one instance per application (sect. 7.2), or as a service (sect. 7.3). The deployment alternatives have not been analyzed yet: this will be one of the major topics for next year's work.

The next WP4 deliverables due by M12 will be D4.4, D4.7 and D4.10, respectively about deployment and adaptation, monitoring, and helpers. These deliverables will report about the initial ADAPT implementation according to the architecture designed in this deliverable. These deliverables will also report about any update to the ADAPT tool design and to the Application Description definition, following from the experience gained with the implementation.

1 Introduction

1.1 About this deliverable

This deliverable corresponds to Task 4.1, *DECIDE ADAPT Architecture and Integration*. It aims at describing DECIDE ADAPT's high-level architecture. The requirements elicited collaboratively amongst partners are the starting point of this document. They are analyzed and refined to obtain the functionalities that ADAPT must provide. These functionalities help break down ADAPT into components, that are described in this document along with their interfaces, which will be implemented in tasks 4.2, 4.3 and 4.4.

1.2 Document structure

The document starts listing the requirements that affect ADAPT (sect. 2), and analyzing whether those requirements apply to the whole DECIDE framework or just to ADAPT itself. The last part of this section is devoted to multi-cloud applications: what they are, what advantages they provide and what are the main issues.

The next section deals with the high-level functionalities of DECIDE (sect. 3). It starts with the description of ADAPT's role within DECIDE and the use cases that affect this Key Result (KR). Then, the requirements are mapped with the use cases, to allow requirements tracking down to functionalities and their implementation.

The deliverable continues with the high-level description of ADAPT's architecture (sect. 4). It identifies the main components and analyzes their behavior and interactions.

The next section is devoted to ADAPT's interfaces (sect. 5). It presents the Application Description, which is the main source of information for ADAPT, and describes the interfaces used for monitoring and those that will be consumed by this KR.

Lastly, the document analyzes candidate technologies that can be used for ADAPT or some of its functionalities (sect. 6) and explores different deployment possibilities for this tool (sect. 7).

2 Initial ADAPT requirements

2.1 Requirements collected from Partners

Requirements for the ADAPT tool have been collected and discussed as a collaborative work among DECIDE WP4 Partners. An online live document [1] has been used to collect requirements and to classify and analyse them.

In Table 1 below all the Accepted requirements have been reported.

Table 1. DECIDE ADAPT Accepted requirements

Req. ID	Description	Reference	Comment
WP4-REQ1	DECIDE ADAPT will support the self-adaptation and dynamic re-deployment of (parts of) multi-cloud applications when certain conditions are not met.	DoA sect. 1, p. 7	High level description of DECIDE ADAPT
WP4-REQ2	[Conditions not met] include for instance, that the defined composite multi-cloud application SLA is not being fulfilled, the application is not performing as established or the cloud service providers (CSPs) are violating the contracted SLAs.	DoA sect. 1, p. 7	List of possible adaptation triggers
WP4-REQ3	DECIDE ADAPT will pro-actively adjust the running configuration of the application	DoA sect. 1.1, p. 9; DoA sect. 1.3.2.1, p. 13	Adjusting the application configuration is one possible adaptation action, not the only one. Doing that proactively means that ADAPT will trigger the workflow for application reconfiguration
WP4-REQ4	{Adjustment will be} based on measurements that are derived from the dynamic monitoring activities of both the application and the non-functional properties of the CSPs and cloud offerings where the application is deployed and making use of.	DoA sect. 1.1, p. 9; DoA sect. 1.3.2.1, p. 13	List of possible adaptation triggers leading to application configuration adaptation [see also REQ2]
WP4-REQ5	KPI 5.1: Degree of correctness of the re-deployment configuration compared with NFR and other requirements (at least 90% of NFR).	DoA sect. 1.1, p. 9	KPI shared with OPTIMUS, since it generates the deployment configuration
WP4-REQ6	KPI 5.2: 70% of the functionalities will be validated overall in the DECIDE use cases	DoA sect. 1.1, p. 9	[collaboration: WP6]

Req. ID	Description	Reference	Comment
WP4-REQ7	DECIDE will address the enhancement in the development productivity and time-to-market of 1) applications which require high rates of performance and reliability (i.e.: network management and online gaming) and 2) applications in which the legal compliance of the CSP where the application is deployed as well as the legal compliance of the used cloud services and resources is critical due to the type of data managed (i.e.: sensitive data in eHealth).	DoA sect. 1.1, p. 9	This is a requirement more at the project level
WP4-REQ8	DECIDE ADAPT [KR5], allows operators to focus on the development of innovative features of the multi-cloud application, as it provides the mechanisms, algorithms and tools to let the application self-adapt itself and deploy (semi-) automatically the application or parts of the application	DoA sect. 1.2, p. 10	To be highlighted: "tools to let the application self-adapt itself", whereas REQ3 says "will pro-actively adjust"
WP4-REQ9	DECIDE ADAPT [KR5], the application self-adaptation tool will support automated dynamic deployment of service components as well as the runtime monitoring of functional and non-functional service properties.	DoA sect. 1.2, p. 10	High level description of DECIDE ADAPT, see REQ1
WP4-REQ10	DECIDE ADAPT [KR5] will be able to change the configuration and topology of services at operational time based on continuous monitoring of both the conditions of the application and the CSPs where the application is deployed on.	DoA sect. 1.2, p. 11	see also REQ8
WP4-REQ11	A multi-cloud application is a distributed application over heterogeneous cloud resources whose components are deployed on different CSPs and still, they all work in an integrated way and transparently for the end-user.	DoA sect. 1.3.2.1, p. 12	Definition of multi-cloud application
WP4-REQ12	DECIDE OPTIMUS [..] will provide [..] automation of the provisioning resources and deployment scripts for multi-cloud native applications	DoA sect. 1.3.2.1, p. 13	After further analysis, the Partners agreed that ADAPT will generate the deployment scripts and that resource provisioning, will be performed through ACSmI; see section 4.1

Req. ID	Description	Reference	Comment
WP4-REQ15	Each deployment configuration will be stored in the multi-cloud native application controller, maintaining the current deployment configuration situation as well as the historic of the previous deployment configuration used, so that they can be checked in the re-deployment phase	DoA sect. 1.3.2.1, p. 14	Application controller (output of T3.4) will keep a history of deployment configurations [collaboration: WP3]
WP4-REQ17	The following CSPs and corresponding technologies will be integrated in DECIDE: ARSYS (VMWare), AIMES (modified OpenStack), TECNALIA (OpenStack and OpenShift), Amazon as well as other European Cloud Service providers.	DoA sect. 1.3.2.1, p. 14	Lists which are the CSPs to be supported [multi WP requirement]
WP4-REQ18	During the application operation phase, the DECIDE self-adaptation application provisioning tool (ADAPT [KR5]) will continuously monitor and assess the fulfillment of the established NFR and MCSLA	DoA sect. 1.3.2.1, p. 15	Continuous monitoring of NFR and MCSLA
WP4-REQ19	If a violation of any of the [former: NFR and MCSLA] metrics occurs, the self-adaptation tool through the ACSmI will assess the operation of the (combination of) cloud services selected and discard those that are affecting the MCSLA	DoA sect. 1.3.2.1, p. 15	In case of a violation, ACSmI will indicate which the responsible cloud services are. [collaboration: WP5]
WP4-REQ20	The level of technological risk (Low or High) must be defined for the application	DoA sect. 1.3.2.1, p. 15	We expect the level of technological risk to be defined in the Application Description [Requirement for WP3]
WP4-REQ21	If the application configuration has been established as of low technological risk, the multi-cloud application will be self-adaptive and it will be redeployed automatically, following a new deployment configuration.	DoA sect. 1.3.2.1, p. 15	No need to get operator permission for redeployment in case of low technological risk application
WP4-REQ22	In case the application has been identified as high technological risk, once it has identified the aspects that are affecting the malfunctioning of the application, it will alert the operator and using the OPTIMUS tool it will look for new (combination of) cloud services to set up a new deployment schema	DoA sect. 1.3.2.1, p. 15	ADAPT must be able to alert the operator through some UI; ADAPT will get new deployment schema from OPTIMUS [collaboration: WP3]
WP4-REQ23	[DECIDE ADAPT] will support the selection of the new deployment scripts (based on the architectural patterns for deployment), and thus semi-automatically re-deploy it [the application].	DoA sect. 1.3.2.1, p. 15	This is still the case of high technological risk application, where the operator should select and confirm redeployment

Req. ID	Description	Reference	Comment
WP4-REQ25	DECIDE will also support multiple cloud layers	DoA sect. 1.3.2.1, p. 15	Both IaaS and PaaS (specifically OpenShift and containerized apps) will be supported
WP4-REQ27	DECIDE ADAPT [KR5] will provide the operator a report with the NFPs that are not being fulfilled and an input file to be able to simulate a new deployment topology through DECIDE OPTIMUS	DoA sect. 2.1.1, p.29	ADAPT will provide a report about violations to the operator and to OPTIMUS [two different formats]
WP4-REQ28	DECIDE ADAPT is a framework providing a deployment orchestrator (workflow manager for orchestrating the actions required for deploying the multi-cloud application)	DoA part A: Work package descriptions Objectives/T4.2	Deployment Orchestrator is a main ADAPT component which will orchestrate multi-cloud application deployment
WP4-REQ29	DECIDE ADAPT is a framework providing a monitoring engine to monitor if the working conditions meet the NFR and to trigger alarms (high technological risks) or actions (low technological risks)	DoA part A: Work package descriptions Objectives/T4.3	Monitoring Engine is a main ADAPT component which will monitor multi-cloud applications and will activate remediation actions in case of violation
WP4-REQ30	DECIDE ADAPT is a framework providing an adaptation workflow manager (establishes the workflow and performs the adaptations actions)	DoA part A: Work package descriptions Objectives/T4.2	
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	DoA part A: Work package descriptions Objectives/ T4.3	Helpers are reported in deliverable D4.10
WP4-REQ32	Helpers are specific for application and /or for architectural pattern	DoA part A: Work package descriptions Objectives	Helpers are reported in deliverable D4.10
WP4-REQ33	Helpers will be used/developed for the use cases	DoA part A: Work package descriptions	Helpers are reported in deliverable D4.10

Req. ID	Description	Reference	Comment
		Objectives/ T4.3	
WP4-REQ34	The actual deployment, monitoring and adaptation of an application is obtained "feeding" DECIDE ADAPT with a description of the application containing the related helpers	DoA part A: Work package descriptions Objectives	The Application Description document is the basis of all ADAPT functionalities
WP4-REQ35	ADAPT will support applications based on composition of stateless (possibly micro) services	Tech assessment	For technical feasibility of multi-cloud deployment. We focus on new or refactored applications [Affects use cases]. Actually, also stateful microservices should be supported
WP4-REQ36	ADAPT will support modular applications where each composition unit is a containerized service	Tech assessment	For technical feasibility of multi-cloud deployment ADAPT will not target the containerization of apps, its input is expected to contain a set of containerized modules each one providing one (or a category of) service(s)
WP4-REQ37	DECIDE ADAPT [KR5], relying on the formal description of the multi-cloud application, orchestrates the deployment, monitors the working conditions and performs adaptations actions	DoA part A: Work package descriptions Objectives	ADAPT relies on the formal description of the application [collaboration: WP2/WP3]
WP4-REQ41	DECIDE ADAPT [KR5] can also be easily extended with the inclusion of more NFPs that need to be measured, as well to different types of applications such as those following the mobile cloud paradigm or IoT	DoA sect. 2.1.2, p.30	This requirement is cross WP, requiring definition at architectural level in WP2
WP4-REQ42	Applications are natively multi-cloud; as such, they can take advantage of the different cloud service offers - even in the activation of a single application - because they can be actually distributed on different cloud providers	DoA sect. 2.1.4, p.32	ADAPT targets native Multi-cloud applications, which can be distributed over different CSPs; see also WP4-REQ11 and WP4-REQ35

Req. ID	Description	Reference	Comment
WP4-REQ43	If the application configurations are classified with high technological risk, [re-adaptation involves] simulating again the deployment with DECIDE OPTIMUS but in a more driven and accurate way, entering now as input, the identified problems so that the new configuration can provide a solution to that problem.	DoA sect. 2.1.5, p.33	ADAPT will provide to OPTIMUS information on the identified violation; see also WP4-REQ27
WP4-REQ44	Users will perceive relevant improvements in the business continuity since as soon as there is a problem (i.e. lack of resource due to a peak of requests) the software is automatically re-adapted and re-deployed	DoA sect. 2.1.5, p.33	Benefit to applications' users; expectation is to improve business continuity, therefore during adaptation applications' downtime should be minimized
WP4-REQ45	[DECIDE ADAPT is] a tool that allows the (semi-)automatic adaptation of the application and re-deployment in another multi-cloud configuration when certain conditions are not met. These conditions are on one hand the violations of the application's own multi-cloud SLA (MCSLA) and on the other hand, the non-fulfilment of the NFP of the CSPs where the application is deployed as well as the non-fulfilment of the NFP of the services provided by the ACSmI that the application is using. These conditions will trigger an alert and will cause the OPTIMUS tool to be launched again in order to search for another deployment configuration. Depending on the technological complexity requirement, and the initially prioritized requirements by the user, the application will be re-adapted automatically or an alert to the operator will be launched along with a diagnosis of what malfunctioned so that a new optimal configuration can be found.	DoA sect. 2.2.5.5, p.41	High-level description of DECIDE ADAPT. Clear list of adaptation triggering conditions (violation of application's MCSLA, non-fulfillment of NFP of CSPs and of services provided by ACSmI to the application). Prioritization of SLAs is a REQ for WP3

2.2 DevOps Requirements

This section reports some requirements deriving from DevOps principles which have been collected by WP2. The **Table *2*** lists the DevOps requirements from DECIDE Deliverable D2.1 [2] that apply to ADAPT.

Table 2. DevOps Requirements applicable to ADAPT

Req. ID	Description	Source
DEVOPS-REQF3	DECIDE framework must be able to monitor the deployed micro-services	DevOps Principles #7
DEVOPS-REQF6	DECIDE framework must support the continuous deployment of the developed apps	DevOps Principles #5
DEVOPS-REQF9	DECIDE framework must be able to track the issues that affect the deployed services and use registries to store this information	DevOps Principles #7
DEVOPS-REQF15	DECIDE must support the monitoring of the multi-cloud application SLA and the SLAs of the underlying cloud resources	Extended DevOps #3
DEVOPS-REQF16	DECIDE must support the semi-automatic adaptation and redeployment of the application into new cloud services when needed based on the assessment of the continuous monitoring	Extended DevOps #3
DEVOPS-REQF17	DECIDE must support the monitoring of the SLAs of the underlying cloud resources	Extended DevOps #3

2.3 Requirement Analysis

The collected DECIDE ADAPT requirements have been collaboratively analysed during several conference calls by WP4 Partners and also autonomously by Partners in charge of specific components.

As a first classification, the higher level requirements have been identified. Some of them are general DECIDE requirements, which can be applied to the whole DECIDE framework. Other are useful high level descriptions of the DECIDE ADAPT component. This section reports, in the following **Table *3***, the requirements from the previous sections which have been identified as high level and their classification according to the two categories "General ADAPT requirement" and "ADAPT description". The requirements' description is the same as that reported in the previous sections and has been repeated here for reading convenience.

A further detailed analysis has been performed to map lower level requirements with the identified use cases. The result of this work is reported in section 3.3.

Table 3. High level general requirements

Req. ID	Description	General DECIDE req.	ADAPT description
WP4-REQ1	DECIDE ADAPT will support the self-adaptation and dynamic re-deployment of (parts of) multi-cloud applications when certain conditions are not met.		X
WP4-REQ2	[Conditions not met] include for instance, that the defined composite multi-cloud application SLA is not being fulfilled, the application is not performing as established or the		X

Req. ID	Description	General DECIDE req.	ADAPT description
	cloud service providers (CSPs) are violating the contracted SLAs.		
WP4-REQ7	DECIDE will address the enhancement in the development productivity and time-to-market of 1) applications which require high rates of performance and reliability (i.e.: network management and online gaming) and 2) applications in which the legal compliance of the CSP where the application is deployed as well as the legal compliance of the used cloud services and resources is critical due to the type of data managed (i.e.: sensitive data in eHealth).	X	
WP4-REQ8	DECIDE ADAPT [KR5], allows operators to focus on the development of innovative features of the multi-cloud application, as it provides the mechanisms, algorithms and tools to let the application self-adapt itself and deploy (semi-) automatically the application or parts of the application		X
WP4-REQ9	DECIDE ADAPT [KR5], the application self-adaptation tool will support automated dynamic deployment of service components as well as the runtime monitoring of functional and non-functional service properties.		X
WP4-REQ10	DECIDE ADAPT [KR5] will be able to change the configuration and topology of services at operational time based on continuous monitoring of both the conditions of the application and the CSPs where the application is deployed on.		X
WP4-REQ11	A multi-cloud application is a distributed application over heterogeneous cloud resources whose components are deployed on different CSPs and still, they all work in an integrated way and transparently for the end-user.	X	
WP4-REQ12	DECIDE OPTIMUS [..] will provide [..] automation of the provisioning resources and deployment scripts for multi-cloud native applications	X	
WP4-REQ17	The following CSPs and corresponding technologies will be integrated in DECIDE: ARSYS (VMWare), AIMES (modified OpenStack), TECNALIA (OpenStack and OpenShift), Amazon as well as other European Cloud Service providers.	X	
WP4-REQ25	DECIDE will also support multiple cloud layers	X	
WP4-REQ35	ADAPT will support applications based on composition of stateless (possibly micro) services		X
WP4-REQ36	ADAPT will support composable applications where each composition unit is a containerized service		X
WP4-REQ37	DECIDE ADAPT [KR5], relying on the formal description of the multi-cloud application, orchestrates the deployment, monitors the working conditions and performs adaptations actions		X

Req. ID	Description	General DECIDE req.	ADAPT description
WP4-REQ41	DECIDE ADAPT [KR5] can also be easily extended with the inclusion of more NFPs that need to be measured, as well to different types of applications such as those following the mobile cloud paradigm or IoT		X
WP4-REQ42	Applications are natively multi-cloud; as such, they can take advantage of the different cloud service offers - even in the activation of a single application - because they can be actually distributed on different cloud providers	X	
WP4-REQ44	Users will perceive relevant improvements in the business continuity since as soon as there is a problem (i.e. lack of resource due to a peak of requests) the software is automatically re-adapted and re-deployed		X
WP4-REQ45	[DECIDE ADAPT is] a tool that allows the (semi-) automatic adaptation of the application and re-deployment in another multi-cloud configuration when certain conditions are not met. These conditions are on one hand the violations of the application's own multi-cloud SLA (MCSLA) and on the other hand, the non-fulfilment of the NFP of the CSPs where the application is deployed as well as the non-fulfilment of the NFP of the services provided by the ACSmI that the application is using. These conditions will trigger an alert and will cause the OPTIMUS tool to be launched again in order to search for another deployment configuration. Depending on the technological complexity requirement, and the initially prioritized requirements by the user, the application will be re-adapted automatically or an alert to the operator will be launched along with a diagnosis of what malfunctioned so that a new optimal configuration can be found.		X

2.4 From Cloud to Multi-cloud applications

To better understand ADAPT requirements, a brief explanation is needed about the context in which those requirements apply. The object to which the ADAPT tool is applied is a *Cloud Native Application* (CNA) possibly developed according to the *micro-services architecture* and deployed on *multiple Clouds* using *Containers technology*. This section briefly explains the concepts indicated above and analyzes some issues arising when deploying applications in a multi-cloud environment.

2.4.1 What is a Cloud Native Application

A Cloud Native Application is software precisely developed for virtualisation environments or the so-called cloud computing platforms. Cloud native applications are designed in such a manner that they get the most out of services provided by virtualised infrastructures.

In general, cloud native applications have these characteristics:

- **Large number of parallel processing units.** It consists of a large number of parallel processing units, in addition to being able to run different parts of the application at the same time and store data in different locations simultaneously.
- **Full use of cloud resources**. These applications use Application Programming Interfaces (API) and other methods to simplify administration tasks and efficiently use most of the available resources (processor, memory and storage).
- **Multi-cloud paradigm**. Migration and deployment in multiple Cloud Service Providers (CSP) in a simple, transparent and with the least possible impact on end users.

In other words, the cloud-native approach is based on developing and running applications that exploit all the advantages of the cloud computing service model. The importance lies in how applications are created and deployed, and not so much where they are running. In this way, developers can make use of almost unlimited computing resources, according to their needs, along with cutting-edge data and application services. As a result, companies developing and deploying cloud native applications are able to bring new products to markets faster and meet customer requirements sooner.

Organizations eager to start developing cloud native software need platforms that automate and integrate concepts like DevOps, continuous delivery, microservices and containers.

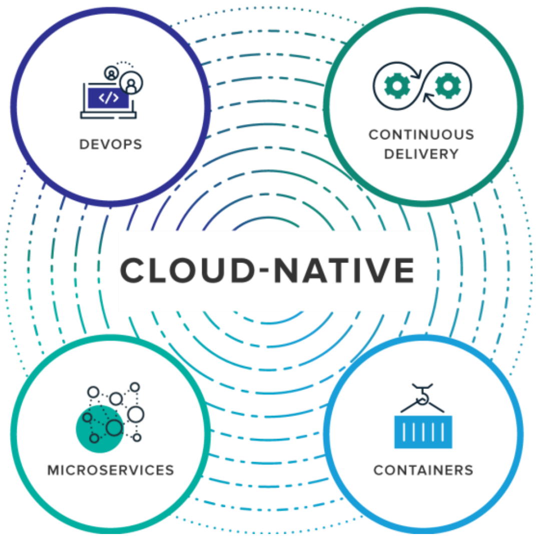


Figure 1. Fundamentals of a Native Cloud Application [3]

2.4.2 Microservices Architecture

As a summary, a microservice architecture could be defined as an application development technique consisting of several independent services that execute a single process and communicate with each other through simple and transparent protocols to fulfil a common purpose. In this way, each microservice can be deployed and restarted in isolation, offering the possibility to launch updates regularly, and maintaining the SLA (Service Level Agreement) agreed with end customers.

Needless to say that communication between services is tied to the application's requirements, but a large majority of developers seem to opt for HTTP/REST with JSON or Protobuf [4]. It is worth mentioning that DevOps professionals usually choose Representational State Transfer (REST) as the most appropriate communication protocol because it is lighter than others [5].

In order to better understand the microservice architecture, a brief comparison with its opposite, the traditional monolithic architecture, is presented in the table below.

Table 4. Microservices vs Traditional Architecture

Microservices Architecture	Traditional Architecture
Single focus. They are targeted at a specific problem, and contain everything needed including data.	**Wide focus**. They attempt to solve many problems at once in a tightly closed software package.
Simple updates. Only the required services are updated, built and deployed.	**Tedious updates**. A modification made to a small section of an application might require building and deploying an entirely new version.
Loosely coupled. This architecture demands services to be as independent as possible, avoiding hard-coded references to others.	**Tightly coupled**. These systems are often a tangled set of interrelated components that require meticulous manual procedures to be deployed successfully.
Scalable by nature. The microservice architecture allows modifying the resources needed for each service on demand.	**Almost no scalable**. In the case of having to scale specific functions, it may result in having to scale the entire application instead of just the desired components.
Delivered continuously. Microservices are ideal for applications with constantly updates. They usually require delivering value to market as quickly as possible; hence microservices are regularly deployed in production through automated tasks.	**Scheduled delivery**. Applications are developed and updates are made according to schedule, often at much slower rates than the microservice architecture.
Independent teams owning the service life cycle. Microservices are built, deployed, and run by independent teams.	**Many teams owning the services life cycle**. Developers are responsible for building the first iteration of software, and then it is handed over to Operations to be maintained accordingly.
Design patterns and distributed systems at scale. Microservices architectures rely on a set of tools for service discovery, messaging, network routing, failure detection, logging, storage and identity.	**Processes come first**. Isolated tools and processes, focused on development, Quality Assurance (QA), and release to production, produce monolithic applications.

2.4.2.1 *Stateful vs Stateless Applications*

REST is one of the most widely used communication protocols in microservices architecture. This protocol is based on the concept of statelessness. A stateless object maintains neither information relative to the user, nor context related to the application between calls to the same object. Each call is independent of previous calls and is not part of an ongoing call. This does not mean that data or context are not used, but this information is not preserved between different calls. If it were necessary to save state information, it would be stored in the client. In contrast, stateful applications use the data stored in previous client sessions each time the client makes a new request. The following diagram illustrates the essential differences between stateful and stateless applications.

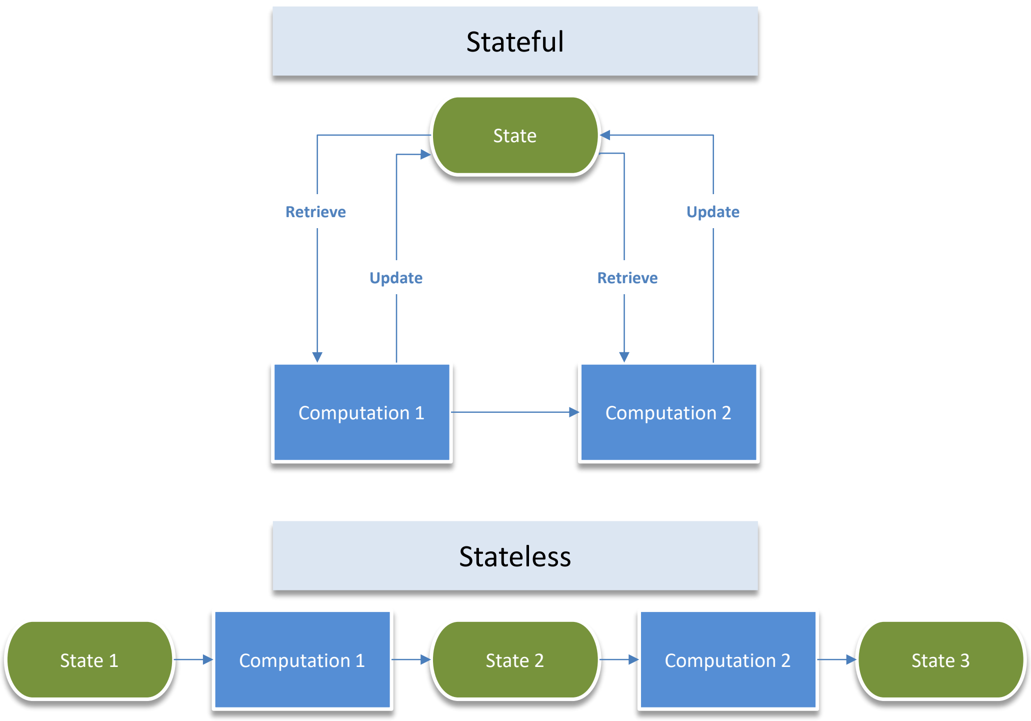


Figure 2. Stateful vs Stateless Software Design [6]

According to the figure above, a stateless application splits the code from the state so that the state becomes an immutable object, meaning that it changes one state into another. This is what computer scientists call referential transparency [7]: the ability to replace a computation with the value it returns without changing the behaviour of the program.

So why the growing interest in stateless software design? One reason is that stateless applications can either be easily deployed again in the event of failure, or be scaled to balance peak loads. Another reason is that stateless applications can effortlessly connect to other applications through APIs.

In addition to this, the so-called functional programming, which uses microservices and containers technology, is also promoting the development of stateless applications. This programming approach is based on the development of small parts of immutable code. In this way, each function executes its task without depending on the rest of the functions in the program and without taking into account previous executions. This flexibility gives developers the ability to join functions in different manners without the risk of dependencies among them.

2.4.2.2 Advantages of Multi-cloud Applications

After outlining the concepts of cloud native applications, microservice architecture and stateless applications, this subsection presents the advantages of embracing the paradigm of multi-cloud applications.

- **Minimise the risk of widespread data loss and keep at a minimum downtime**. Cloud platforms are very complex computing environments that inherently have multiple points of failure: hardware, software or infrastructure. Due to the use of two or more cloud services it is possible to ensure SLAs agreed with customers.
- **Improve the overall performance of organisations**. Multi-cloud strategies promote the use of open source and standardised technologies, avoiding vendor lock-in with proprietary solutions. The diversity of multi-cloud ecosystems facilitates compliance with the requirements of a wider range of partners and customers.

- **Impact directly on customer satisfaction**. The speed with which a particular website loads its content is strongly linked to the satisfaction of end users. Websites with faster page loads usually have more frequent and longer visits, so search engine rankings are also affected. A multi-cloud deployment can help significantly reduce the load time of an organisation's web applications.
- **Optimise the traffic of customers and partners**. In addition to providing the redundancy required to efficiently reduce fault tolerance, in a multi-cloud environment it is also possible to route traffic based on the type of client, the content to be distributed, and the nature of the application. For instance, some CSPs offer servers and networks best suited to handle large numbers of requests requiring small data transfers, while other CSPs have a portfolio which performs best for smaller numbers of requests with larger data transfers.

2.4.2.3 Disadvantages of Multi-cloud Applications

In the same way that some advantages of multi-cloud applications have just been presented, below are the challenges of this cutting-edge paradigm:

- **Limited interoperability**. There is currently no complete interoperability between different cloud providers. This forces developers to use workarounds to successfully deploy applications on different platforms and clouds.
- **Greater complexity**. This is the biggest challenge of multi-cloud applications. Developers and administrators have to deal with different interfaces, technologies and services. There are currently neither standardised terminologies nor methodologies across cloud providers.
- **More workload**. Implementing a multi-cloud environment brings a greater workload for developers and DevOps teams. It first takes longer to select the right services to use from each provider. They also have to learn how to integrate their applications with the different infrastructures and APIs available. In fact, it is sometimes necessary to maintain specific source code versions for each provider. Once applications are in production, it is more complex for DevOps engineers to manage and maintain their performance across multiple clouds.
- **Difficulty to estimate costs**. The flow of application data into or out of the infrastructure of each cloud provider generates costs which are difficult to calculate accurately. It is required to conduct an in-depth review of the pricing structure of each service used to come up with an approximation of the overall cost.

2.4.3 Containers Technology

At present, there is a real buzz about container technology and the multi-cloud paradigm. But what is a container? In simple terms, containers wrap software up within a complete file system that contains everything it needs to run: code, runtime, system tools and system libraries. This is really significant because this guarantees that it will always run the same, regardless of the environment it is running within.

Containers use namespace isolation, resource control, and process-isolation technologies to restrict the files, network ports, and running processes that each container can access, so that applications running in containers cannot interact or see other applications running in the host OS or in other containers.

Sometimes Docker is mistakenly used to refer to containers. Docker is the name of a company which has made containerization easy. Docker makes applications deployable anywhere in spite of the underlying infrastructure. Nevertheless, container technology is actually provided by the Linux Kernel. It is an operation call *chroot* or change root that changes the apparent root directory for the current process and its children. Consequently, a process running in such an environment cannot access files outside the designated directory tree.

A container is kind of a "super chroot", with Kernel namespaces[1] and *cgroups* or control groups[2]. The latter one provides resource limitation (CPU, memory, disk I/O) in order to prevent a container from running always and forever by uniquely hogging all available hardware resources. As for the namespaces, there are several of them, but the most important are PID and NET which provide process isolation and network isolation, respectively. This gives containers isolation from all other processes. Containers assume they have their own dedicated operating system.

So, are containers like Virtual Machines? The following table summarises the essential differences between Virtual Machines and Containers.

Table 5. Differences between VMs and Containers

Virtual Machines	Containers
Run an entire OS. A VM has to run a complete Operating System.	**Run an application**. However, a container just run an application.
Heavyweight. A VM is very heavyweight in terms of file size. It might be 1 or 2 GB inside.	**Lightweight**. Containers can be lightweight, even down to just a couple hundred of megabytes or even less than that.
Slow provisioning. Because of file size, VMs are very slow provisioning, normally around a minute.	**Real-time provisioning**. Containers can spin out under half a second, even in real-time like a few hundred milliseconds.
Hardware-level virtualisation. This is the most important difference. The VMs are really concern about virtualising the hardware.	**Operating System virtualisation**. Containers are concern about virtualising the Operating System.

In brief, the following comparisons can be used. VMs are like houses, they are fully self-contained. They have their own heating and plumbing. Whether or not you want your own kitchen, that is what you get. Containers are like apartment buildings. They share their resources, the plumbing and the heating. The following diagram can help visualise both technologies.

[1] Linux Kernel feature that isolates and virtualise system resources of a collection of processes.

[2] Linux Kernel feature that allows to allocate resources among user-defined groups of processes running on a system.

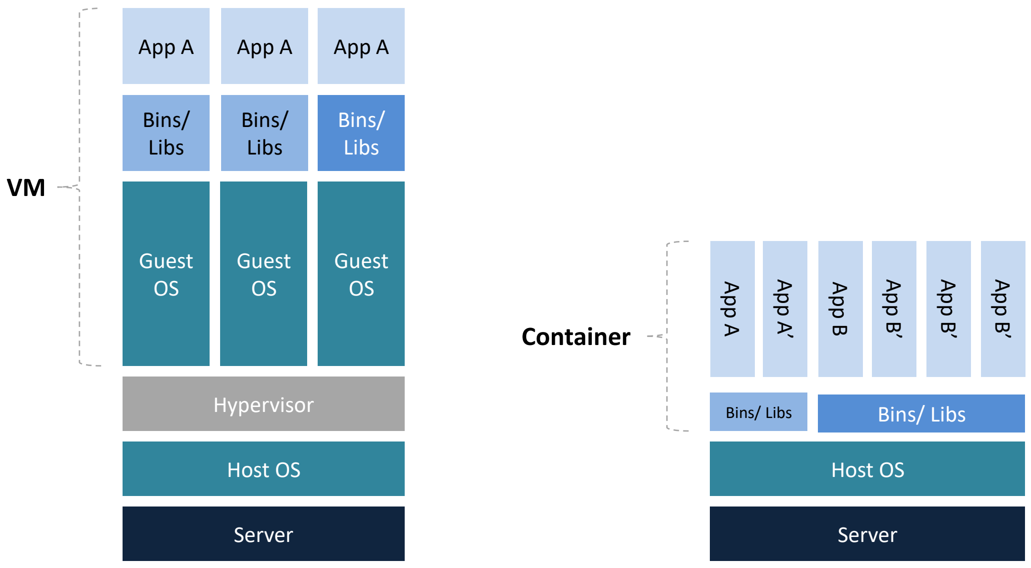


Figure 3. Traditional VM vs Containers Technology [8]

2.4.4 Deployment Issues

After providing an overview of cloud native applications, microservice architecture, and container technology, this last subsection outlines recommendations for the successful deployment of multi-cloud applications.

The first question to address is how many microservices should be accommodated per container. The answer from developers is clear: only one microservice per container. This allows containers to be tailored to that specific service. In this way, the dependencies of that service will not conflict with those of other services as they would be running within their own containers. Examples of this are specific versions of libraries, or even the specific flavour of an operating system that works best for a particular microservice.

The next question is how many containers per virtual machine. Deploying multiple containers on a single VM is expected. As mentioned earlier, containers do not have the overhead of running a whole OS, which means that it is more RAM-efficient to run multiple containers on the same server. However, to run many containers, DevOps teams tend to use an orchestration tool to network them together and manage their life cycle. At the moment, the most popular tools are:

- **Docker Swarm** [9]. More and more developers are using Docker to deploy applications, although administrators need to spend time managing even more containers, particularly those used for large-scale web applications, especially for high availability or access to large computing power. Orchestrating those applications requires a multi-host approach, so Docker Inc. has recently introduced this tool to manage the distribution and orchestration aspect of those applications on multiple machines.
- **Kubernetes** [10]. Also called K8s, it is an open-source system for automating deployment, scaling and handling applications hosted in containers. It was originally designed by Google and donated to the Cloud Native Computing Foundation, part of the Linux Foundation.
- **OpenShift** [11]. Red Hat OpenShift is a complete container application platform that natively integrates technologies such as Docker and Kubernetes, and combines them with a business foundation in Red Hat Enterprise Linux.

Finally, how could we find and use containers deployed in multiple clouds? If an organisation opts to do things 'The Docker Way' [12], then as long as the virtual machines can expose the Docker Machine daemon, it is possible to use VMs in different clouds as part of the same Docker Swarm. If a team favours Kubernetes, or a similar, enterprise container-orchestration system, it is possible to run multiple Kubernetes clusters in different Data Centres or within different clouds and 'federate' [13] them, so that containers can be all controlled from a single API as if they were in a single cloud. As a reference, section 3.4 Improving Business Continuity introduces how the DECIDE framework could alleviate issues of multi-cloud environments.

3 High Level ADAPT Functionalities

This section reports about the high level functionalities identified so far for the DECIDE ADAPT tool.

3.1 ADAPT in the overall DECIDE flow

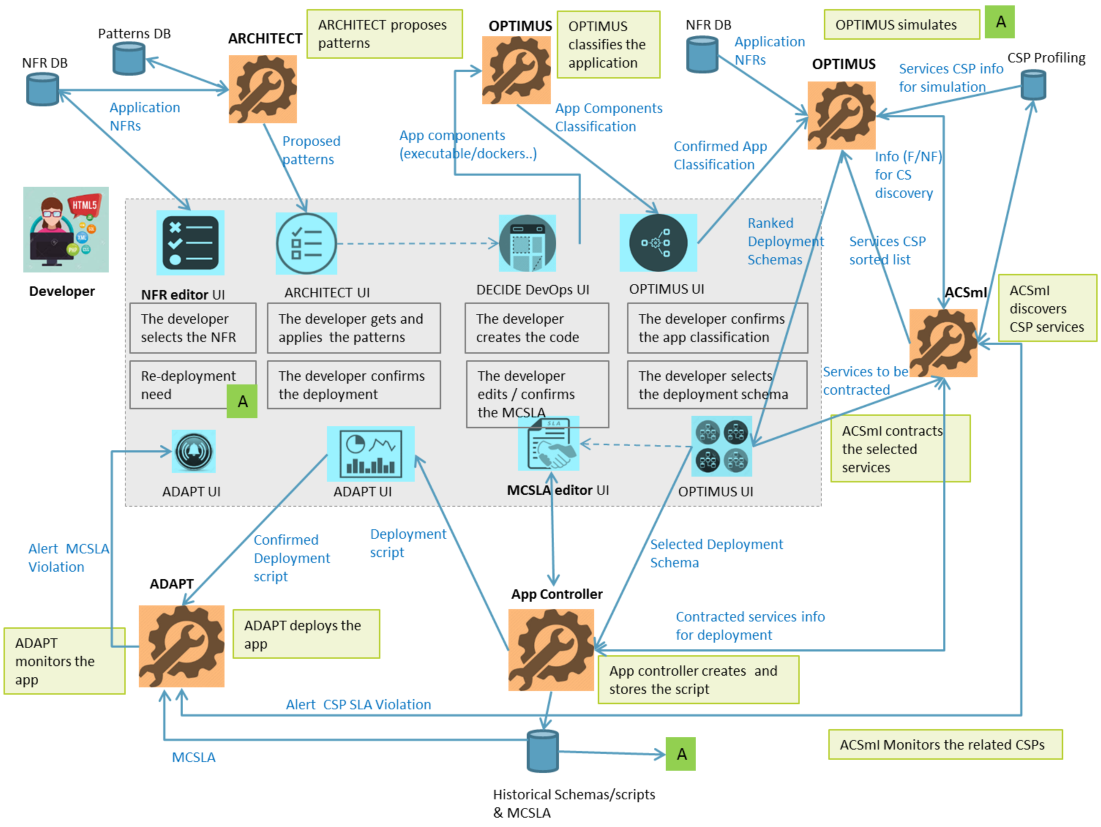

Patterns DB
ARCHITECT
ARCHITECT proposes patterns
OPTIMUS
OPTIMUS classifies the application
NFR DB
Application NFRs
OPTIMUS simulates
A
NFR DB
Application NFRs
Proposed patterns
App components (executable/dockers..)
App Components Classification
OPTIMUS
Services CSP info for simulation
CSP Profiling
Confirmed App Classification
Info (F/NF) for CS discovery
Ranked Deployment Schemas
Services CSP sorted list
Developer
NFR editor UI
ARCHITECT UI
DECIDE DevOps UI
OPTIMUS UI
ACSmI
ACSmI discovers CSP services
The developer selects the NFR
The developer gets and applies the patterns
The developer creates the code
The developer confirms the app classification
Re-deployment need
A
The developer confirms the deployment
The developer edits / confirms the MCSLA
The developer selects the deployment schema
Services to be contracted
ACSmI contracts the selected services
ADAPT UI
ADAPT UI
MCSLA editor UI
OPTIMUS UI
Alert MCSLA Violation
Confirmed Deployment script
Deployment script
Selected Deployment Schema
ADAPT
App Controller
ADAPT monitors the app
ADAPT deploys the app
Contracted services info for deployment
App controller creates and stores the script
Alert CSP SLA Violation
MCSLA
A
ACSmI Monitors the related CSPs
Historical Schemas/scripts & MCSLA

Figure 4. DECIDE workflow

In the previous picture the overall DECIDE workflow at M6 is shown (more detail can be found in [2]). This workflow represents the support that DECIDE tools provide to the complete software development and operation lifecycle for multi-cloud aware applications. This workflow is continously evolving, as the specification of the tools are still in the initial stages and they may change during the project.

DECIDE ADAPT is a framework that provides generic functions for :

- deploying multi-cloud applications to diverse cloud platforms,
- monitoring the deployed applications to verify that the non-functional requirments are being fullfilled
- adapting the deployment in order to restore the required working conditions in case the NFR are violated

DECIDE ADAPT supports the operation phase of multi-cloud aware applications by providing means for automatic distributed deployment, run-time monitoring of the current deployment with repect to selected non-funtional requirements and SLOs and re-deployment adaptation when needed.

DECIDE ADAPT has several interfaces, both with the user (operator) of the multi-cloud application and with external components (other DECIDE tools) as explained next.

DECIDE ADAPT user interface will provide the operator of the application with the functionality of confirming the deployment configuration automatically provided by ADAPT and with a graphical interface where to check the actual compliance of the NFRs and SLOs of the multi-cloud application. This ADAPT UI will provide the required alerts if the required working conditions are not met.

With respect to the interfaces with other DECIDE components, DECIDE ADAPT will interact with the following tools:

- DECIDE OPTIMUS: When any of the NFRs or SLOs are not met, DECIDE ADAPT will request OPTIMUS to launch a new simulation so that the re-adaptation of the deployment takes place.
- DECIDE Application Controller: In order to perform the actual deployment the different components of the multi-cloud application, DECIDE ADAPT needs the corresponding resources, contracted and properly detailed so that the software components can be deployed. The Application Controller will request the deployment to ADAPT when all these parameters are updated in the Application Description.
- DECIDE ACSmI: DECIDE ADAPT monitors the fullfillment of the NFR and the SLOs, at application level. That compliance, depends on the underlying resources where the components of the multi-cloud application are deployed. For this reason, DECIDE ADAPT will request from the ACSmI the CSPs metrics in order to assess the compliance or not of the NFR and SLOs at application level. ADAPT will also obtain and release cloud resources through ACSmI.

The Application Description is the main information repository for the different tools in DECIDE toolchain, including DECIDE ADAPT. DECIDE ADAPT will use the Application Description to retrieve all the necessary information for performing its functionalities. More information about Application Description is included in section 5.1 of the current document.

ADAPT uses the information contained in the Application Description to deploy the multi-cloud application in the different CSPs. ADAPT performs the actual deployment and after executing any optional test, if provided by the developer, switches the application online. ADAPT gets the information about what/when/where and how to monitor from the Application Description. ADAPT continuously monitors the MCSLA and if a violation occurs (the violation can come from the violation of the SLA of any of the cloud services contracted) informs the developer through the ADAPT UI, sends an alert and launches the automatic re-deployment process if applicable (depending on the level of technology risk). In any case, ADAPT sends OPTIMUS a request for a new deployment schema and a report about the violation, so that OPTIMUS can include this information in the simulation. OPTIMUS performs the simulation (the application classification and the NFRs remain the same). When the new deployment configuration is created, ADAPT is invoked again.

3.2 ADAPT Use Cases

High level functionalities in DECIDE ADAPT are represented in the UML Use Case Diagram shown in ***Figure 5*** below, grouped by component. The functionalities are briefly explained in the following subsections. The mapping between the functionalities identified by the use cases and the collected requirements is reported in section 3.3.

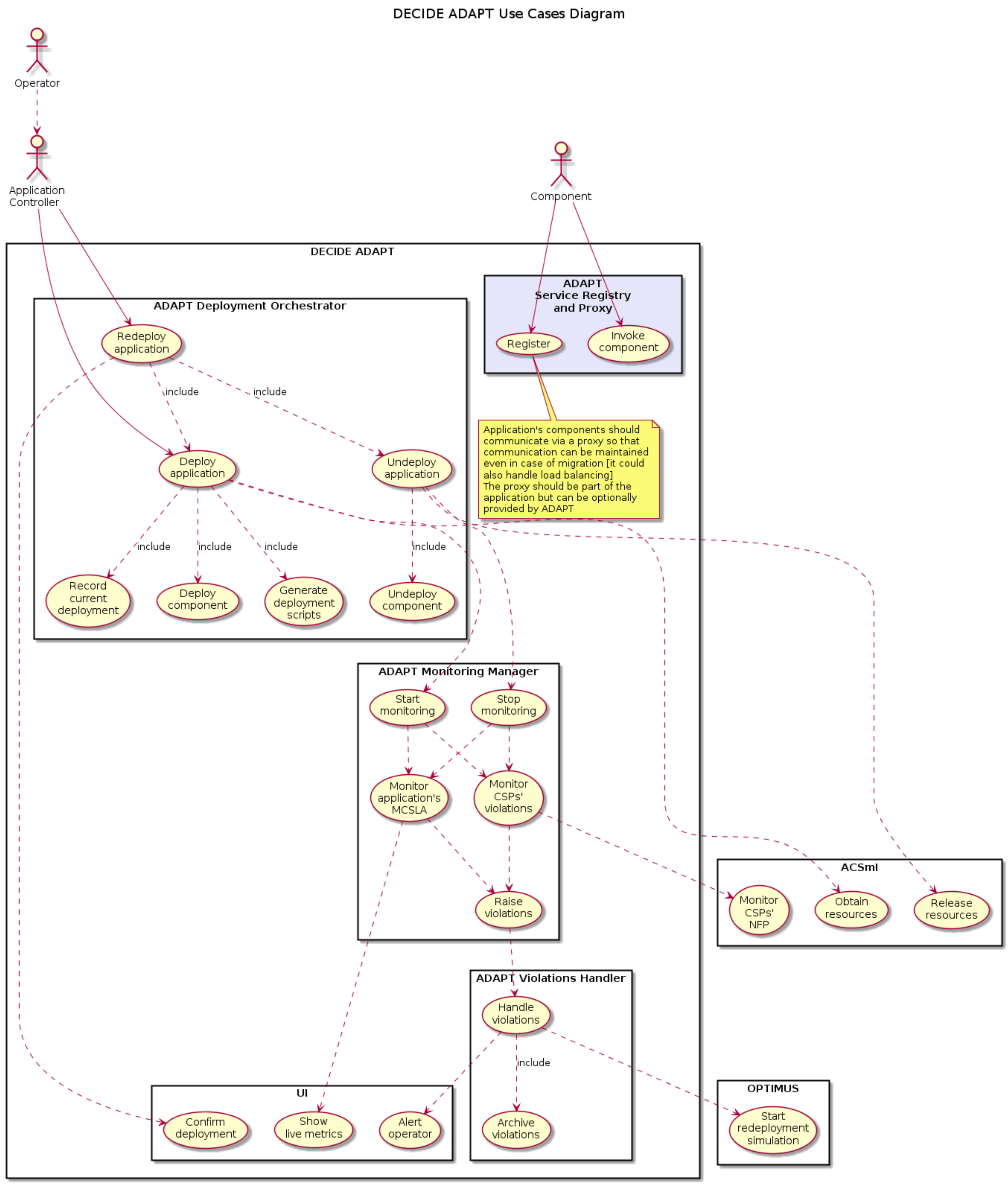


Figure 5. DECIDE ADAPT Use Case Diagram

3.2.1 ADAPT Deployment Orchestrator

UCDO01 – Deploy application

This is the main functionality of ADAPT, called by the DECIDE Application Controller. ADAPT will get all the needed information from the input Application Description document, as indicated in requirements WP4-REQ34 and WP4-REQ37, and will orchestrate the multi-cloud deployment of the user application. The Application Description is detailed is section 5.1.

This use case represents the first deployment of the user application, any further deployment is represented by Use Case UCDO03. No confirmation is needed by the operator for this first deployment.

The Use Case starts by obtaining the required resource for each Cloud Service Provider indicated in the Application Description (uses UCACSmI02), then generates the deployment scripts (uses UCDO06), performs the deployment of each of the given components (uses UCDO02), and finally starts up monitoring for the application (uses UCMM01). The new deployment state is then archived (uses UCDO07).

This use case is also responsible for ensuring that the resources obtained through ACSmI offer a proper container environment to the UCDO02 included use case. For example, in case the resources are plain Virtual Machines, it should setup the (Docker) container environment into them.

UCDO02 – Deploy component

As component to be deployed here we mean a containerized (micro)-service, as indicated by requisite WP4-REQ36.

Prerequisite for deploying a component is a running container (i.e. Docker) environment.

The first step for this use case is to retrieve the container image, using the information provided in the Application Description; then to apply any needed configuration, such as networking, storage, or even monitoring support; and finally to start the container, with all the required parameters that will ensure a smooth execution of the included microservice(s), such as ports to be published, environment variables to be defined, etc.

UCDO03 – Redeploy application

When the ADAPT deployment entry point is called for a running application, the redeployment functionality is invoked.

If the application is flagged as a High Technology Risk application (see requirements WP4-REQ20, WP4-REQ21 and WP4-REQ22), ADAPT will first ask the operator to confirm the deployment configuration indicated in the Application Description. The following logical steps are to undeploy the running application (uses UCDO04) and to deploy the application according to the new configuration (uses UCDO01).

This functionality can be optimized in several ways, if the right conditions apply. For example, if only a subset of the application components should be redeployed, only those components should be stopped and then restarted, possibly elsewhere. In this case the stop/start monitoring operations should impact only the redeployed components. Another optimization, which can be applied if ADAPT can control the Service Registry and Proxy component of the application, is the dynamic management of the service endpoints mapping described in Section 3.2.5, which aims at minimizing application downtime.

UCDO04 – Undeploy application

Both in case of a redeployment and in case of a complete stop, a subset or all of the micro-services composing the application should be undeployed. Completely undeploying an application entails undoing its last deployment.

The original Application Description (either kept by ADAPT or retrieved from elsewhere) should be available, along with further information about the last deployment of the application, such as the address and SSH keys of the VMs hosting the components (likely kept by ADAPT).

The steps needed for undeploying an application are opposite to those used for deployment, both in type and in order, and should be executed for each one of the CSPs where the application has been deployed: stop monitoring (use UCMM05[3]), undeploy each component (uses UCDO05), and finally release the resources (use UCACSmI03).

UCDO05 – Undeploy component

Undeploying a component will stop the related application functionality. In the interest of business continuity, even if the component is stateless (see requirement WP4-REQ35), this should happen as gracefully as possible, ideally by stopping all the incoming requests and waiting until all the current ones return their result to the caller. This graceful behavior should be supported by the application, which typically can implement it by handling a proper warning signal in the shortest possible time[4].

The first step for undeploying a component is therefore to warn the running process that a stop will shortly follow, then the processing of the component is stopped and finally any needed cleanup is performed.

UCDO06 – Generate deployent scripts

The deployment scripts, which will operate to deploy components on the resources obtained from ACSmI, are generated for the specific implementation technology starting from the information in the application description. This functionality has been moved from OPTIMUS to ADAPT: see an explanation in section 4.1.

UCDO07 – Record current deployment

After the deployment has been performed the new deployment state is archived.

3.2.2 ADAPT Monitoring Manager

UCMM01 – Start monitoring

Once the deployment of the application is triggered by the ADAPT deployment orchestrator, the monitoring request occurs. The ADAPT Monitoring Manager will receive the request for starting the monitoring of the application. The 'Start monitoring' process will include the following actions:

- Request the SLOs, the NFRs and the MCSLA values to be monitored
- Establish the (new) measurements to be gathered both at software level and at resource level. If any of the established metrics is already being monitored, only the new ones will be configured and deployed for monitoring.
- Deploy the data collection means (i.e. agents) along with the multi-cloud application components.
- Launch the Monitor CSP's violation process.

[3] Note that during redeployment only the part of the monitoring functionality related to the old infrastructure should be stopped: monitoring the high level application SLA should continue, otherwise the redeployment operation would artificially appear not to have any impact on the application availability.

[4] The `docker stop` command for example takes a parameter (-t) to specify the timeout between the SIGTERM and SIGKILL signals sent to the main process inside the container.

UCMM02 – Monitor application's MCSLA

After the monitoring of the application has been configured, the monitoring process starts. The deployed agents continously provide measurements to be stored (sampling of the metrics). The ADAPT monitoring components aggregates the stored data to create the required values to be compared with the established SLOs and MCSLA thresholds. The monitored measures are provided to the multi-cloud application through the UI.

When an application is not being monitored any more, the ADAPT monitoring component will mark the corresponding registries in the data base as not being monitored (they will not receive any measurement anymore). The corresponding measures are not being shown any more in the user interface.

UCMM03 – Monitor CSPs' violations

This functionality covers the request for start/stop the monitoring the metrics of the CSPs and the subscription to the monitorization channel (listener to the CSPs metrics).

UCMM04 – Raise violations

The measured and compiled data is continously being assessed against the SLOs and thresholds. When a violation is detected an alert is sent by the ADAPT Monitoring to the operator. At the same time the violation detection is sent to the Violations Handler so that the corresponding actions are triggered.

UCMM05 – Stop monitoring

When the ADAPT Monitoring receives the request for stopping the monitoring for any of the components of the application, the following actions need to be performed:

- Stop the monitoring agents and release the corresponding resources.
- Stop the calculation and assessment of the aggregated measurements.

3.2.3 ADAPT Violations Handler

UCVH01 – Handle violations

When a violation occurs, ADAPT is able to automatically perform certain actions, such as redeploying the application to ensure that it meets the established SLOs and MCSLAs. The ADAPT Violations Handler receives violation notifications from the Monitoring Manager and:

- Notifies the operator by email.
- Communicates with OPTIMUS, providing it with the necessary information to trigger a new simulation.

If the application is a low technological risk one, the operator will be notified and OPTIMUS will be triggered. Then, according to the results of the simulation, the application will be automatically redeployed by ADAPT.

If it is a high technological risk one, the operator will be notified and OPTIMUS triggered as well, but the operator will have the chance to modify the simulation parameters before it takes place, and the redeployment will not occur without the operator's confirmation.

UCVH02 – Archive violations

This functionality takes care of storing the history of alerts, along with the information about what caused the alert.

3.2.4 ADAPT UI

UCUI01 – Confirm deployment

In case the application has been identified as high technological risk, as indicated by requirements WP4-REQ21, WP4-REQ22 and WP4-REQ23, its redeployment cannot be automatic but must be confirmed by the operator.

The User Interface will propose to the operator the new deployment configuration, as calculated by OPTIMUS. Any selection of deployment alternatives has already been done as part of the OPTIMUS workflow. The new configuration may involve using resources from one or more Cloud Service Provider, which must have already been contracted by ACSmI. The operator has just to confirm that the selected configuration can be deployed.

Confirming the previously selected configuration automatically selects the related deployment scripts and authorizes their execution.

UCUI02 – Alert operator

When a violation occurs, this functionality will take care of notifying the operator by means of an email that contains the most relevant information about the violation and to display the alert on the UI.

UCUI03 – Show live metrics

ADAPT will show the monitored metrics through the UI. A dashboard with the status of the monitored NFRs will be shown, as well as more detail for each metric being measured (SLO of the metric vs. actual measured value. As the metrics will be continously measured, the operator of the multi-cloud application can access the dashboard whenever he or she wants during the operation of the application.

3.2.5 ADAPT Service Registry and Proxy

Application's components (microservices) should communicate via a proxy so that communication can be maintained even in case of migration (it could also handle load balancing). The proxy should be part of the application and is a useful pattern that ARCHITECT could propose to the developer, but ADAPT can also optionally provide it.

This component is shown with a grey background in **Figure *6*** to indicate that it is an optional component.

UCSP01 – Register

This functionality allows a component to update the Service Registry of the application to indicate its reachability status and the related endpoint coordinates. This information is used by the Proxy to properly redirect incoming calls to the component.

UCSP02 – Invoke component

This functionality allows an application component (or even an external client) to invoke a registered application component. If the desired component is available the incoming call is redirected to it by the (reverse) Proxy, using the registered endpoint. A circuit breaker pattern can be optionally applied to avoid too many repeated timeouts in case of service failure.

This proxying functionality can include load balancing, in case more than one endpoint is registered for the same component.

The Proxy, by interposing in every call to a component, could also allow to monitor specific application metrics, such as availability or response time.

3.2.6 External DECIDE components

Use cases in this section briefly describe functionalities expected to be implemented by other DECIDE components external to ADAPT.

UCACSmI01 – Monitor CSPs' NFP

When a deployment configuration has been selected, and the deployment of the multi-cloud application has been performed, the monitoring component of ADAPT needs to start the monitoring of the behaviour of the multi-cloud application and the underlying cloud resources. The ADAPT monitoring component will send the request to the ACSmI for getting the needed metrics from the cloud resources where the components are being deployed. The collection of this data will be continously collected as they are real data.

UCACSmI02 – Obtain resources

This use case is the first step of the UCDO01 – Deploy application use case. When the DECIDE Application Controller calls ADAPT, this need to request ACSmI to obtain the needed resources from the CSPs to deploy the different components of the multi-cloud application. The information about the concrete resources needed in the request is obtained from the Application Description. The Application Description will be updated by the ACSmI with the corresponding information so that the application can be deployed in the selected cloud resources (UCDO01).

UCACSmI03 – Release resources

Once a violation has been detected the re-deployment process starts (UCDO03 – Redeploy application). As part of this re-deployment process and once the component have been undeployed from the cloud resources (UCDO05 – Undeploy component), the "old" resources need to be released. The request to release these resources is sent form ADAPT to the ACSmI, which will release the resources and send a confirmation back to ADAPT, stating that the release of the resources have been successfully implemented.

UCOPTIMUS01 – Start redeployment simulation

When ADAPT identifies a violation in the application MCSLA, or in one of the supporting CSPs' SLAs, a new deployment configuration has to be found to cope with the identified issue. In this case ADAPT asks OPTIMUS to start a redeployment simulation, passing as a parameter details about the identified violation (see requirements WP4-REQ27 and WP4-REQ43) so that the simulated configuration can provide a solution to it.

3.3 Requirements mapping

As already indicated in section 2.3, a detailed analysis has been performed to map lower level requirements with the identified use cases. The result of this work is reported in the following sections, dedicated to the major ADAPT components.

3.3.1 Deployment requirements mapping

The following **Table *6*** shows the mapping between ADAPT deployment-related use cases and their requirements. The requirements' description is the same as that reported in section 2 and has been repeated here for reading convenience.

Table 6 . Mapping between deployment use cases and requirements

Req. ID	Description	UCDO 01 Deploy application	UCDO 02 Deploy component	UCDO 03 Redeploy application	UCDO 04 Undeploy application	UCDO05 Undeploy component	UCDO-06 Generate deployment scripts	UCDO 07 Record current deployment	UCU-I01 Confirm deployment
WP4-REQ1	DECIDE ADAPT will support the self-adaptation and dynamic re-deployment of (parts of) multi-cloud applications when certain conditions are not met.	X	X	X	X	X		X	
WP4-REQ3	DECIDE ADAPT will pro-actively adjust the running configuration of the application			X			X		
WP4-REQ9	DECIDE ADAPT [KR5], the application self-adaptation tool will support automated dynamic deployment of service components as well as the runtime monitoring of functional and non-functional service properties.		X			X			
WP4-REQ12	DECIDE OPTIMUS [..] will provide [..] automation of the provisioning resources and deployment scripts for multi-cloud native applications						X		
WP4-REQ14	The developer will select the deployment scripts based on the selected configuration from the simulation phase through the continuous deployment supporting tools and the architectural patterns for deployment								X
WP4-REQ15	Each deployment configuration will be stored in the multi-cloud native application controller, maintaining the current deployment configuration							X	

Req. ID	Description	UCDO 01 Deploy application	UCDO 02 Deploy component	UCDO 03 Redeploy application	UCDO 04 Undeploy application	UCDO05 Undeploy component	UCDO-06 Generate deployment scripts	UCDO 07 Record current deployment	UCU-I01 Confirm deployment
	situation as well as the historic of the previous deployment configuration used, so that they can be checked in the re-deployment phase								
WP4-REQ21	If the application configuration has been established as of low technological risk, the multi-cloud application will be self-adaptive and it will be redeployed automatically, following a new deployment configuration.	X		X	X				
WP4-REQ23	[DECIDE ADAPT] will support the selection of the new deployment scripts (based on the architectural patterns for deployment), and thus semi-automatically re-deploy it [the application].			X					X
WP4-REQ28	DECIDE ADAPT is a framework providing a deployment orchestrator (workflow manager for orchestrating the actions required for deploying the multi-cloud application)	X	X						
WP4-REQ30	DECIDE ADAPT is a framework providing an adaptation workflow manager (establishes the workflow and performs the adaptations actions)			X					
WP4-REQ34	The actual deployment, monitoring and adaptation of an application is obtained "feeding" DECIDE ADAPT with a description of the application containing the related helpers	X	X	X	X	X			X
WP4-REQ35	ADAPT will support applications based on composition of stateless (possibly micro) services				X	X			

Req. ID	Description	UCDO 01 Deploy application	UCDO 02 Deploy component	UCDO 03 Redeploy application	UCDO 04 Undeploy application	UCDO05 Undeploy component	UCDO-06 Generate deployment scripts	UCDO 07 Record current deployment	UCU-I01 Confirm deployment
WP4-REQ36	ADAPT will support composable applications where each composition unit is a containerized service		X			X			
WP4-REQ44	Users will perceive relevant improvements in the business continuity since as soon as there is a problem (i.e. lack of resource due to a peak of requests) the software is automatically re-adapted and re-deployed			X	X	X			
DEVOPS-REQF6	DECIDE framework must support the continuous deployment of the developed apps	X	X						
DEVOPS-REQF16	DECIDE must support the semi-automatic adaptation and redeployment of the application into new cloud services when needed based on the assessment of the continuous monitoring			X					X

3.3.2 Monitoring requirements mapping

The following **Table *7*** shows the mapping between ADAPT monitoring-related use cases and their requirements. The requirements' description is the same as that reported in section 2 and has been repeated here for reading convenience.

Table 7. Mapping between monitoring use cases and requirements

Req. ID	Description	UCMM01 Start monitoring	UCMM02 Monitor application's MCSLA	UCMM03 Monitor CSPs' violations	UCMM04 Raise violations	UCMM05 Stop monitoring	UCUI03 Show live metrics
WP4-REQ2	[Conditions not met] include for instance, that the defined composite multi-cloud application SLA is not being fulfilled, the application is not performing as established or the cloud service providers (CSPs) are violating the contracted SLAs.				X		
WP4-REQ4	{Adjustment will be} based on measurements that are derived from the dynamic monitoring activities of both the application and the non-functional properties of the CSPs and cloud offerings where the application is deployed and making use of.		X				
WP4-REQ18	During the application operation phase, the DECIDE self-adaptation application provisioning tool (ADAPT [KR5]) will continuously monitor and assess the fulfillment of the established NFR and MCSLA		X	X			
WP4-REQ19	If a violation of any of the [former: NFR and MCSLA] metrics occurs, the self-adaptation tool through the ACSmI will assess the operation of the (combination of) cloud services selected and discard those that are affecting the MCSLA		X	X	X		
WP4-REQ27	DECIDE ADAPT [KR5] will provide the operator a report with the NFPs that are not being fulfilled and an input file to be able to simulate a new deployment topology through DECIDE OPTIMUS						
WP4-REQ29	DECIDE ADAPT is a framework providing a monitoring engine to monitor if the working conditions meet the NFR and to trigger alarms (high technological risks) or actions (low technological risks)		X	X	X		
WP4-REQ34	The actual deployment, monitoring and adaptation of an application is obtained "feeding" DECIDE ADAPT with a description of the application containing the related helpers	X			X	X	

Req. ID	Description	UCMM01 Start monitoring	UCMM02 Monitor application's MCSLA	UCMM03 Monitor CSPs' violations	UCMM04 Raise violations	UCMM05 Stop monitoring	UCUI03 Show live metrics
WP4-REQ41	DECIDE ADAPT [KR5] can also be easily extended with the inclusion of more NFPs that need to be measured, as well to different types of applications such as those following the mobile cloud paradigm or IoT	X	X				
DEVOPS-REQF3	DECIDE framework must be able to monitor the deployed micro-services	X	X			X	
DEVOPS-REQF9	DECIDE framework must be able to track the issues that affect the deployed services and use registries to store this information			X			X
DEVOPS-REQF15	DECIDE must support the monitoring of the multi-cloud application SLA and the SLAs of the underlying cloud resources		X	X			
DEVOPS-REQF17	DECIDE must support the monitoring of the SLAs of the underlying cloud resources		X	X			

3.3.3 Violations Handling requirements mapping

The following **Table *8*** shows the mapping between the use cases related to violations handling and their requirements. The requirements' description is the same as that reported in section 2 and has been repeated here for reading convenience.

Table 8. Mapping between violation handling use cases and requirements

Req. ID	Description	UCVH01 Handle violations	UCVH02 Archive violations	UCUI02 Alert operator
WP4-REQ19	If a violation of any of the [former: NFR and MCSLA] metrics occurs, the self-adaptation tool through the ACSmI will assess the operation of the (combination of) cloud services selected and discard those that are affecting the MCSLA	X		

Req. ID	Description	UCVH01 Handle violations	UCVH02 Archive violations	UCUI02 Alert operator
WP4-REQ21	If the application configuration has been established as of low technological risk, the multi-cloud application will be self-adaptive and it will be redeployed automatically, following a new deployment configuration.	X		
WP4-REQ22	In case the application has been identified as high technological risk, once it has identified the aspects that are affecting the malfunctioning of the application, it will alert the operator and using the OPTIMUS tool it will look for new (combination of) cloud services to set up a new deployment schema	X		X
WP4-REQ27	DECIDE ADAPT [KR5] will provide the operator a report with the NFPs that are not being fulfilled and an input file to be able to simulate a new deployment topology through DECIDE OPTIMUS	X		X
WP4-REQ40	KPI EI3.3: fulfilment of the application's MCSLA by 99%		X	
WP4-REQ43	If the application configurations are classified with high technological risk, [re-adaptation involves] simulating again the deployment with DECIDE OPTIMUS but in a more driven and accurate way, entering now as input, the identified problems so that the new configuration can provide a solution to that problem.	X		
DEVOPS-REQF9	DECIDE framework must be able to track the issues that affect the deployed services and use registries to store this information	X		

3.4 Improving business continuity

Continuous Integration followed by Continuous Deployment entails many incremental development steps, each followed by a software release, leading to frequent application deployments, which may impact applications' uptime and therefore business continuity.

On the other end DECIDE promises to improve business continuity (WP4-REQ44) by automatically re-adapting and re-deploying an application as soon as a problem is raised (e.g. lack of resources due to a peak of requests), but also coping with SLA violations of the running production software usually results in further deployments.

Redeployment of an application involves stopping and undeploying the current release, or at least part of it, and deploying the new one. The expected high number of application redeployments therefore poses a fundamental question: how to redeploy an application without impacting business continuity? Or at least: how to lower impact on business continuity while redeploying an application? The key requirement here is to minimize application's downtime during redeployment.

The first answer to this requirement is to build cloud-native applications according to the **microservices architecture** (see section 2.4.2). In this way an application is built up by multiple cooperating services, each one providing its own specialized functionality. Each of these services could be stopped and restarted independently from the others, thus adding a finer granularity dimension to redeployment and laying the basis to avoid stopping the whole application when releasing a new version.

Building an application with microservices is just the first step, but is not enough to have redeployments which do not impact availability: to be able to easily stop and restart a microservice, it should be **stateless**. Stopping a stateless microservice will not result in data loss (even if it may lead to data inconsistency if the application is not properly designed). Obviously if the stopped microservice is the only one performing a particular functionality, then that functionality will not be available again until the microservice is restarted, possibly elsewhere.

The next step towards low impacting redeployments is therefore the **replication** of microservices. If the application has more than one instance (microservice) of a functionality running, then the single microservice can in theory be redeployed without impacting availability of that functionality. The full benefit of replication comes when it is coupled with load balancing: the load balancer evenly distributes requests among the active instances and is aware of when an instance is shutting down thus redirecting the load to other microservices.

Speaking about load redirection it becomes apparent the need for a fundamental pattern in microservices-based applications: the **service registry**. This registry allows clients to discover the location (host and port) of the active instances for a specific service. Every instance advertises its availability status to the registry in both its startup and shutdown phases. In this way requests can be directed only to available instances and a graceful service shutdown is made possible: the terminating microservice will not get additional requests and can wait until all the current ones return their result to the caller before actually stopping.

The service registry maps services to their endpoints, and it can be used either through client-side or server-side discovery. In the first case it is the client (directly or through some utility layer) which accesses the registry to discover the location of any needed service. In the second case there is a router or **proxy** component between the client and the service: the client always invokes the same proxy endpoint for a given service, and the proxy takes care of accessing the registry and redirecting the request to an active microservice, possibly load-balancing it.

A further step towards business continuity-friendly redeployments is the technique commonly called "**Blue-green deployment**" [14]. The idea is to keep both the old and new deployments running at the same time and use the proxy / router component in front of them to switch from the old to the new one. This allows to deploy the new release in production in a "hidden" environment, test it, and switch it online when all is ready and running. After switching all client requests are redirected to the new release and at the same time the old one becomes hidden and is ready to be stopped and undeployed.

4 DECIDE ADAPT Architecture

A good high level description of DECIDE ADAPT is the following (see requirement WP4-REQ45). *DECIDE ADAPT is a tool that allows the (semi-) automatic adaptation of the application and re-deployment in another multi-cloud configuration when certain conditions are not met. These conditions are on one hand the violations of the application's own multi-cloud SLA (MCSLA) and on the other hand, the non-fulfilment of the NFP of the CSPs where the application is deployed.*

These conditions will trigger an alert and will cause the OPTIMUS tool to be launched again in order to search for another deployment configuration. Depending on the technological complexity requirement, and the initially prioritized requirements by the user, the application will be re-adapted automatically or an alert to the operator will be launched along with a diagnosis of what malfunctioned so that a new optimal configuration can be found.

A logical view of DECIDE ADAPT architecture is shown in the following **Figure *6***.

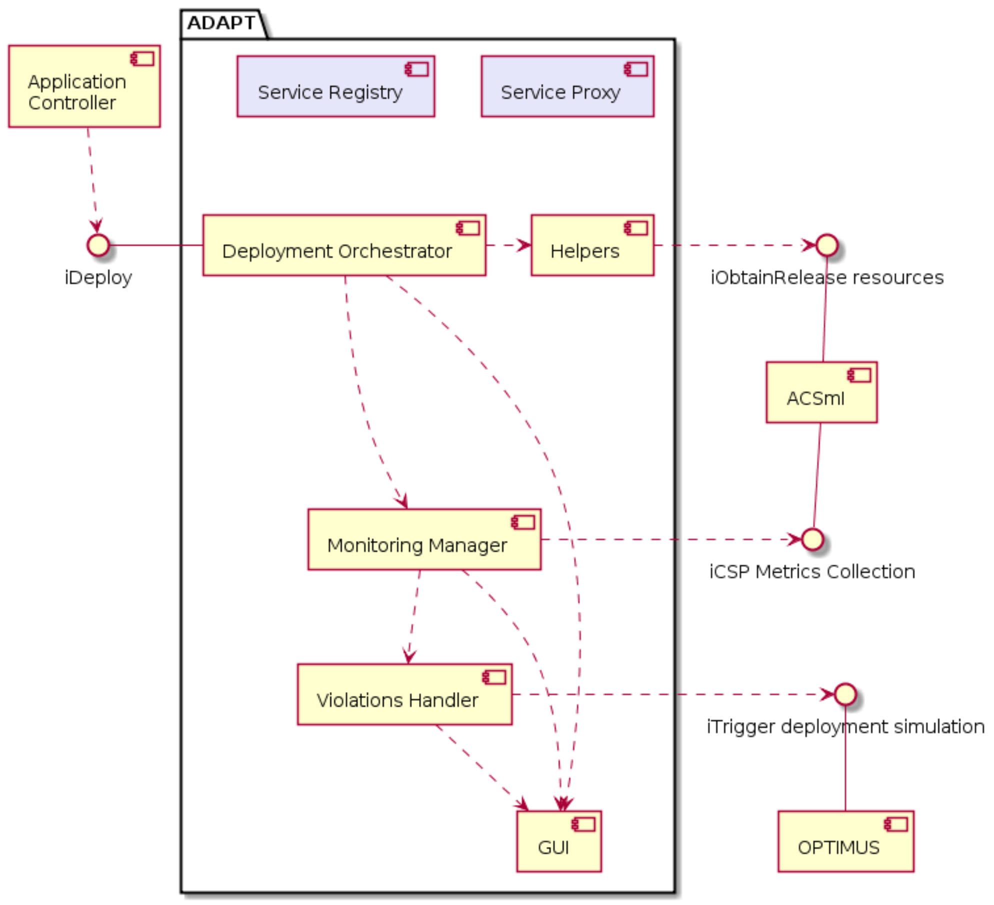


Figure 6. DECIDE ADAPT logical architecture

The main components of DECIDE ADAPT are the following.

Deployment Orchestrator. The Deployment Orchestrator is in charge of orchestrating the deployment lifecycle (deployment, undeployment, user confirmation, redeployment) for user applications and their components. More details on the Deployment Orchestrator can be found in deliverable D4.4.

Monitoring Manager. The Monitoring Manager controls the monitoring functionality for the application, according to its defined (Multi-Cloud) SLA, and identifies and raises any related violations, including those for the CSPs where the application is deployed on. More details on the Monitoring Manager can be found in deliverable D4.7.

Violations Handler. The Violations Handler will handle any violation raised by the Monitoring Manager, either regarding the application MCSLA or the CSPs' NFRs. Violation handling may lead both to alerting the operator and to contacting OPTIMUS to trigger a new re-deployment simulation for the application., thus starting a readaptation process.

Service registry / Service proxy. The Service registry and proxy are components, usuallly part of the application iself, which enable to maintain microservices communication even in case of migration (they may also handle load balancing). ADAPT can optionally provide them as implementations of the respective patterns.

Helpers. The Helpers 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. In the first version of ADAPT the Helpers support the Deployment Orchestrator in interfacing with ACSmI. More details can be found in deliverable D4.10.

The main input information for all ADAPT functionalities is the Application Description document, and the main interfaces of ADAPT are with OPTIMUS, ACSmI and the ADAPT GUI.

4.1 Deployment components

The DECIDE ADAPT deployment components get as input the Application Desctiption and are in charge of deploying and undeploying the application and its components on the indicated cloud providers. Infrastructure resources from the cloud providers are obtained and released through ACSmI.

Application adaptation to cope with SLA violations is obtained by redeploying the application according to the new configuration calculated by OPTIMUS.

After some discussion among the DECIDE partners, it has been decided to move the deployment script generation functionality from the Application Controller to ADAPT itself, since the scripts are closely dependent on the technology selected for the ADAPT implementation. This allows to keep low level information about resources in a single place and to handle in a single component the whole deployment flow from user confirmation to resource provisioning to the actual deployment.

The ADAPT Deployment Orchestrator is composed by the following components.

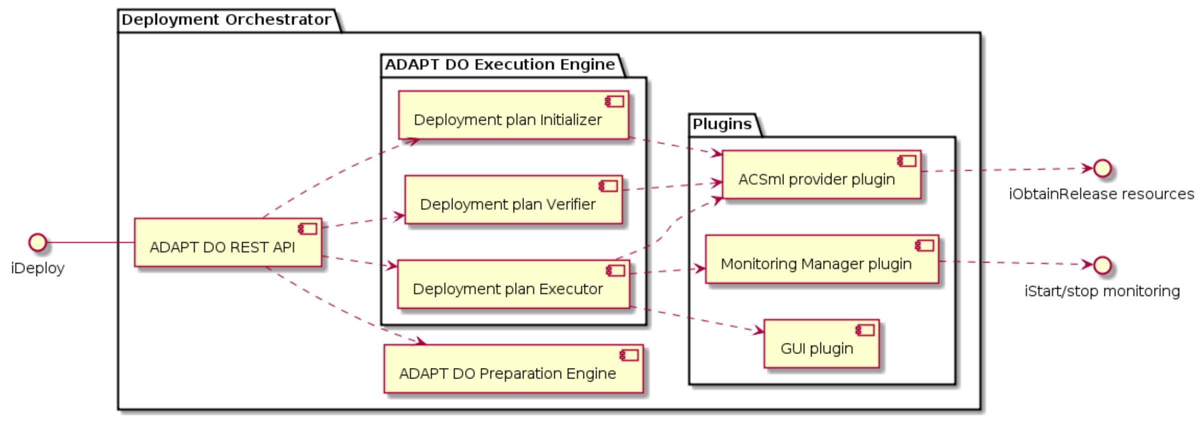


Figure 7. ADAPT Deployment Orchestrator components

ADAPT DO REST API. This is the main component of the ADAPT Deeployment Orchestrator. It orchestrates the needed deployment steps, manages the generation of the deployment plan, manages the plugins of the ADAPT DO Execution Engine and activates it, asks for confirmation if needed, archives the deployment state.

ADAPT DO Preparation Engine. This component, based on the information inluded in the Application Description, generates all the needed deployment plans and scripts to be executed by the ADAPT DO Execution Engine.

Deployment plan Initializer. The initializer is part of the ADAPT DO Execution Engine; it creates an initial deployment state for each environment and also verifies the availability of the required plugins

Deployment plan Verifier. The Verifier is part of the ADAPT DO Execution Engine and is needed to check the generated deployment plan before executing it.

Deployment plan Executor. The Executor is the main part of the ADAPT DO Execution Engine. It executes the generated deployment plan and activates the needed plugins depending on what is specified in the plan. A deployment plan defines the infrastructure resources needed for deployment along with their expected configuration and any command script to bootstrap them.

ACSmI provider plugin. The ACSmI plugin allows to use ACSmI as a service 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.

Monitoring Manager plugin. This plugin allows the Deployment Executor to interact with the ADAPT Monitoring Manager for starting / stopping monitoring of the deployed application.

GUI plugin. This plugin allows the Deployment Executor to interact with the operator, mainly for confirming the deployment of high technology risk applications.

4.2 Monitoring components

The DECIDE ADAPT monitoring components monitor the deployed multi-cloud based application and verify that the non-funtional requirements and the SLOs are being fullfilled. If a violation of any of the NFRs or SLOs is detected, ADAPT monitoring components will inform the violation handlers component which will generate the proper actions depending on each situation and context. If the violation occurs, an information message saying that the working conditions are not met will be sent to the operator. If the application is low technology risk, the “adaptation” process will be launched, through the violation handlers component.

The main functionalities of the ADAPT monitoring components are:

- Collect data from the deployed multi-cloud application and the underlying cloud resources: The ADAPT monitoring component needs to get the data from the deployed components and their NFRs as well as from the underlying cloud services to collect data from their service level metrics at real time. The data related to the cloud services will be collected from ACSmI.
- Store the data collected from the deployed multi-cloud application and the underlying cloud resources: To assess the required working conditions of the multi-cloud application ADAPT monitoring will deal with data at real time. These time series will be stored for further analysis of the data.
- Create the aggregated data to be assessed: from the raw metrics, the ADAPT monitoring component will need to create aggregated data for assessing the violations.
- Visualize the measurements: DECIDE ADAPT monitoring will provide the user with an interface to visualize the monitored data, from the multi-cloud application and from the underlying

cloud services. DECIDE ADAPT monitoring will also provide monitoring information of the violation occurred.

- Detect when a violation occurs: DECIDE ADAPT will detect violations on the working conditions (NFRs/SLOs), send the corresponding alert to the operator and launch the "adaptation" process. The adaptation process launched will include the actions related to stop the monitoring of the previous components and resources. Monitoring of the resources will not be stopped until the resources are undeployed.

These functionalities will be covered by the following sub-components inside ADAPT monitoring:

- ADAPT M manager: This sub-component is in charge of managing the different processes and requests that need to be triggered in each of the other sub-components of the ADAPT monitoring. ADAPT M manager, will launch the following processes:
    - Start monitoring
    - Stop monitoring
    - Monitor application's MCSLA
    - Monitor CSPs violations
    - Raise violations
- ADAPT M Data collection: This sub-component will collect the data from different sources. On one hand, it will collect data from the resources where the different micro services are deployed (through ACSmI monitoring). On the other hand, it will collect data from the microservices themselves. The Data collection sub-component will be based on agents deployed withtin the different components and cloud resources. The agents will be pre-defined and pre-implemented to be installed when deploying the multi-cloud application. The DAPT M Data collection sub-component will receive the requests from the ADAPT deployment orchestrator, both for starting and stopping the monitoring of a multi-cloud application.
- ADAPT M Data storage and aggregation: This sub-component will be in charge of storing the data collected from the ADAPT M Data collection and agreggating it to create the actual measures that will be assessed by the ADAPT M violation detection.
- ADAPT M violation detection: This component will be in charge of assessing that the required working conditions are or are not being met. The ADAPT M violation detection will need to get the SLOs for the different metrics from the MCSLA editor /App description.
- ADAPT M UI: This is the graphical user interface for the ADAPT monitoring component. It will provide the means for the operator of the multi-cloud application to visualize the metrics in run time and the information form the violations ocurred.

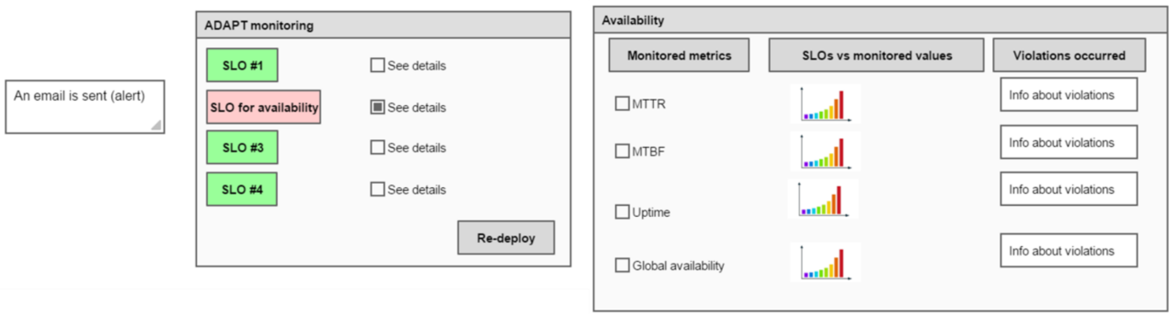


Figure 8. Initial mockups for the ADAPT M UI

In the following **Figure *9***, the component diagram of the ADAPT Monitoring component is shown.

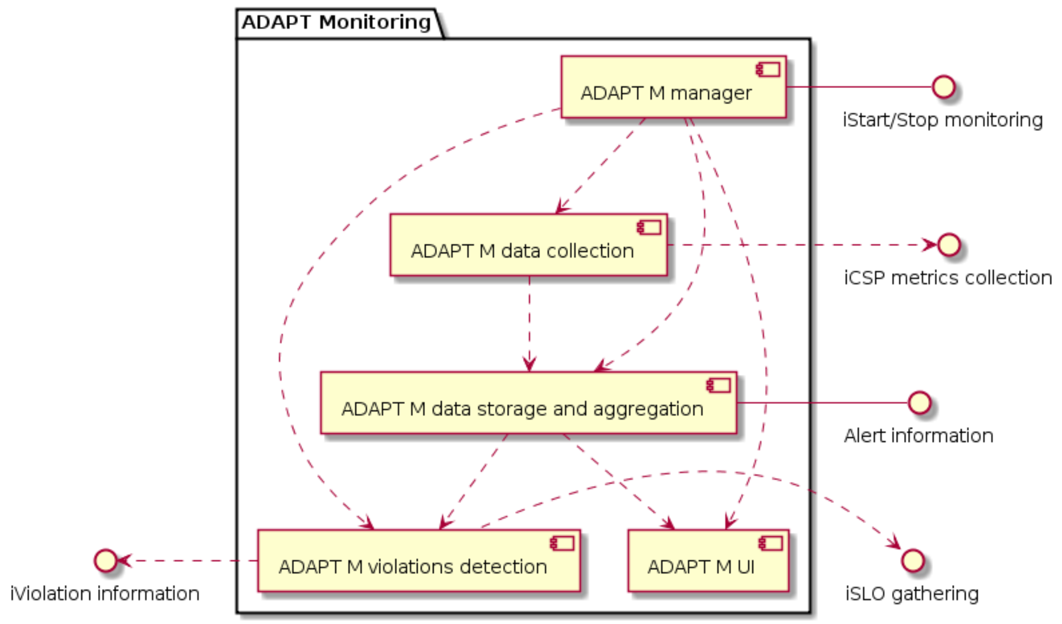


Figure 9. ADAPT monitoring internal component diagram

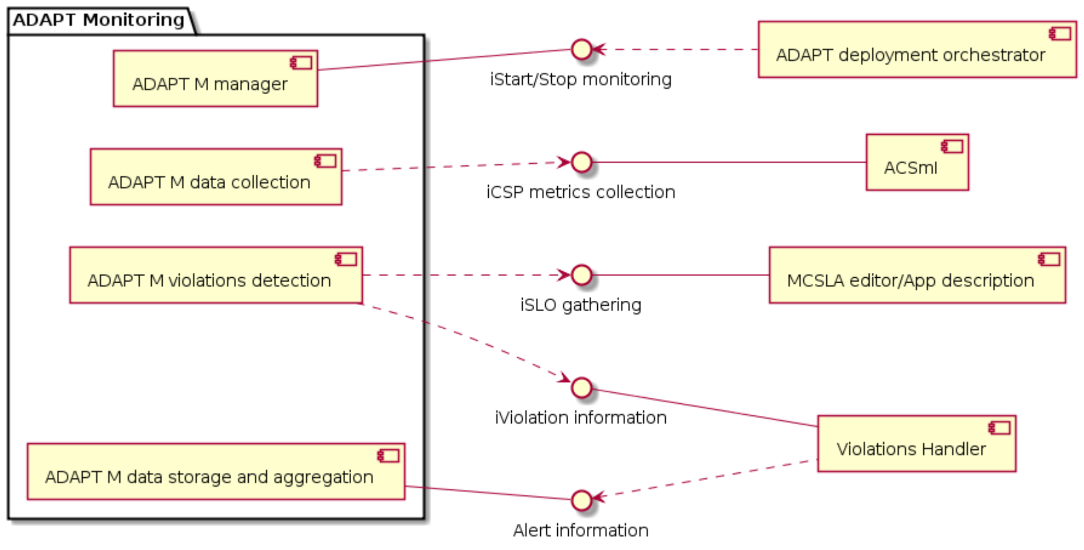
ADAPT monitoring will gather information from the the MCSLA editor/ App description (the threshold values for the different metrics to be assessed,) , from the Violation Handlers (information about the alerts), and form ACSmI (information about the metrics coming from the CSPs).

ADAPT monitoring will provide information to the Violation Handlers about any violation occurred.

ADAPT monitoring will receive requests from ADAPT deployment (through ADAPT deployment orchestrator) to start/stop the monitoring.

In the following **Figure *10*** these communications are shown:

Figure 10. ADAPT monitoring external component diagram



4.3 Violation Handlers

The Violation Handlers are in charge of notifying the operator in case a violation occurs, triggering the redeployment process whenever it is necessary and storing the history of violations.

When a violation takes place, it is detected by the Monitoring components, which will send a message to the Violation Handlers that includes the parameter that was violated. Then, depending on the application's technological risk, the following actions will take place:

- **Low technological risk**: the operator will be notified and OPTIMUS will be triggered. Then, according to the results of the simulation, the application will be automatically redeployed by ADAPT .
- **High technological risk**: the operator will be notified and OPTIMUS triggered as well, but the operator will have the chance to modify the simulation parameters before it takes place, and the redeployment will not occur without the operator's confirmation.

The following **Figure *11*** shows the component diagram of the Violation Handler:

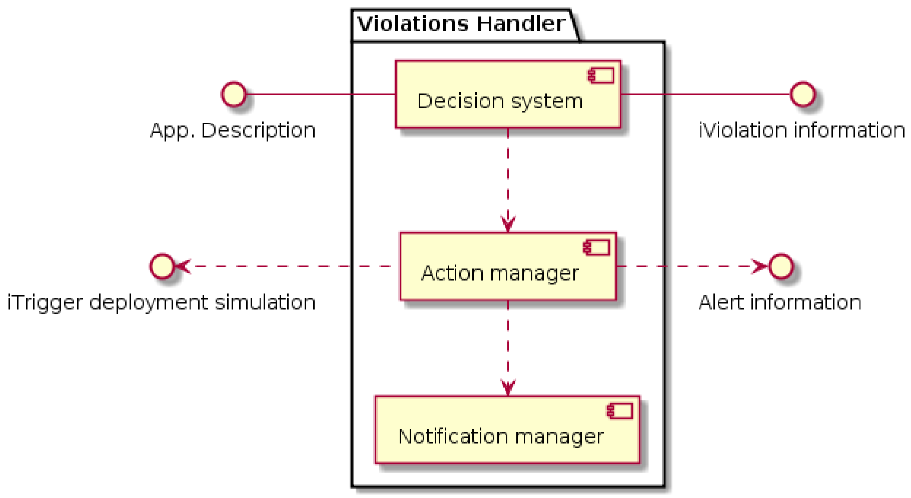


Figure 11. ADAPT Violation Handlers component diagram

The *Decision system* receives the violation and requests the Application Description for the application's technological risk. Depending on this variable, it will instruct the *Action manager* to perform the corresponding actions:

- It will instruct the *Notifications Manager* to notify the operator. The notification will be done through an email that will contain information regarding the alert
- It will send a message to OPTIMUS, containing the necessary data to start a new simulation process

The *Action Manager* will also store the received alert in the database (ADAPT M data storage and aggregation) to keep a history of alerts and to visualize them from ADAPT's UI.

Figure *12* shows these communications:

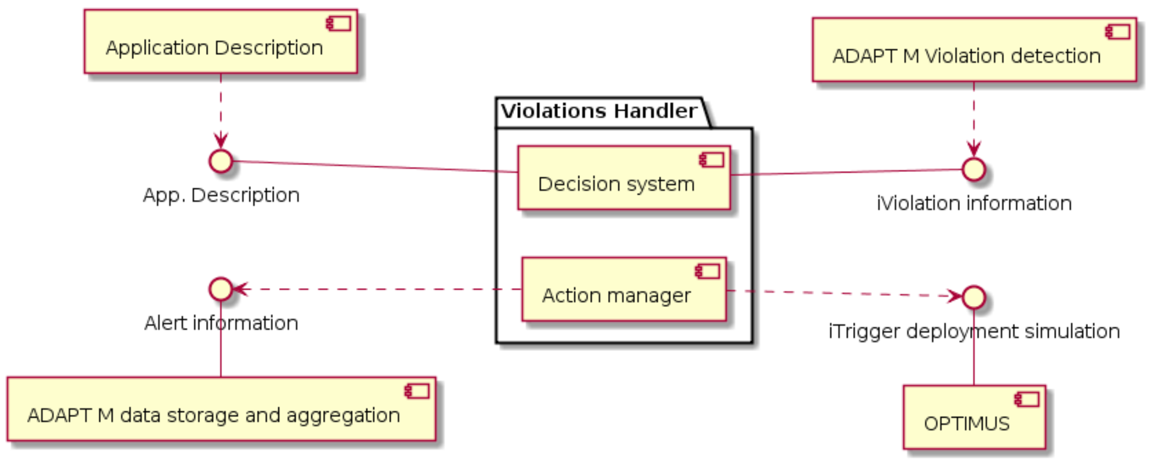


Figure 12. ADAPT Violation Handler communications diagram

5 DECIDE ADAPT Interfaces

DECIDE ADAPT integration points have already been identified and listed in Deliverable D2.1. This section adds more details about the information expected to be exchanged in the main integration points.

5.1 Application Description

The Application Description (see Appendix)is the main source of information for the execution of ADAPT functionalities, both deployment- and monitoring-related.

The Application Description will be a JSON document versioned and hosted in a Git repository, this repository will act as the main orchestration repository for the DECIDE application. The whole text of an Application Description (or a reference to a specific Git version of it) will be passed as a parameter to the call triggering ADAPT deployment functionality.

The Application Description definition has evolved since the first data model shown in the DECIDE deliverable D2.1, and it is still evolving in parallel with the design and implementation iterations. Its main fields are the following.

- microservices: list and detailed description of application's microservices
- app_mcsla: the application's MCSLA
- containers: list and detailed description of application's containers
- virtual_machines: list and detailed description of the virtual machines on which the application should be deployed

The information contained in the Application Description is provided by the tools preceding ADAPT in the DECIDE workflow (see section 3.1).

The current detailed definition of the Application Description is provided in the Appendix to this deliverable.

5.2 Exported interfaces

This section lists interfaces implemented by ADAPT components and used by other DECIDE tools or by ADAPT itself. The following definitions will be detailed and possibly updated during the integration work expected in Y2 after the first implementation.

iDeploy

```
Interface iDeploy {
        + deploy(appdescription uri): void
}
```

This interface allows requesting the ADAPT Deployment Orchestrator component to start the deployment process of the user application described by the given Application Description.

iStart/stop monitoring

```
Interface iStart/stop monitoring {
        + startmonitoring(appdescription uri): void
        + stopmonitoring(appdescription uri): void
}
```

This interface will provide the means for requesting the ADAPT monitoring component to start/finish a specific monitoring process from the ADAPT deployment orchestrator.

This interface contains some operations:

- The *startmonitoring* method starts the monitoring of a certain multi-cloud application.
- The *stopmonitoring* method stops the monitoring of a certain multi-cloud application.

iViolation information

Interface **iViolation information** *{*
+ provide violation(): violation type, component affected, violation ocurred
}

This interface will be consumed by the Violation Handler and provided by ADAPT monitoring.

This interface will serve to inform the Violations Handler component that a violation has occurred. Along with it, information about the violation itself will be also provided (type, metric monitored, etc) so that the Violations Handler can derive the corresponsing action.

The *provide violation ()* method provides the information of a violations that has occurred.

Alert information

Interface **iAlert information** *{*
+ provideAlert(): component, violation ocurred, technological risk, suggested action
}

This interface will be consumed by the Violation Handlers and implemented by ADAPT monitoring.

It serves the purpose of providing the ADAPT monitoring components with the alert details, to be visualized *by ADAPT M data storage and aggregation* module.

The method *provideAlert()* returns the information about the alert to be displayed.

5.3 Consumed interfaces

This section lists interfaces expected to be implemented by other DECIDE components. The following definitions will be detailed and possibly updated during the integration work expected in Y2 after the first implementation.

iCSP metrics collection

Interface ***iCSP metrics collection****{*
+ providesMetric(cloudservice id, metric1, metric2,…, metric n): List <measurements>
}

This interface will be implemented by ACSmI and consumed by ADAPT monitoring. The main objective of this interface is to collect the metrics from the CSP where the different components of the multi-cloud application are deployed.

The *providesMetric ()* method gets a list of actual measurements collected from the CSPs for the metrics specified.

iSLO gathering

```
Interface iSLO gathering {
            + provide SLOs(appdescription uri): List <SLOs>
}
```

This interface will collect the SLOs values from the MCSLA editor or directly from the Application Description. This information will be used by the ADAPT monitoring component to compare the actual metrics with the SLOs of the metrics.

The *provide SLOs ()* method provides a list of the corresponding agreed SLOs for a multi-cloud application.

iObtainRelease resources

```
Interface iObtainRelease resources {
            + Obtainresources(appdescription uri): List <Obtainedresources >
            + Releaseresources (appdescription uri): List <Releasedresources >
}
```

This interface will be implemented by ACSmI and consumed by ADAPT deployment. The main objective of this interface is to receive request for obtaining resources or releasing resources in the ACSmI.

The *Obtainresources ()* method Obtains the resources from the the App Description to deploy the component.

The *Releaseresources ()* method releases the requested resources (from the information in the App Description) when a violation of the working conditions occurs.

iTrigger deployment simulation

```
Interface iTrigger deployment simulation {
            + triggerSimulation(adddescription uri): void
}
```

This interface is implemented by OPTIMUS and consumed by the Violation Handlers.

It allows the Violation Handlers to instruct OPTIMUS to carry out a new simulation automatically, in case of a violation in a low technological level application.

The method *triggerSimulation()* provides OPTIMUS with the necessary variables to trigger a new simulation.

6 Candidate implementation technologies

This section introduces some existing technologies that have been identified as reusable starting points for the implementation of ADAPT.

6.1 Terraform

Terraform [15] is a model-based open source tool created by HashiCorp that allows to create or update IT infrastructure resources and prepare them for use by applications. Terraform implements the "Infrastructure as Code" paradigm which is at the base of the DevOps approach (see DECIDE D2.1 [2], appendix A2). The Community version of Terraform is licensed under the Mozilla Public License 2.0; two Enterprise versions (Pro and Premium) are also available with additional features and support.

A Terraform Plan document is the model or "code" that allows to define the IT resources that should compose the required infrastructure. The automation of IT resources creation and configuration based on a plan makes this process more reliable and repeatable.

The Terraform plan describes the final desired state of the required infrastructure resources. Applying the plan results in transitioning from the current state to the defined target state. Applying twice the same plan, assuming the first application is successful, is an idempotent operation.

Terraform automatically archives the deployment state, and this storage can be externalized, using backends[5]; for example Consul can be configured as a Terraform backend.

Resources defined in the plan may depend on one another, either implicitly or explicitely; Terraform, when applying a plan, sequentially follows dependencies to create and configure resources, but operates in parallel on resources without dependencies.

Terraform accesses provider resources through plugins, in this way multiple resource providers, even with different APIs, can be used to define and create complex infrastructures.

Terraform will be used by ADAPT as its *ADAPT DO Execution Engine* component (see section 4.1). In particular the *Verifyer* corresponds to the `terraform plan` command and the *Executor* to the `terraform apply` command.

An ADAPT deployment plan will be implemented as a (set of) Terraform configuration file(s), which is a text file ending in `.tf`. The ADAPT **DO Preparation Engine** component will be able to generate Terraform configuration files starting from information in the Application Description.

6.2 Consul

Consul [16] from HashiCorp is an open source tool to simplify the connection and configuration of software components. It includes functionalities such as service discovery, health checking and key-value store. The Community version of Consul is licensed under the Mozilla Public License 2.0; two Enterprise versions (Pro and Premium) are also available with additional features and support.

ADAPT will use Consul to provide the optional Service Registry functionality for applications. The health check functionality of Consul avoids routing requests to unhealthy services and could also be used for monitoring the status of deployed (micro)services. Consul key-value store functionality can be used by applications to store their configuration and also by the ADAPT to remotely store the deployment state (supports state locking during writing to avoid corruption).

[5] See https://www.terraform.io/docs/backends/index.html

6.3 Traefik

Traefik [17] is an open source HTTP reverse proxy and load balancer, specifically built for microservices. Traefik handles its configuration dynamically and automatically: it connects to a service registry API to know when microservices are added, relocated, or removed, and automatically generates / updates its configuration. Traefik is licensed under the MIT License. Commercial support is also available from Containous.

ADAPT will use Traefik to provide the optional Service Proxy functionality for applications. It could also be used as a load balancer in production settings with multiple ADAPT instances.

6.4 Nagios

Nagios is a monitoring platform developed by Nagios Enterprises[6] . Nagios solution has two versions Nagios Core and Nagios XI. Nagios Core is a free open source solution (GNU-General Public License) that can be obtained and used freely. Nagios XI is a commercial product with a yearly cost that can go from 1500 to 6000 dollars depending on the version and the number of nodes[7]. The differences between the two versions are explained in the Nagios site[8].

For the purpose of the project we will keep with the evaluation of Nagios Core, in order to improve the usage and reuse of the resulting platform and outcomes for evaluation purposes. Later on the evaluating partners may decide to continue with commercial support based on their needs.

Nagios is the "de facto standard" of monitoring in the industry. It is the oldest one, and it is the one that shows more relevance if we look at google trends. It is the monitoring technology that every monitoring alternative compares with.

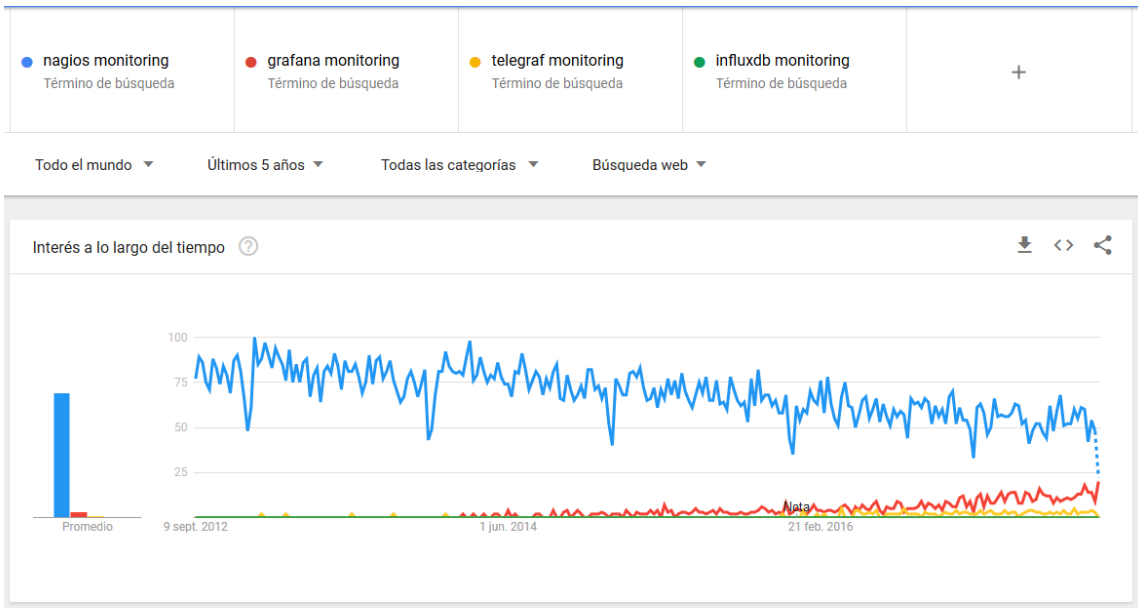


Figure 13. Interest of the different monitoring tools, Nagios, Grafana, Telegraf, Influxdb.

[6] https://www.nagios.org/
[7] https://assets.nagios.com/handouts/nagiosxi/Nagios-XI-Pricing-Documentation.pdf
[8] https://www.nagios.org/downloads/nagios-core/

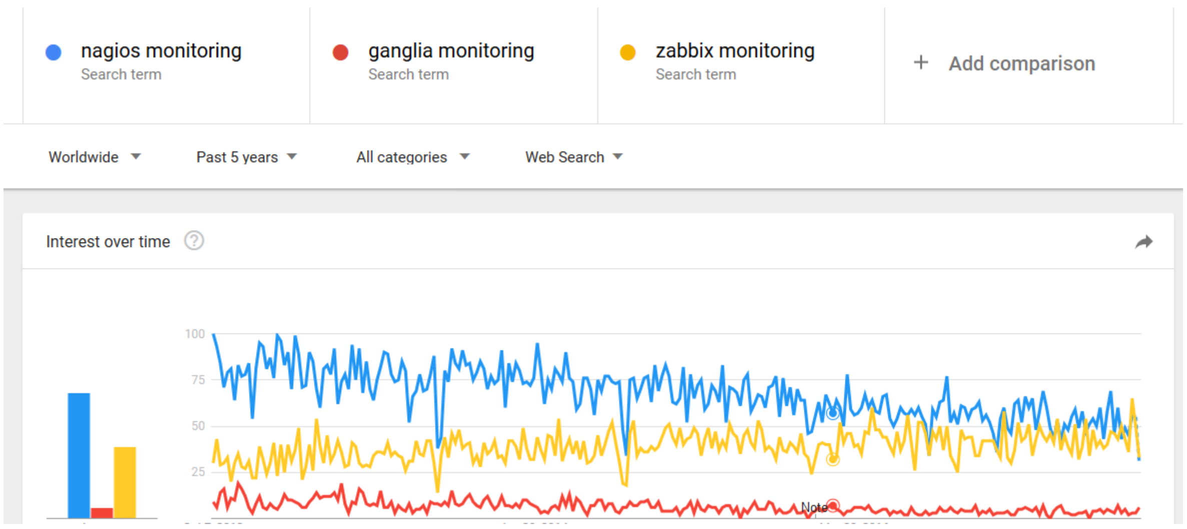


Figure 14. Interest of the different monitoring tools, Nagios, ganglia

Focusing on the typical features covered by the monitoring platform and arranging them in a "monitoring stack" we can see that Nagios Core covers: checks, Storage, Aggregation, View and alerting.

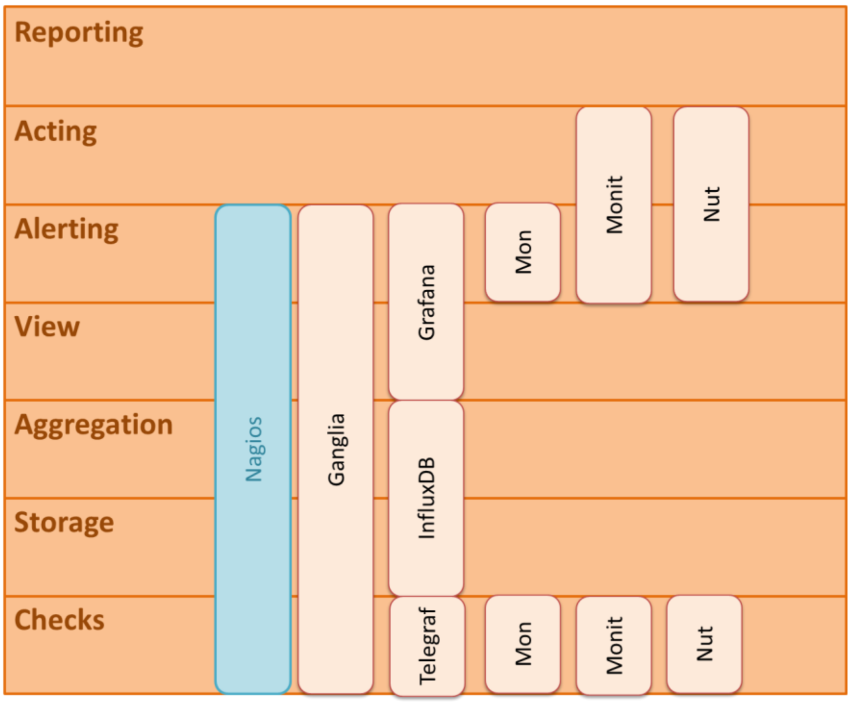


Figure 15. Nagios monitoring stack coverage

The usage of this technology can be applied to the monitoring of the components or cloud resources managed by ADAPT.

6.5 Telegraf, Influx DB, Grafana

Telgraf, Influxdb and Grafana are different monitoring technologies that can be combined to build a monitoring platform. Telegraf and influxDB are developed by InfluxData, Inc[9]. While grafana is

[9] https://www.influxdata.com/

developed by grafana labs[10]. Telegraf and influxDB are provided as open source (MIT license)[11] covering the basic funtionality, but additionally influxData offers a commercial offer for covering additional enterprise needs. The difference between the open source and the commercial offers are explained in influxData site[12]. The pricing of the commercial version can go from 2000 to 33000 dollars depending on the needs[13]. Grafana is also provided as open source (Apache license)[14] covering the basic functionality, but additionally grafana labs provide a hosted grafana service managed by them. The hosted grafana cost varies from 250 to 7700 dollars depending on the number of servers[15].

For the purpose of the project we will keep with the basic features of telegraf, influxdb and grafana, in order to improve the usage and reuse of the resulting platform and outcomes for evaluation purposes. Later on the evaluating companies may decide to continue with commercial support based on their needs.

Telegraf and influxDB are also shipped together with other two products of influxData: chronograf and Kapacitor. This configuration is known as TICK. In our case, we will replace the graphical representation (chronograf) and the alerting (Kapacitor) by grafana, which is also very common in the state of the practice. The selection of grafana over chronograf and kapacitor is based on our preferences with respect to the graphical representation of the measures. Therefore we will use telegraf to gather the metrics, influxDB to gather and aggregate them, and grafana to represent and alert.

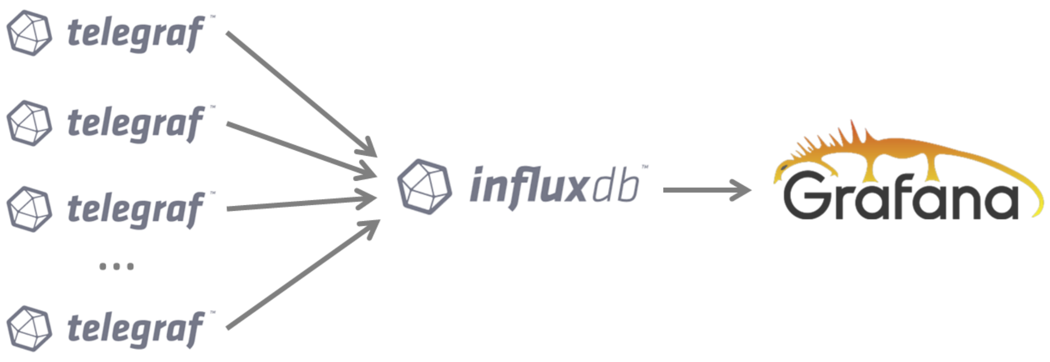
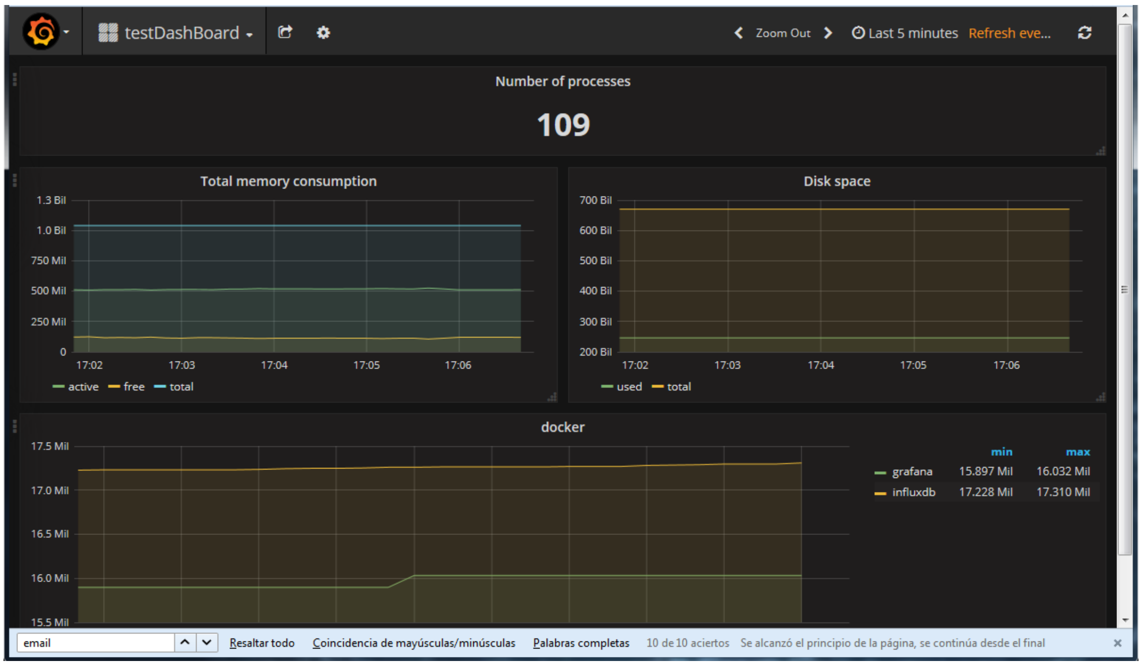


Figure 16. The monitoring stack: Telegraf,Iinfluxdb and Grafana.

Telegraf, influxdb and grafana are more at the edge of the state of the art. They apply latest technologies for lightweight metrics gathering, data series storage and graphical user interfaces. Many examples of graphical user interface configuration may be found in grafana site.

[10] https://grafana.com/
[11] https://en.wikipedia.org/wiki/MIT_License
[12] https://www.influxdata.com/products/editions/
[13] https://www.influxdata.com/influxcloud-pricing/
[14] https://www.apache.org/licenses/LICENSE-2.0
[15] https://grafana.com/cloud/metrics



	min	max
grafana	15.897 Mil	16.032 Mil
influxdb	17.228 Mil	17.310 Mil

Figure 17. Grafana graphical interface[16].

Focusing on the typical features covered by the monitoring platform and arranging them in a "monitoring stack" we can see that Telegraf, influxdb and grafana together covers: checks, Storage, Aggregation, View and alerting.

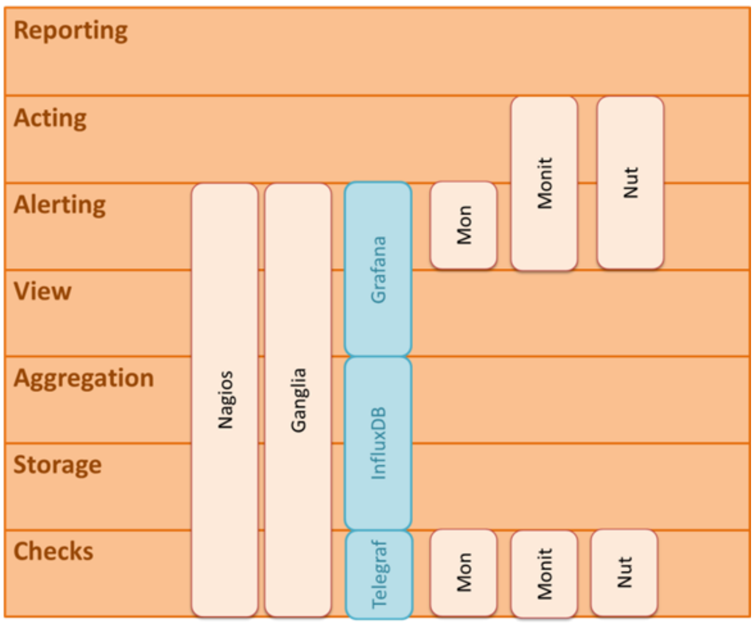


Figure 18. Telegraf+InfluxDB+Grafana monitoring stack coverage.

The usage of this technology can be applied to the monitoring of the components or cloud resources managed by ADAPT.

[16] https://grafana.com/dashboards

6.6 Ganglia

Ganglia is a monitoring tool for high-performance computing systems. Ganglia is an Open Source project[17].

Ganglia provides a complete, real-time monitoring and execution environment based on a hierarchical design. It uses a multicast listen/announce protocol to monitor node status, and uses a tree of point-to-point connections to coordinate clusters of clusters and aggregate their state information. Ganglia uses the eXtensible Markup Language (XML) to represent data, eXternal Data Representation (XDR) for compact binary data transfers, and an open source package called RRDTool for data storage (in Round Robin databases) and for graphical visualization.

It runs on Linux, Solaris, FreeBSD, AIX, IRIX, Tru64, HPUX, Mac OS X and Windows (using *cygwin*). Ganglia's in use on more than 500 clusters around the world. Moreover, PlanetLab has used Ganglia to monitor clusters at over 100 sites in half a dozen countries on a single Linux box.

The Ganglia Monitoring Core consists of the monitoring daemon,"*gmond",* which runs on every node; the meta daemon, "*gmetad",* which runs on a central machine, and collects and stores state information; a command-line status client called "*gstat*" which connects to a monitoring daemon and outputs a load-balanced list of cluster nodes; and a command-line tool called "*gmetric*" that defines new metrics for the monitoring daemons to track.

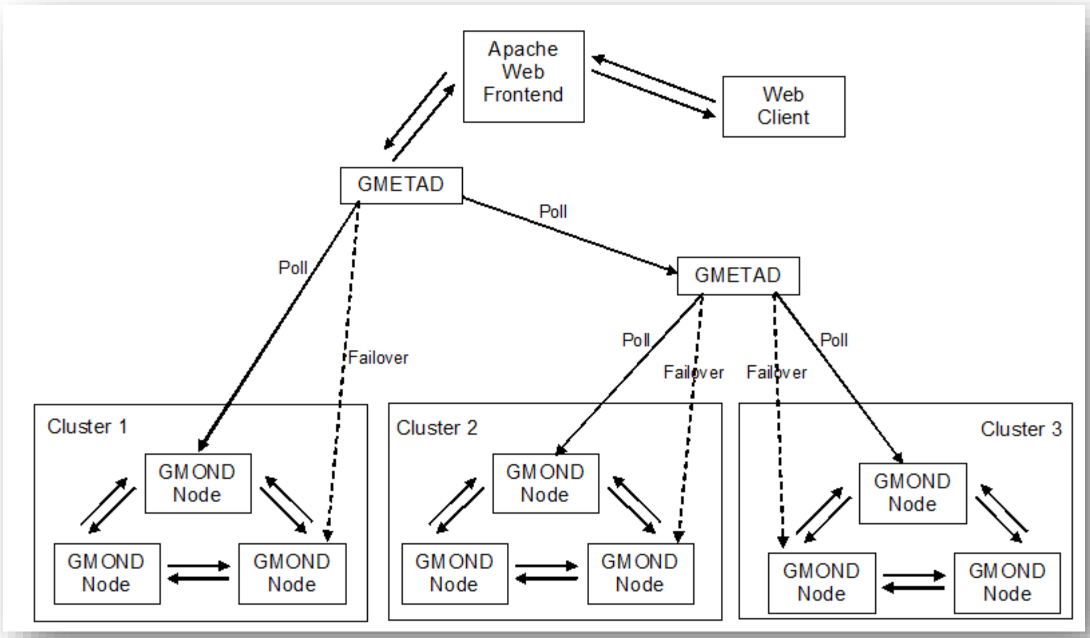


Figure 19. Ganglia architecture[18]

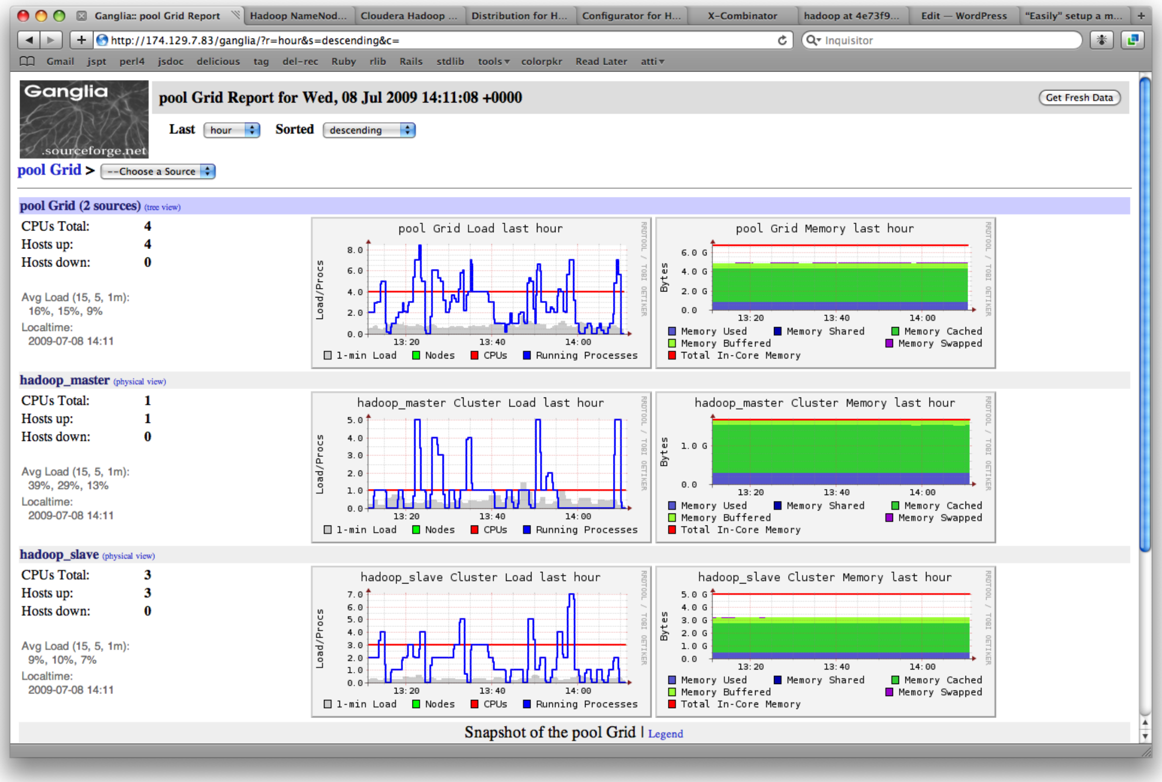
In addition, a command-line tool called "*gexec*" is available. It is useful for starting parallel or distributed jobs on a cluster or on the computational grid. A web front-end for displaying real-time statistics and graphics from the meta daemon is also available. Finally, a Python class is available for sorting and classifying large clusters using the monitoring core.

[17] available on SourceForge at http://ganglia.sourceforge.net) with a BSD license. It grew out of the University of California, Berkeley, Millennium Cluster Project (seehttp://www.millennium.berkeley.edu)
[18] From: https://www.slideshare.net/sudhirpg/ganglia-monitoring-tool (slide 5)

- **Ganglia Monitoring Daemon (gmond)** is a daemon which needs to sit on every single node which needs to be monitored, gather monitoring statistics, send as well as receive the stats to and from within the same multicast or unicast channel
    - If it is a sender (mute=no), it will collect basic metrics such as System Load (load_one), CPU Utilization. It can also send user defined metrics through addition of C/Python modules.
    - If it is a receiver (deaf=no), it will aggregate all metrics sent to it from other hosts. It will keep an in memory cache of all metrics
- **Ganglia Meta Daemon (gmetad)** is a daemon that polls gmonds periodically and stores their metrics into a storage engine like RRD. It can poll multiple clusters and aggregate the metrics. It is also used by the web frontend in generating the UI.
- **Ganglia PHP Web Front-end.** It should sit on the same machine as gmetad as it needs access to the RRD files. The Ganglia web front-end provides a view of collected data through real-time dynamic web pages. It provides up to three levels of data displays: one for the grid (or multi-cluster view), one for each cluster (physical view), and one for each host or node. Utilization data can be viewed over the past hour, day, week, month, or year. This makes it easy for both system administrators and grid/cluster users to quickly see the status of the resources and their trends through time.
  All the information and graphics are generated on-the-fly by parsing a complete Ganglia XML tree — obtained by contacting the local gmetad — for every page accessed. Therefore, the front-end should run on a fairly powerful computer; if a large grid is monitored this way, a dedicated server may be best. The web front-end is written in the PHP scripting language, and it works well under the Apache web server with the PHP 4.1 module.

By default gmond communicates on UDP port 8649 (specified in udp_send_channel and udp_recv_channel) and gmetad downloads metrics data over TCP port 8649 (specified as tcp_accept_channel). Any rules that block traffic on those ports will avoid metrics to show up. The firewall has to be open also for HTTP connections coming from the Ganglia server to the nodes of the monitored system. This is due to the monitoring system Ganglia provides that, as it is explained in short, is based on HTTP connections.

Every node in the system is monitored through the web-console. Apart from the default parameters that are monitored (usage of CPU, memory, disk, input/output streams…); the operator can define his own monitoring functions. This is done through a default functionality, which allows defining standardized queries (HTTP, FTP, IMAP, MySQL, etc.) to IPs or URLs.



pool Grid Report for Wed, 08 Jul 2009 14:11:08 +0000
Last hour Sorted descending
pool Grid (2 sources)
CPUs Total: 4
Hosts up: 4
Hosts down: 0
Avg Load (15, 5, 1m): 16%, 15%, 9%
Localtime: 2009-07-08 14:11
hadoop_master
CPUs Total: 1
Hosts up: 1
Hosts down: 0
Avg Load (15, 5, 1m): 39%, 29%, 13%
Localtime: 2009-07-08 14:11
hadoop_slave
CPUs Total: 3
Hosts up: 3
Hosts down: 0
Avg Load (15, 5, 1m): 9%, 10%, 7%
Localtime: 2009-07-08 14:11
Snapshot of the pool Grid | Legend

Figure 20. A cluster monitoring using Ganglia[19]

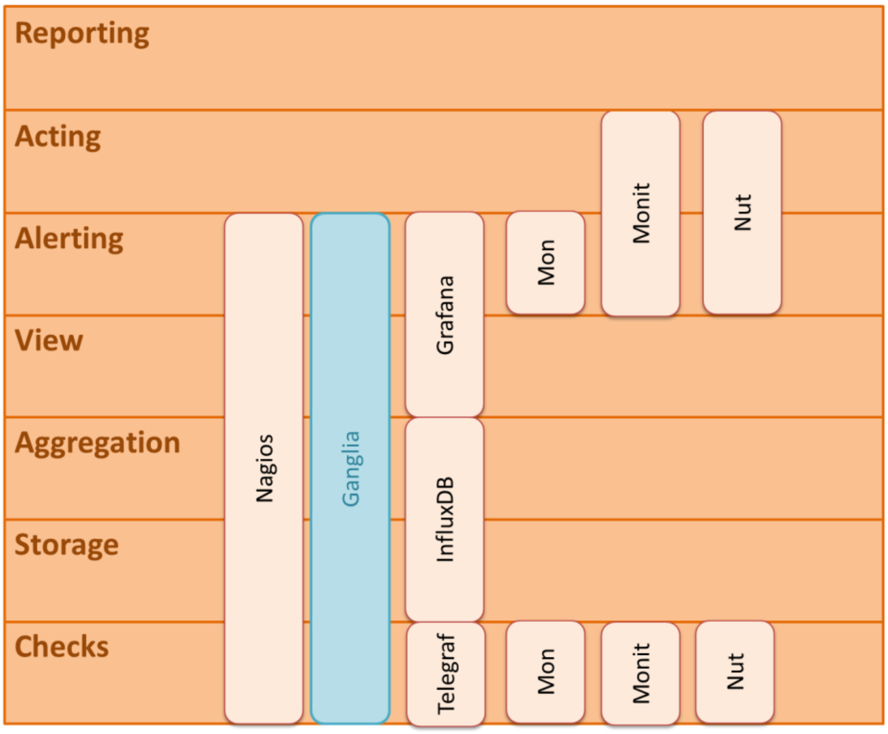


Figure 21. Ganglia monitoring stack coverage

[19] http://eigenjoy.com/2009/07/08/easily-setup-a-monitored-hadoop-hive-cluster-in-ec2-with-poolparty/

The usage of this technology can be applied to the monitoring of the components or cloud resources managed by ADAPT.

7 ADAPT deployment alternatives

Several possible answers exist to the research question “How to deploy the deployment tool?”. Some of those answers have been analyzed in the first year of the DECIDE project, both in the context of normal WP4 activities and during General Technical Meetings.

A final implementation decision has not yet been reached as related experiments are still ongoing; a decision will be taken by month 24 of the DECIDE project (M24). This section summarizes some of the options discussed so far.

7.1 The basic idea: containerized microservices

The initial idea for ADAPT implementation is to follow the same approach proposed by DECIDE to the developers for their multi-cloud applications: containerized microservices.

The ADAPT Deployment Orchestrator will be implemented as a microservice providing the iDeploy interface (see section 4.1) and deployed as a Docker container. A separate microservice/container will host the deployment state storage system.

Also the Monitoring Manager can be implemented with a central containerized microservice collecting data from agents tied to application’s components. The monitoring agents will be either already present in eachcomponent’s container image or installed as part of the deployment. The monitoring DB will be deployed as a separated microservice/container.

More details about the implemented ADAPT microservices can be found in deliverables D4.4 (for deployment) and D4.7 (for monitoring).

7.2 One ADAPT for each application

The set of microservices explained in the previous section are the constituents of ADAPT, but how those microservices will be deployed with respect to each application? The current idea is to have one instance of ADAPT for each application.

ADAPT can be the last ring in the chain of the application’s Continuous Integration pipeline: the DevOps environment builds the application’s microservices and then ADAPT is activated to deploy them. The monitoring components of ADAPT are deployed along with the application and are dedicated to keep its QoS under control. When the application is redeployed to cope with violations the monitoring components can be redeployed as well, but their monitoring history is maintained.

This kind of deployment will be the first to be tested and analyzed; results of this analysis will be reported in the next WP4 deliverables.

7.3 ADAPT as a Service

Another possibility to be analyzed is to deploy ADAPT as a service. Other DECIDE components, for example ACSmI, could be deployed as services: maybe the whole DECIDE environment could be offered in this way. In the case of ADAPT customers might pay for a service which deploys their application on multiple clouds, continuously monitors it and is able to adapt it to possibly changing external conditions.

A more detailed business analysis of this kind of scenario has been performed in DECIDE WP7.

8 Conclusions

This deliverable has described ADAPT's high-level architecture, as well as candidate technologies that have been investigated for its implementation. The requirements gathered amongst partners have been used to elicit a series of high-level functionalities. These functionalities have been grouped, resulting in three main blocks that compose ADAPT's architecture: deployment component, monitoring component and violation handler.

The deployment component takes care of tasks strictly related to deploying the application into cloud providers, the monitoring component monitors application metrics to make sure that they fulfill their SLAs and generate alerts if they do not, and the violation handler processes these alerts and triggers the corresponding actions.

The information exchanges have also been analyzed to define interfaces for each component, and the interactions with other tools of the DECIDE framework.

Furthermore, different technologies have been explored to be used as a starting point for some of the components, identifying some interesting options such as Telegraf, Influx and Grafana for the monitoring part of ADAPT.

Lastly, the document analyzes deployment possibilities for ADAPT. In general it has been considered to deploy ADAPT as a series of microservices, following the philosophy of the project, but specific options, such as using one ADAPT for each application or deploying ADAPT as a service have also been identified, although a final decision has yet to be reached. Analysis of the various deployment alternatives will be one of the major topics of next year's work for WP4, along with integration and improvement of the developed components.

References

[1] DECIDE-WP4-Partners, “DECIDE WP4 Requirements,” [Online]. Available: https://docs.google.com/spreadsheets/d/1ZopEEXmxXe_Rqr5xnZihh5bBjprXKJju0iwsAkfX1UE/edit#gid=0. [Accessed 27 July 2017].

[2] DECIDE, "Deliverable D2.1 - Detailed Requirements Specification," 2017.

[3] Pivotal, “Cloud-Native Apps,” [Online]. Available: https://pivotal.io/cloud-native. [Accessed 09 2017].

[4] Google, “GitHub for Protobuf,” [Online]. Available: https://github.com/google/protobuf/. [Accessed 09 2017].

[5] SmartBear.com, “Understanding SOAP and REST Basics and Differences,” [Online]. Available: https://blog.smartbear.com/apis/understanding-soap-and-rest-basics/?_ga=2.92057180.770716268.1504774943-915349788.1504774943. [Accessed 09 2017].

[6] L. Mergen, “www.leonmergen.com,” [Online]. Available: https://leonmergen.com/on-stateless-software-design-what-is-state-72b45b023ba2. [Accessed 09 2017].

[7] J. C. Mitchel, in *Concepts in Programming Languages*, Cambridge University Press, 2002, p. 78.

[8] Fasthosts, 1&1, “Why containers? Development through the ages,” in *TEC Day 2017*, Gloucester, 2017.

[9] docker, “Swarm mode key concepts,” [Online]. Available: https://docs.docker.com/engine/swarm/key-concepts/. [Accessed 09 2017].

[10] kubernetes, “kubernetes,” [Online]. Available: https://kubernetes.io/. [Accessed 09 2017].

[11] Red Hat, “OpenShift,” [Online]. Available: https://www.openshift.com/. [Accessed 09 2017].

[12] Docker, “The Docker Way,” [Online]. Available: https://github.com/cozy/cozy-setup/wiki/2.4.-The-Docker-Way. [Accessed 09 2017].

[13] kubernetes, “Federation,” [Online]. Available: https://kubernetes.io/docs/concepts/cluster-administration/federation/. [Accessed 09 2017].

[14] M. Fowler, "BlueGreenDeployment," [Online]. Available: https://martinfowler.com/bliki/BlueGreenDeployment.html.

[15] HashiCorp, "Terraform home page," [Online]. Available: https://www.terraform.io/. [Accessed September 2017].

[16] HashiCorp, "Consul home page," [Online]. Available: https://www.consul.io/. [Accessed September 2017].

[17] Containous, "Traefik home page," [Online]. Available: https://traefik.io/. [Accessed September 2017].

APPENDIX: Application Description

The following table describes the current version of the Application Description for both deployment and monitoring, resulting from Partners' research and discussions, with a brief description for each field. The same Application Description definition can be found in DECIDE deliverable D2.4; here the column Used By indicates if the field is needed by the ADAPT Deployment (DEP) or ADAPT Monitoring (MON). Please note that the Application Description definition has evolved since the first data model shown in the DECIDE deliverable D2.1, and it's still evolving in parallel with the design and implementation iterations.

Table 9. Application Description model for deployment

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component	Used By
id			String	1	Unique identifier for the Application Description	Creation Wizard	
name			String	1	Name of the application	Creation Wizard	DEP
description			String	1	Textual description of the application		
highTechnologicalRisk			Boolean	1	Indicates if the application has high technological risk: confirmation for (re)deployment is needed		DEP
version			String	1	Indicates the version number of the app description "schema", for compatibility purposes		
microservices			Array of Objects	1..*	List of microservices	Creation Wizard	
	id		String	1	Unique Identifier for the microservice	Creation Wizard	
	repo		String	1	Reference to location of microservice repo	Creation Wizard	
	name		String	1	Human readable name for the microservice	Creation Wizard	
	programmingLanguage		String	1	Type of programming language used for microservice (hint)	Creation Wizard	

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component	Used By
	container_ref		String	1	Id or URI of container in which the microservice is located for ADAPT to be able to deploy it	Creation Wizard	
	endpoints		Array of Objects (URI)	1..*	List of URI to access the services and their methods[20]	Creation Wizard	
	stateful		Boolean	1	Is the microservice stateful or stateless?	Creation Wizard	
	type		String	1	The type of the microservice, is it a data component or user interface component , etc.	Creation Wizard	
	patterns		Array of Objects	0..*	List of patterns applied to the microservice	ARCHITECT	
	dependencies		Array of Strings	0..*	List of microservice names the current one depends on	Creation Wizard	
	nfrs		Array of Strings	1..*	List of selected NFRs per microservice	NFR Editor	
	publicIP		Boolean	1	True if the microservice has a public IP address	OPTIMUS Classification	
	infrastructure_requirements		Object	1	Requirements for the infrastructure hosting the microservice	OPTIMUS Classification	
		disk_min	String	1		OPTIMUS Classification	
		disk_max	String	1		OPTIMUS Classification	
		RAM_min	String	1		OPTIMUS Classification	
		RAM_max	String	1		OPTIMUS Classification	
app_mcsla			Object	1	The MCSLA for the application (see **Table *10***)	MCSLA Editor	MON

[20] Port numbers in each URI are those exposed by the microservice, the container can be configured to map them to a different port number

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component	Used By
projectNfrs			Array of Strings	1..*	List of selected NFRs for the application, might apply to individual NFRs	NFR Editor	
virtual_machines			Array of Objects	0..*	Description of the VMs that will host the containers		DEP
	id			1			DEP
	csp_name		String	1	Name of the CSP providing this VM	OPTIMUS	DEP
	csp_id		String	1	Internal UUID for the CSP providing this VM	OPTIMUS	DEP
	RAM		String	1	Amount of memory (in GB)	OPTIMUS	DEP
	cores		Integer	1	Number of cores	OPTIMUS	DEP
	storage		String	1	Amount of disk space (in GB)	OPTIMUS	DEP
	image		String	1	Name of the VM image (identifies also the OS and its version)	OPTIMUS	DEP
	cloudbrokerEndpoint		String	1	Endpoint of the Cloudbroker (ACSmI) API, to which all the cloud resource provisioning requests are sent	ACSmI	DEP
	cloudbrokerUsername		String	1	Username for the Cloudbroker API access	ACSmI	DEP
	cloudbrokerPassword		String	1	Password for the Cloudbroker API access	ACSmI	DEP
	vmSoftwareId		String	1	Id of the software resource from the Cloudbroker catalog dictionary. Represents the OS and version of the VM (e.g. “Ubuntu 16.04”)	ACSmI/Optimus	DEP
	vmResourceId		String	1	The id of the Cloudbroker VM resource, which represents the underlying CSP that will perform the real provisioning	ACSmI/Optimus	DEP
	vmRegionId		String	1	The id of the “Region” where the VM will run, taken from the Cloudbroker catalog dictionary (E.g.: Zrh, US Standard, …)	ACSmI/Optimus	DEP

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component	Used By
	instanceTypeId		String	1	The id of the "instanceType" which represents the combination of resources allocated to the vm (e.g. "2 Total cores, 2GB RAM)	ACSmI/Optimu s	DEP
	keyPairId		String	1	The id of the keypairs needed to access Cloudbroker resources (associated to the Cloudbroker user profile)		DEP
	openedPorts		String	0..1	The comma separated list of ports to be open on the VM	Developer	DEP
	consulJoinIp		String	1	Address of the master Consul (service registry) node; if "self", it means that this VM will act as master	TBD: it will be the address of a node running ADAPT	DEP
	dockerPrivateRe gistryIp		String	0..1	IP of a Docker private registry, which will host custom container image prepared by a developer that are not published to the public Docker Hub repository	Developer	DEP
	dockerPrivateRe gistryPort		Integer	0..1	Port of the private Docker registry	Developer	DEP
	dockerHostNode Name		String	1	Name of the Docker node (referenced by the same field in each container definition)	Developer / OPTIMUS	DEP
containers			Array of Objects	1..*	Description of the containers that will host the microservices		DEP
	Id		String	1	Id of the container		
	containerName		String	1	Name of the container	Developer	DEP
	imageName		String	1	Name of the container image	Developer	DEP
	imageTag		String	1	Tag to identify the container in the registry	Developer	DEP
	dockerPrivateRe gistryIp		String	0..1	IP of a Docker private registry, which will host custom container image prepared by a	Developer	DEP

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component	Used By
					developer that are not published to the public Docker Hub repository		
	dockerPrivateRegistryPort		String	0..1	Port of the private Docker registry	Developer	DEP
	dockerPrivateRegistryUser		String	0..1	Username to access the private Docker registry	Developer	DEP
	dockerPrivateRegistryPassword		String	0..1	Password to access the private Docker registry	Developer	DEP
	hostname		String	1	Hostname of the container	Developer	DEP
	restart		String	1	Attribute indicating the restart policy for this container (e.g. “always”)	Developer	DEP
	command		Array of Strings	0..*	Comma-separated list of commands to be passed to the container, as for the “CMD” Dockerfile specs	Developer	DEP
	entrypoint		Array of Strings	0..*	Comma-separated list of commands and parameter to be passed to the container, as for the “ENTRYPOINT” Dockerfile specs	Developer	DEP
	DockerHostNodeName		String	1	Name of the VM hosting the container	OPTIMUS	DEP
	networks		Array of Strings	0..*	This field will trigger the creation of one or more dedicated Docker network(s) on the container to allow two containers to see each other in case it does not exist	Developer / OPTIMUS	DEP
	volumeMapping		Array of Objects	0..*	Mapping of volumes from host paths to container paths	Developer	DEP
		hostPath	String	1	Path on the host	Developer	DEP
		containerPath	String	1	Path on the container	Developer	DEP
	environment		Array of Strings	0..*	List of comma-separated KEY=VALUE environment variables to be defined before	Developer	DEP

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component	Used By
					starting the container, as for the “ENV” Dockerfile specs		
	consulKvProviderNodeName		String	1	Name of the node hosting the Consul Key-Value provider	(TBD: it will be the node running ADAPT)	DEP
	addConsulService		Boolean (0|1)	0..1	Specify whether to register the service to a Consul service registry (enables basic health-check)	(TBD: it may be enabled by default)	DEP
	addConsulTraefikRules		Boolean(0|1)		Specify whether to add reverse proxy routing rules to the Consul K/V store (based on “Host:” header)	Developer	DEP
	portMapping		Array of Objects	0..*	List of ports to be published by this container	Developer	DEP
		hostPort	String	1	Port to be exposed on the host	Developer	DEP
		containerPort	String	1	Port exported by the container	Developer	DEP
	endpoints		Array of Objects	0..*	List of endpoints for this container	Developer	DEP
		protocol	String	1	Typically, “http”, but can be something else according to URL syntax	Developer	DEP
		port	Integer	1	The port to which the endpoint is bound	Developer	DEP
		skipRule	Boolean (0|1)	0..1	Set to 1 to discard the routing rule based on hostname (“Host:” header)	Developer	DEP
		containerNameOverride	String	0..1	Overrides the standard routing rule based on hostname; hence, it allows to consider a different hostname for this service	Developer	DEP

The following tables describe the Application Description model for monitoring with a brief description for each field. **Table *10*** describes the nested elements for the field app_mcsla of the Application Description. The MCSLA Editor is responsible for eliciting this information from the user.

Table 10. Application Description model for monitoring the application via its MCSLA (nested elements for "app_mcsla")

Element Name	**app_mcsla**		
Description	General information about the MCSLA		
attribute -or- Element	**Type**	**Multiplicity / Default**	**Definition**
id	String	1	Unique Identifier for the MCSLA
description	String	1	This is MCSLA description line.
visibility	String	1	public or private
validityPeriod	Integer	1	The validity period of the MCSLA in days
microservice_SLAs	Microservice_SLAs	1..*	The list of SLAs for each microservice

The following **Table *11*** describes the fields nested in the microservice_SLAs field of the MCSLA.

Table 11. Nested elements for microservice_SLAs

Element Name	**Microservice_SLAs**		
Description	The general information about the SLAs for each microservice		
attribute -or- Element	**Type**	**Multiplicity / Default**	**Definition**
id	String	1	Unique Identifier for the microservice_SLA
ms_id	String	1	Unique Identifier of the microservice this SLA belongs to
csp_id	String	1	Unique Identifier of the CSP from which the SLA comes from
visibility	String	1	public or private
validityPeriod	Integer	1	The validity period of the SLA in days, should not be higher than that of the MCSLA
microservice_SLO	microservice_SLO	1..*	List of microservice SLOs
microservice_SQO	microservice_SQO	1..*	List of microservice SQOs

The following **Table *12*** describes the fields nested in the microservice_SLO and microservice_SQO fields of microservice_SLAs.

Table 12. Nested elements for microservice_SLO and microservice_SQO

Element Name	**microservice_SLO and microservice_SQO**		
Description	The general information about the slo or sqo defined for a microservice		
attribute -or- Element	**Type**	**Multiplicity / Default**	**Definition**
id	String	1	Unique Identifier for the microservice_SLA
termName	String	1	Name of the term to which it refers to
value	Integer	1	Term value that should not be violated based on calculation formula
unit	String	1	Term unit
comparisonOperator	String	1	Comparison operator for monitoring the SLO
violationTriggerRule	ViolationTriggerRule	1	The violation Trigger Rule
remedy	Remedy	0..1	the compensation available to the cloud service customer in the event the cloud service provider fails to meet a specified cloud service level objective
metrics	Metrics	1..*	The definition of how to measure the SLO or SLA
violation_report	String	0..1	Indicates where to report violations for this application (optional)

The following **Table *13*** describes the fields nested in the violationTriggerRule field of microservice_SLO and microservice_SQO.

Table 13. Nested elements for ViolationTriggerRule

Element Name	**ViolationTriggerRule**		
Description	The general information about the violation trigger rule		
attribute -or- Element	**Type**	**Multiplicity / Default**	**Definition**
interval	string	1	Indicates the monitoring frequency for each SLO
breaches_count	Integer	1	The count of how many breaches have taken place

The following **Table *14*** describes the fields nested in the remedy field of microservice_SLO and microservice_SQO.

Table 14. Nested elements for Remedy

Element Name	**Remedy**		
Description	The general information about the compensation available to the cloud service customer in the event the cloud service provider fails to meet a specified cloud service level objective		
attribute -or- Element	**Type**	**Multiplicity / Default**	**Definition**
type	String	1	The type of remedy the cloud service provider will be offering the cloud service customer
value	Integer	1	The value of the type of remedy offered by the cloud service provider
Unit	String	1	The unit for the value offered
validity	Integer	1	The validity period for this remedy

The following table holds the fields (taken directly from ISO 19086-2 Metric Model.) that are nested within the metrics field of microservice_SLO and microservice_SQO. The MCSLA Editor is responsible for eliciting this information from the user.

Table 15. MCSLA Metric data model for monitoring

Element Name	**Metric**		
Description	The general information about the metric		
attribute -or- Element	**Type**	**Multiplicity / Default**	**Definition**
descriptor	String	0..1	a short description of the metric
Id	String	1	a unique identifier for the metric within a context
source	String	1	the individual or organization who created the metric
scale	enumeratedList	1	classification of the type of measurement result when using the metric. The value of scale shall be "nominal, ordinal, interval, or ratio". SLOs shall use either the "interval" or "ratio" scale. SQOs shall use the "nominal" or "ordinal" scales.
note	String	0..1	additional information about the metric and how to use it.
category	String	0..1	a grouping of metrics with similar expressions, rules, and parameters
expression	Expression	0..1	The expression of the calculation of the Metric and supporting information. An SLO metric shall have an expression while an SQO may or may not have an expression (e.g., specified using natural language). It shall be written using the ids to represent UnderlyingMetrics, Parameters, and Rules.
parameters	Parameter	0..*	a Parameter is used to define a constant (at runtime) needed in the expression of an Metric. A Parameter may be used by more than one Metric if it is defined using a unique ID within the set of metrics it is used in.
rules	Rule	0..*	a Rule is used to constrain a Metric and indicate possible method(s) for measurement.
underlyingMetrics	Metric	0..*	a metric element that is used within an expression element to define a variable. The Expression shall use the Underlying Metric id to reference the Underlying metric within the expression.

The following **Table *16*** describes the fields nested in the expression field of a Metric.

Table 16. Nested elements for Expression

Element Name	**Expression**		
Description	The expression of the calculation of the **Metric** and supporting information		
attribute -or- Element	**Type**	**Multiplicity / Default**	**Definition**
Id	String	1	a unique identifier (within the context of the metric) for the expression
expression	String	1	the expression statement written using the ids to represent UnderlyingMetrics, parameters, and rules.
expressionLanguage	String	1	the language used to express the metric (i.e. ISO80000)
note	String	0..1	additional information about the expression
unit	String	0..1 required when scale is ratio or interval	real scalar quantity, defined and adopted by convention, with which any other quantity of the same kind can be compared to express the ratio of the two quantities as a number.
subExpression	Expression	0..*	an associated element of type element that is used within the expression to define a variable. The Expression shall use the SubExpression id to reference the SubExpression within the expression.

The following **Table *17*** describes the fields nested in the parameters field of a Metric.

Table 17. Nested elements for Parameter

Element Name	**Parameter**		
Description	A Parameter is used to define a constant (at runtime) needed in the expression of a Metric. A Parameter may be used by more than one Metric if it is defined using a unique ID within the set of metrics it is used in.		
attribute -or- Element	**Type**	**Multiplicity / Default**	**Definition**
id	String	1	the unique identifier of the parameter
parameterStatement	String	1	the statement or value of the parameter
unit	String	1	the unit of the parameter
note	String	0..1	additional information about the parameter

The following **Table *18*** describes the fields nested in the rules field of a Metric.

Table 18. Nested elements for Rule

Element Name	**Rule**		
Description	A Rule is used to constrain a Metric and indicate possible method(s) for measurement. For instance an “AvailabilityDuringBusinessHour” Metric could be defined with a scope that constrains some piece of a generic “Availability” Metric element that limits the measurement period to defined business hours. A Rule describes constraints on the metric expression. A constraint can be expressed in many different formats (e.g. plain English, ISO 80000, SBVR)		
attribute -or- Element	**Type**	**Multiplicity / Default**	**Definition**
Id	String	1	the unique identifier for the rule
ruleStatement	String	1	a constraint on the metric
ruleLanguage	String	1	the language used to express the rule in the ruleStatement
Note	String	0..1	additional information about the rule