Deliverable D2.1

Detailed Requirements Specification

Editor(s):	Javier Gavilanes
Responsible Partner:	Innovati
Status-Version:	Final – v1.0
Date:	31/05/2017
Distribution level (CO, PU):	CO

Project Number:	GA 731533
Project Title:	DECIDE

Title of Deliverable:	Requirement Specification
Due Date of Delivery to the EC:	31/05/2017

Workpackage responsible for the Deliverable:	WP2 – DECIDE requirements and DECIDE solution integration
Editor(s):	Innovati
Contributor(s):	Juncal Alonso (TECNALIA) Marisa Escalante (TECNALIA) María Jose López (TECNALIA) Lena Farid (FhG) Lorenzo Blasi (HPE) Javier Gavilanes (Innovati) Luis Miguel Silva (Innovati) Javier Álvarez (Innovati) Daniel Rodríguez (Innovati) Gema Maestro (Innovati) Antony Shimmin (AIMES)
Reviewer(s):	Nicola Fantini (CB)
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP4, WP5

Abstract:	This document will contain all the technical functional, non-functional and technical requirements of DECIDE DevOps Framework and all the components to be developed in the context of WP3, WP4 and WP5.
Keyword List:	Requirements, functionalities, framework, tools
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	29/01/2017	First draft version	Innovati
v0.2	28/02/2017	Corrections to the ToC	Innovati
v0.3	20/03/2017	First contributions added	Innovati
v0.4	25/04/2017	Second round of contributions and corrections	Innovati, Tecnalia
v0.5	27/04/2017	Annex extended	Tecnalia
v0.6	12/05/2017	Third round of contributions and corrections	All
v1.0	24/05/2017	Corrections after review	Innovati

Table of Contents

List of Figures

List of Tables

Terms and abbreviations

(MC)SLA	(MultiCloud) Service Level Agreement
ACSmI	Advanced Cloud Service (meta-) intermediator
API	Application Program Interface
CCDL	Common Development and Distribution License
CSLA	Composite Cloud Service Level Agreement
CSP	Cloud Service Provider
DoA	Description of the Action
EC	European Commission
EMS	Enterprise System Management
GPL	GNU General Public License
GUI	Graphical User Interface
HTTP(S)	Hypertext Transfer Protocol (Secure)
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IT	Information Technology
JVM	Java Virtual Machine
KPI	Key Performance Indicator
KR	Key Result
NFP	Non-functional Property
NFR	Non-functional Requirement
PaaS	Platform as a Service
QoS	Quality of Service
RCP	Rich Client Platform
REST	Representational State Transfer
SDK	Software Development Kit
SDLC	Systems Development Life Cycle
SLO	Service Level Objective
SOLC	Systems Operation Life Cycle
UI	User Interface
WP	Work Package
WTP	Web Tool Platform

Executive Summary

The purpose of the DECIDE framework (which corresponds to Key Result 1) is to integrate all the functionalities of the project's components to be validated in the three use-case scenarios defined in Work Package 6, and to support the DevOps principles. The main goal of this document is to gather the requirements that the framework must fulfil to support its functionalities, and the integration requirements of each component to integrate with it.

DevOps is a set of practices aimed at motivating collaboration, communication and integration between software developers and IT operators, while automating software delivery and infrastructure deployment. The framework developed within the DECIDE project must facilitate the application of the DevOps principles, which are distilled into a series of requirements, such as using a microservices architecture, support continuous integration, delivery and deployment, or monitoring metrics and logs.

As mentioned before, one of the goals of the framework is to integrate the different components. DECIDE is composed of several components or Key Results (KR): the framework itself (KR1), ARCHITECT (KR2), which provides a set of patterns; OPTIMUS (KR3), which evaluates the best deployment options; ACSmI (KR4), which monitors the fulfilment of the non-functional properties of the services; and ADAPT (KR5), which semi-automatically deploys and readapts the applications. All these components handle different types of data and, in some occasions, exchange data with other components. This deliverable analyzes the data needs of the KRs in order to achieve a proper integration.

1 Introduction

This deliverable aims at obtaining a list of requirements that the KR1 must fulfil, as well as the integration requirements for the different DECIDE components.

The document starts with a compilation of the key terms used in it, to provide a common understanding of those terms. The same section also includes the definition of the KPIs that will be monitored in the project. These KPIs are the mechanism that will be used to measure the level of accomplishment of the DECIDE.

After describing the main actors that will make use of the DECIDE framework, the document presents the requirements of the Key Result 1, which come from the objectives set out in the DoA, from an analysis of the State of the Art on DevOps, and must be validated by the use cases.

The next section contains a table with the high-level functionalities that DECIDE must support, which have being defined as a collaborative effort among the other WPs and will be used as a starting point to classify the requirements of each component.

Lastly, in the *Integration Analysis* section, the data exchanges among the different KRs and within each KR are studied. For every component a description and a data flow diagram are included, making a reference to the high-level functionalities that the KR supports. Then, the data exchanges are detailed and grouped by functionality.

In the appendix a study of the State of the Art on DevOps and an analysis of the different tools that will be integrated in KR1 can be found.

2 Definitions

This section contains the definition of key terms used throughout the document, to define a common language for the consortium and to ensure that every reader interprets those terms in the same way.

- **Multi-cloud.** Multi-cloud is the use of multiple computing services for the deployment of a single application or service across different cloud technologies and/or Cloud Service providers. This may consist of PaaS, IaaS and SaaS entities in order to deliver an end solution.

- **NFR**. NFR (Non Functional Requirement) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. In the context of DECIDE NFRs that the multi-cloud application should have are defined by the developer from a closed list of predefined NFRs. These NFRs and their metrics have being defined collaboratively and they will be implemented in the corresponding tools in different phases, according to the following table (v1 corresponds to M12, v2 to M24 and v3 to M30):

Table 1. DECIDE NFRs

Release	DECIDE NFR
v1	Availability
v1	Cost
v1	Technology risk
v1	State
v2	Security /Legal
v2	Location
v3	Performance
v3	Scalability

- **Availability**. Availability is the ratio of time a system or component is functional to the total time it is required or expected to function. This can be expressed as a direct proportion (for example, 999/1000 or 0.999) or as a percentage (for example, 99.999%).

  Availability in DECICE applies in two senses:

  1- From the CSP`s side: the Infrastructure Availability on which the particular Cloud/Multi-Cloud service is running.

  But it also concerns the Application as well.

  2- From the Application Side: for compliance the NFRs of an application that is running on those Cloud/Multi-Cloud Platforms, the MCSLAs are also included in the Availability.

  NOTE: A factor that must be taken into account in above definition is that most CSPs do not include maintenance downtime in their Availability Ratios.

- **Cost**. Total expenses related to the deployment and operation of a microservice in a certain CSP.

- **Technological risk**. The technological risk refers to the potential for losses due to technological differences of the cloud service providers where the components of the multi-

cloud application are deployed. In DECIDE application configuration can be classified as *low* technological risk if the baseline technology of the source and target cloud service do not need match (e.g. source is VMWare, target is OpenStack), in which case the application will be self-adaptive and able to be redeployed automatically, or as *high* technological risk, in which case the operator will be informed if malfunctioning occurs and the re-deployment will have to be performed in a manual way.

- **State.** State is a term related to the "memory" of a service. Services can be stateful or stateless. A stateful service stores data between different sessions, so previous actions can affect present ones, while a stateless service does not store data, every operation is performed as if it were being done for the first time.

- **Security.** Security is included in DECIDE as one of the NFR that the developer can set up when designing the multi-cloud application. In the context of DECIDE security will refer mainly to the security concerns derived from the location of the application. This will be considered mainly from the perspective of the GDPR and the implications of the location of the application and data in different countries/ sites.

  DECIDE security will be also analysed from the special needs of certain type of data (i.e. sensitive data) with respect to confidentiality (encryption/anonymization needs).

- **Location.** This term refers to the place where the application data is deployed. This is relevant for the project since different countries have different data management policies.

- **Performance.** Performance is a term that can be used in several contexts, although generally it refers to the supervision and measuring of relevant metrics with the purpose of evaluating resources' efficiency. Within DECIDE, it refers to the measuring of certain parameters to guarantee the fulfilment of the contracted services.

- **Scalability.** Scalability is a desirable characteristic in any system or process. It allows for the manipulation of increasing workloads or increasing data volume, uniformly and leaving performance unaffected. For DECIDE, scalability is the capacity it will have to let users increase data volume or number of accesses in a uniform way without performance or availability loss.

- **Reliability.** Reliability is an important characteristic for a storage or data transfer service, to guarantee that the data are not modified or lost on its path from origin to destination. For DECIDE, reliability is what allows the agents to ensure high availability of the services as well as data protection and no data loss.

- **Microservice.** Microservices architecture is an approach to application development in which a large application is built as a suite of modular services. Each module supports a specific business goal and uses a simple, well-defined interface to communicate with other sets of services.

  One of the characteristics of the microservices is the resilience. Rather than relying upon a single virtual or physical machine, components can be spread around multiple severs or even multiple data centers.

  The common definition of microservices generally relies upon each microservice providing an API endpoint, often but not always a stateless REST API which can be accessed over HTTP(S).

MCSLA. A multi-cloud composite service level agreement (MCSLA) is provided by the multi-cloud application developer and consists of SLOs and SQOs belonging to the CSPs the application is deployed on and additionally application specific SLOs and SQOs.

- **Cloud SLA.** A cloud Service Level Agreement (cloud SLA) is a contractual agreement between cloud service customers and cloud service providers. It is part of a cloud service agreement and includes service level objectives (SLO) and cloud service qualitative objectives (SQO) for the covered cloud service(s).

- **CSLA.** A composite cloud Service Level Agreement (CSLA) is an aggregated view of a number of SLAs from different CSPs which have been chosen for the deployment of a multi-cloud application. Aggregation patterns may be used in order to calculate an aggregated value for SLOs occurring across multiple different CSP SLAs.

- **Aggregation pattern.** An aggregation pattern is a formalised arithmetic rule that can be used to aggregate terms coming from different CSP SLAs and resulting in a composite SLA. [1]

- **Multi-cloud architectural pattern.** The multi-cloud architectural patterns will comprise best practices and recommendations on how to design and structure the multi-cloud applications supporting the NFR selected (i.e. componentization through microservices) to guide architects in software architectural decisions. This guidance will be accompanied by a tool proposing the most suitable patterns and the patterns themselves (i.e. nuggets of code, skeleton, etc.) to comply the NFRs and multi-cloud architecture.

- **Metric.** A standard of measurement that defines the conditions and the rules for performing the measurement and for understanding the results of a measurement. [2]

- **Service Level Objectives (SLO).** A Service Level Objective (SLO) describes a specific Quality of Service (QoS) aspect of a Service Level Agreement (SLA). Each guarantee contained in a SLA about QoS is expressed by a SLO. Syntactically a SLO is composed of a SLA Term, indicating a specific aspect of a service (e.g. its availability), and a conditional expression indicating a measurable constraint on the SLA Term itself.

- **Service Qualitative Objectives.** A Service Quality Objective (SQO) indicates a high level goal about the quality of a service.

- **Remedy.** A compensation offered to the cloud service customer in the event the cloud service provider fails to meet specific cloud service objectives. A remedy is typically part of an SLA and is included in its documentation. [2]

- **Application Profile.** This term refers to the profiling of the multi-cloud application so that the simulation can be performed. The multi-cloud application has to be profiled and classified into known stereotypes of components (microservices). Based on those known stereotypes the simulation process can be performed.

- **Deployment Schema (OPTIMUS).** The deployment schema is the information about CSPs that are involved in the best possible deployment taking into account the NFRs, provided by the development. The schema will associate one CSP to each microservice of the multi-cloud application. This option for the deployment is the best theoretical option obtained by the OPTIMUS tool. Based on this Schema, the CONTROLLER will create the script needed for deploying the whole application, asking the ACSmI for the CSPs that are described in it.

- **KPI.** KPI (Key Performance Indicator) in DECIDE is a measurable value that demonstrates how effectively the different WPs and KR are achieving their objectives.

- **Application Description.** The Application description is a holistic view, defined by meta-data, over the properties of an application. It holds properties such as the application name, its composition, classification, deployment topology, etc.

2.1 KPIs

This section will list the KPIs that will be monitored by DECIDE. KPIs, or Key Performance Indicators, are the mechanism that will be used in the project to measure the level of accomplishment of the DECIDE goals. They are classified according to the KR they affect.

2.1.1 KR1

KPI 1.1 - Fully functional	
Definition	The related KR must be functional.
Measurement Mechanism	95% of functional requirements to be validated in each version are functioning well (with no errors). Functional test cases.

KPI 1.2 - 70% requirements covered	
Definition	The 70% of the functionalities are covered.
Measurement Mechanism	The 70% of the functional requirements initially elicited are covered.

2.1.2 KR2

KPI 2.1 - White paper & Web site	
Definition	The multi-cloud architectural patterns are included in a white paper and a web site.
Measurement Mechanism	Accepted white paper (conference or journal) and content available in the DECIDE website.

KPI 2.2 - Use case requirements	
Definition	The resulting architectural patterns support the requirements form the different use cases in DECIDE.
Measurement Mechanism	Some of the patterns are implemented in all the use cases.

KPI 2.3 - Accuracy 70%	
Definition	The accuracy of the supporting tool in providing candidate patterns in the three different phases as well as the order in which they are to be applied is of 70%.
Measurement Mechanism	The accuracy of the tools proposal with respect to the ones proposed manually by the use cases is of 70%.

2.1.3 KR3

KPI 3.1 - Correctness	
Definition	Degree of correctness of deployment alternatives compared with real deployment data.
Measurement Mechanism	The accuracy of the deployment topologies proposed by OPTIMUS with respect to the topologies selected by the use cases (manually) is of 70%.

KPI 3.2 - More than 5 topologies	
Definition	Number of deployment topologies supported more than 5.
Measurement Mechanism	OPTIMUS output can provide at least 5 different topologies (combination of deployments configuration) in different simulation processes.

2.1.4 KR4

KPI 4.1 - Fully functional	
Definition	The related KR must be functional.
Measurement Mechanism	95% of functional requirements to be validated in each version are functioning well (with no errors). Functional test cases.

KPI 4.2 - 70% functionalities validated by the use cases	
Definition	70% of the functionalities are validated by the use cases.
Measurement Mechanism	Percentage of functional requirements for the related KR implemented in the use cases.

KPI 4.3 - Satisfaction by users 90%	
Definition	Satisfaction recorded by user is 90%.
Measurement Mechanism	Satisfaction questionnaires for the use cases, with rates of satisfaction of 90% or more.

2.1.5 KR5

KPI 5.1 - 90% NFR	
Definition	Degree of correctness of the re-deployment configuration compared with NFR and other requirements (at least 90% of NFR).
Measurement Mechanism	The percentage of the degree of fulfilment of the NFRs in automatic re-deployment scenarios will be of 90%.

KPI 5.2 - 70% functionalities validated by the use cases	
Definition	70% of the functionalities are validated by the use cases.
Measurement Mechanism	Percentage of functional requirements for the related KR implemented in the use cases.

3 Actors

In this section, the actors that will make use of the DECIDE platform are described.

- **Salesperson:** gathers the first needs of the client, making an offer according to their requirements. In this phase, technologies to be used are defined, as well as the functional and non-functional application requirements. The salesman can also make an offer for new functionalities that the client asked for.

- **DECIDE operator:** the DECIDE operator fulfils tasks that correspond to both a developer and an operator. In the traditional sense, the developer develops the offer that was presented by the responsible of the acquisition project, as well as new functionalities that arise during development, based on the technologies that the client demanded. On the other hand, the operator is in charge of ensuring the proper working of the application. However, DECIDE fosters a DevOps approach, so the distinction between these two actors gets blurred. The DECIDE operator takes on developer's responsibilities before deployment and operator's responsibilities after deployment.

- **Cloud services provider:** company that offers network services, infrastructure or business applications in the cloud. These cloud services are hosted in the company's data centers and are accessible to users through the Internet.

- **End user:** the end user of the multi-cloud application, to whom the MCSLA applies.

4 DECIDE DevOps Requirements

As mentioned in previous sections, one of the goals of the DECIDE framework is to support the DevOps principles. Thus, this section will list the requirements imposed by the DevOps philosophy. An in-depth DevOps analysis, from which the aforementioned principles are derived, can be found in the annex.

Table 2. DECIDE DevOps Requirements

ID	Description	Source	Related KR
DEVOPS-REQF1	DECIDE framework must facilitate small and frequent updates of the code	DevOps Principles #1	KR1
DEVOPS-REQF2	DECIDE framework must support the automatic deployment of the infrastructure required for the development	DevOps Principles #6	KR1
DEVOPS-REQF3	DECIDE framework must be able to monitor the deployed microservices	DevOps Principles #7	KR5
DEVOPS-REQF4	DECIDE framework must use microservices	DevOps Principles #2	KR1
DEVOPS-REQF5	DECIDE framework must support the continuous integration of the developed apps	DevOps Principles #3	KR1
DEVOPS-REQF6	DECIDE framework must support the continuous deployment of the developed apps	DevOps Principles #5	KR1
DEVOPS-REQF7	DECIDE framework must support the continuous delivery of the developed apps	DevOps Principles #4	KR1
DEVOPS-REQF8	DECIDE framework must facilitate the automatic provisioning of infrastructure	DevOps Principles #6	KR4
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	KR5
DEVOPS-REQF10	DECIDE framework must provide a way for team members to communicate with each other.	DevOps Principles #8	KR1
DEVOPS-REQF11	DECIDE framework must provide a way for team members to plan the development process	DevOps Principles #9	KR1
DEVOPS-REQF12	DECIDE framework must guarantee the security of the microservices	DevOps Principles #10	KR2
DEVOPS-REQF13	DECIDE framework must support the application of best practices and design principles during the first phases of the development	Extended DevOps #1	KR1
DEVOPS-REQF14	DECIDE must provide mechanisms to analyze alternative cloud deployment scenarios and their impact in the NFR of the application, in the MCSLA and the cost	Extended DevOps #2	KR3
DEVOPS-REQF15	DECIDE must support the monitoring of the multi-cloud application SLA and the SLAs of the underlying cloud resources	Extended DevOps #3	KR5
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	KR5
DEVOPS-REQF17	DECIDE must support the monitoring of the SLAs of the underlying cloud resources	Extended DevOps #3	KR4

5 Key Result 1 (KR1) Requirements

The DECIDE framework aims at fulfilling three main goals:

- To support DevOps principles (defined in this deliverable).
- To integrate all the functionalities of the different DECIDE components (developed in Work Packages 3 to 5) to support the project's Key Results.
- To be validated in the three use-case scenarios defined in Work Package 6.

Taking this into account, this section will gather all KR1 requirements, functional and non-functional, that are necessary to fulfil the aforementioned goals.

Furthermore, a list of all the tools needed to support these requirements will be compiled.

5.1 Functional Requirements

Req. ID	KR1-REQ1
Req. Short Title	Entry point
Req. Description	The system must provide the user with an entry point to DECIDE
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Integration
Supported Functionality of the DevOps framework	Integration
Source	DoA
Priority	High

Req. ID	KR1-REQ2
Req. Short Title	UI unification
Req. Description	The system must unify transparently the UIs from the different KRs
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Integration
Supported Functionality of the DevOps framework	Integration
Source	DoA
Priority	High

Req. ID	KR1-REQ3
Req. Short Title	Generic UI
Req. Description	The system must provide a generic DECIDE UI

Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Development
Supported Functionality of the DevOps framework	Development
Source	DoA
Priority	High

Req. ID	KR1-REQ4
Req. Short Title	Patterns reception
Req. Description	The system must receive ARCHITECT's patterns
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Development
Supported Functionality of the DevOps framework	Development
Source	DoA
Priority	High

Req. ID	KR1-REQ5
Req. Short Title	Development environment-Patterns
Req. Description	The developer must have access to a development environment with the received patterns
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Development
Supported Functionality of the DevOps framework	Development
Source	DoA
Priority	High

Req. ID	KR1-REQ6
Req. Short Title	Development environment-Configurations
Req. Description	The developer must have access to a development environment with

	preloaded DECIDE configurations
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Development
Supported Functionality of the DevOps framework	Development
Source	DoA
Priority	High

Req. ID	KR1-REQ7
Req. Short Title	Code submission
Req. Description	The system must allow the developer to submit their code
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Development
Supported Functionality of the DevOps framework	Development
Source	Medium
Priority	High

Req. ID	KR1-REQ8
Req. Short Title	Code versioning
Req. Description	The system must be able to version the code submitted by the developer
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Development
Supported Functionality of the DevOps framework	Development
Source	DoA
Priority	Medium

Req. ID	KR1-REQ9
Req. Short Title	Dependencies
Req. Description	The system must be able to resolve the dependencies of the submitted code

Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Integration
Supported Functionality of the DevOps framework	Integration
Source	DoA
Priority	Medium

Req. ID	KR1-REQ10
Req. Short Title	Compilation
Req. Description	The system must compile the code without errors
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Development
Supported Functionality of the DevOps framework	Development
Source	DoA
Priority	Medium

Req. ID	KR1-REQ11
Req. Short Title	Testing preparation
Req. Description	The system must receive the testing activities that have to be performed on the code
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Testing
Supported Functionality of the DevOps framework	Testing
Source	DoA
Priority	Medium

Req. ID	KR1-REQ12
Req. Short Title	Testing activities
Req. Description	The system must be able to perform the received testing activities

Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Testing
Supported Functionality of the DevOps framework	Testing
Source	DoA
Priority	Low

Req. ID	KR1-REQ13
Req. Short Title	Testing results
Req. Description	The system must present the results from the testing activities
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Testing
Supported Functionality of the DevOps framework	Testing
Source	DoA
Priority	Low

Req. ID	KR1-REQ14
Req. Short Title	Code continuity
Req. Description	The system must guarantee the continuity of the code within DECIDE's workflow
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Integration
Supported Functionality of the DevOps framework	Integration
Source	DoA
Priority	Low

Req. ID	KR1-REQ15
Req. Short Title	Code availability
Req. Description	The system must make the code available for DECIDE

Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Development
Supported Functionality of the DevOps framework	Development
Source	DoA
Priority	Low

Req. ID	KR1-REQ16
Req. Short Title	Pattern fulfilment
Req. Description	The system must guarantee the fulfilment of DECIDE's patterns by the developer
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Development
Supported Functionality of the DevOps framework	Development
Source	DoA
Priority	Low

Req. ID	KR1-REQ17
Req. Short Title	NFR gathering
Req. Description	DECIDE DevOps framework must provide support for NFR gathering
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Design
Supported Functionality of the DevOps framework	NFR specification
Source	DoA
Priority	High

Req. ID	KR1-REQ18
Req. Short Title	Qualitative NFP
Req. Description	The system must support developers establishing qualitative NFP that the

	application must comply with (i.e. security, location, financial, low/high technological risk)
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Design
Supported Functionality of the DevOps framework	NFR specification
Source	DoA
Priority	Medium

Req. ID	KR1-REQ19
Req. Short Title	Quantitative NFP
Req. Description	The system must support developers establishing quantitative NFP that the application must comply with (i.e. MBTF, availability, response time, lag, cost, throughout))
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Design
Supported Functionality of the DevOps framework	NFR specification
Source	DoA
Priority	Low

Req. ID	KR1-REQ20
Req. Short Title	(MC)SLA editor
Req. Description	The system must include a (MC)SLA editor
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Design (pre-deployment)
Supported Functionality of the DevOps framework	(MC)SLA monitoring
Source	DoA
Priority	High

Req. ID	KR1-REQ21
Req. Short Title	Application controller
Req. Description	The system must include an Application Controller
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Deployment preparation
Supported Functionality of the DevOps framework	Current deployment configuration and history
Source	DoA
Priority	High

Req. ID	DEVOPS-REQ1
Req. Short Title	DECIDE framework must facilitate small and frequent updates of the code
Req. Description	Frequent updates
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Implementation
Supported Functionality of the DevOps framework	Development
Source	DevOps Principles #1
Priority	Low

Req. ID	DEVOPS-REQ4
Req. Short Title	Microservices
Req. Description	DECIDE framework must use microservices
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Does not apply
Supported Functionality of the DevOps framework	Does not apply
Source	DevOps Principles #2
Priority	High

Req. ID	DEVOPS-REQ5

Req. Short Title	Continuous integration
Req. Description	DECIDE framework must support the continuous integration of the developed apps
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Integration
Supported Functionality of the DevOps framework	Integration
Source	DevOps Principles #3
Priority	Low

Req. ID	DEVOPS-REQ10
Req. Short Title	Communication
Req. Description	DECIDE framework must provide a way for team members to communicate with each other.
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Does not apply
Supported Functionality of the DevOps framework	Does not apply
Source	DevOps Principles #8
Priority	Low

Req. ID	DEVOPS-REQ11
Req. Short Title	Planning
Req. Description	DECIDE framework must provide a way for team members to plan the development process
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Does not apply
Supported Functionality of the DevOps framework	Does not apply
Source	DevOps Principles #9
Priority	Low

Req. ID	DEVOPS-REQ13
Req. Short Title	Design principles
Req. Description	DECIDE framework must support the application of best practices and design principles during the first phases of the development
Phase of Cloud service life cycle	Does not apply
Phase/subphase of the DevOps framework	Development phase/Implementation
Supported Functionality of the DevOps framework	Development
Source	Extended DevOps #1
Priority	Low

6 DECIDE High-level Functionalities

DECIDE aims at providing a new generation of multi-cloud, services-based software framework. To achieve that, DECIDE will enable techniques, tools and mechanisms to design, develop, operate and deploy multi-cloud-aware applications. It will also provide architectural patterns and the necessary supporting tools to develop and operate following a DevOps approach.

The table below summarizes the high-level functionalities that are required to reach DECIDE's goals. These functionalities have been obtained collaboratively, by analyzing the Project Description, the use cases and the principles of DevOps. They will be used as a starting point to elicit lower level requirements through an analysis that will take place in the corresponding WP. They are classified according to the development phase and ordered following the life cycle of application development, so that traceability between functionalities and requirements can be kept.

Table 3. DECIDE high-level functionalities

Phase	Sub-phase	Supported functionality	Related KR	Related WP
Development phase	Design	NFR specification	KR1	WP2
Development phase	Design	Architectural design	KR2-ARCHITECT	WP3
Development phase	Implementation	Development	KR1	WP2
Development phase	Integration	Integration	KR1	WP2
Development phase	Testing	Testing	KR1	WP2
Development phase	Pre-deployment	Application (nodes and communication included) profiling/classification	KR3-OPTIMUS	WP3
Development phase	Pre-deployment	Theoretical deployment generation	KR3-OPTIMUS	WP3

Phase	Sub-phase	Supported functionality	Related KR	Related WP
Development phase	Pre-deployment	CSP modeling	KR3-OPTIMUS	WP3
Development phase	Pre-deployment	Simulation (deployment)	KR3-OPTIMUS	WP3
Development phase	Pre-deployment	Cloud services discovery	KR4-ACSmI	WP5
Development phase	Optimization	Code optimization	KR2-ARCHITECT	WP3
Development phase	Design/Pre-deployment	(MC)SLA definition	KR1	WP2
Operation phase	Deployment preparation	Manage CSPs connectors	KR4-ACSmI	WP5
Operation phase	Deployment preparation	Cloud services contracting	KR4-ACSmI	WP5
Operation phase	Deployment preparation	Create and update the service catalogue into the ACSmI	KR4-ACSmI	WP5
Operation phase	Application deployment	Deployment	KR5-ADAPT	WP4
Operation phase	Application deployment	Deployment	KR5-ADAPT	WP4
Operation phase	Application deployment	Deployment	KR5-ADAPT	WP4
Operation phase	Application deployment	(Deployment) Configuration management	KR5-ADAPT	WP4
Operation phase	Application Monitoring	Application MCSLA monitoring	KR5-ADAPT	WP4
Operation phase	Application Monitoring	NFR monitoring	KR5-ADAPT	WP4
Operation phase	Application Monitoring	CSP monitoring	KR4-ACSmI	WP5
Operation phase	Application Adaptation	Handle violations	KR5-ADAPT	WP4
Operation phase	Application Adaptation	Application adaptation	KR5-ADAPT	WP4
Operation phase	Application Adaptation	Application re-deployment	KR5-ADAPT	WP4
Operation phase	Deployment preparation	Current deployment configuration and history	KR1	WP2
Operation phase	Application Monitoring	Monitor the usage of the services	KR4-ACSmI	WP5

7 Integration Analysis

This section aims at analyzing the integration requirements between the different DECIDE components in terms of data exchanges.

There is one subsection devoted to each KR. Every subsection includes a brief description of the KR, a summary of its functionalities, a data flow diagram and its integration requirements. The integration requirements are classified by functionality. For each functionality, the data exchanged by the component that gives support to said functionality with other components within the KR (internal) or with other KRs (external) are specified. This data flow is summarized in a table that indicates whether the data exchange is internal or external, the origin or destination of the data and the data that will be exchanged.

The following diagram shows DECIDE's global data flow. A more detailed analysis can be found below.

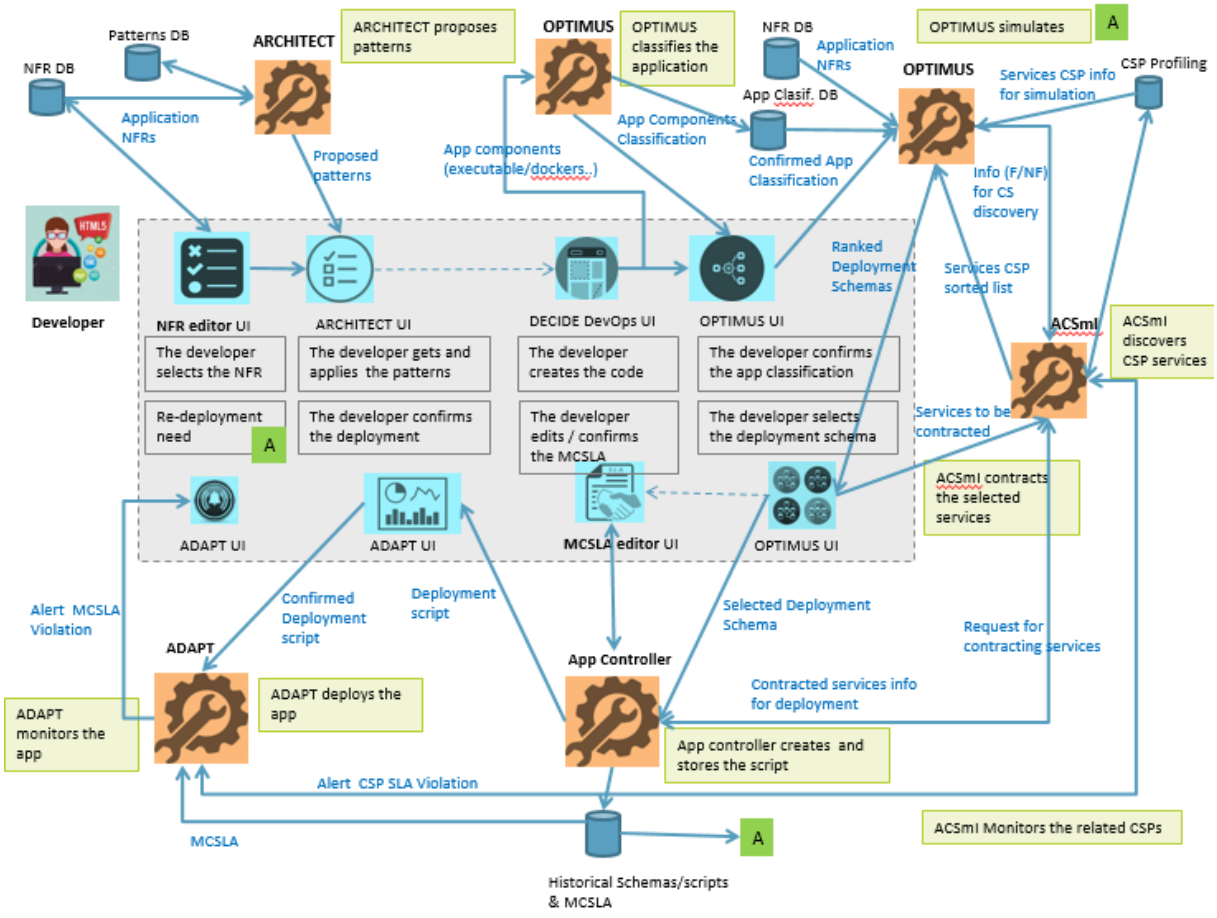

Figure 1. DECIDE's global workflow

7.1 DECIDE Framework (KR1)

7.1.1 Description

DECIDE's Key Result 1 consists in a DevOps framework for multi-cloud applications. This framework should support the development and delivery pipeline for multi-cloud applications, provide means to define the multi-cloud SLAs under which the applications must work, and enable continuous deployment, integration and quality.

This Key Result must support the following functionalities:

Table 4. KR1 Functionalities

Phase	Sub-phase	Functionality
Development phase	Design	NFR Specification
Development phase	Design	(MC)SLA definition
Development phase	Implementation	Development
Development phase	Integration	Integration
Development phase	Testing	Testing
Operation phase	Deployment preparation	Current deployment configuration and history

The figure below shows the workflow that will give support to the listed functionalities:

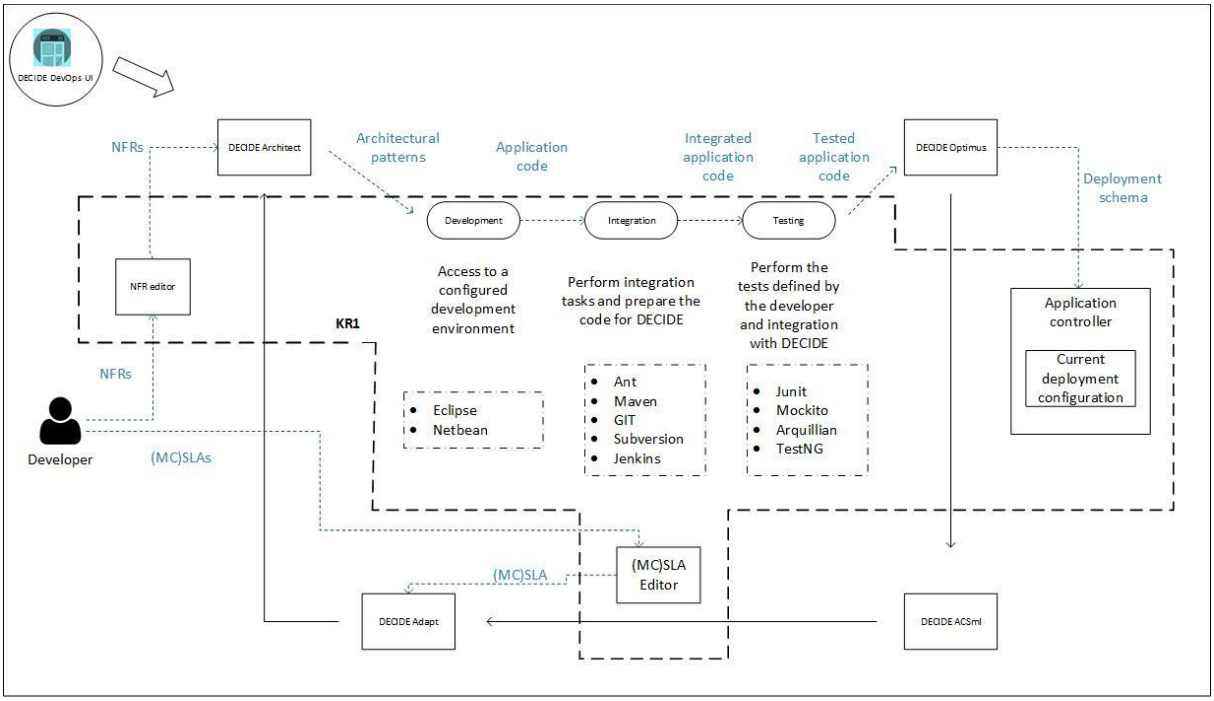

Figure 2. KR1's workflow diagram

The following sub-sections will detail the integration requirements of the KR1 and the different DECIDE components, as well as the integration requirements for the modules within the KR1.

7.1.2 Integration requirements

7.1.2.1 Development phase

7.1.2.1.1 Design

- **NFR Specification**

The NFR Specification module serves as the interface through which the developer will specify the non-functional requirements that the application must fulfill, which affects the definition of the architectural patterns. The module will deliver the NFR to the ARCHITECT KR.

Table 5. Integration requirements of the NFR Specification module

Inputs		
Internal/External	**Origin**	**Data**
External	Developer	NFRs

Outputs		
Internal/External	**Destination**	**Data**
External	ARCHITECT	NFRs

- **(MC)SLA Definition**

This module serves as the interface through which the developer will specify the multi-cloud SLAs agreed with the client, which, as is the case with the NFRs, affects the definition of the architectural patterns. This module will store the MCSLA in the application description to be accessed by ADAPT.

Table 6. Integration requirements of the (MC)SLA Definition module

Inputs		
Internal/External	**Origin**	**Data**
External	Developer	(MC)SLA

Outputs		
Internal/External	**Destination**	**Data**
External	ADAPT	(MC)SLA

7.1.2.1.2 Implementation

- **Development**

DECIDE must provide access to a configured development environment (in particular to Eclipse and Netbean). To achieve that, the KR1 must receive from ARCHITECT the architectural patterns that will serve for the integration during the whole development cycle. It will also deliver to the KR1's integration module the application code.

Table 7. Integration requirements of the Development module

Inputs		
Internal/External	**Origin**	**Data**
External	ARCHITECT	Architectural patterns

Outputs		
Internal/External	**Destination**	**Data**
Internal	Integration module	App. code

7.1.2.1.3 Integration

- **Integration**

The integration module must perform all the needed integration tasks, such as versioning the code or resolving dependencies, as well as prepare the code for the rest of the DECIDE workflow. It will receive the application code from the development module, and provide the testing module with the integrated application code.

Table 8. Integration requirements of the Integration module

Inputs		
Internal/External	**Origin**	**Data**
Internal	Development module	App. code

Outputs		
Internal/External	**Destination**	**Data**
Internal	Integration module	Integrated app. code

7.1.2.1.4 Testing

- **Testing**

This module is in charge of performing the tests defined by the developer. It receives the code to be tested from the integration module and it delivers to DECIDE Optimus tested application code that is ready to continue the DECIDE life cycle.

Table 9. Integration requirements of the Testing module

Inputs		
Internal/External	**Origin**	**Data**
Internal	Integration module	Integrated app. code

Outputs		
Internal/External	**Destination**	**Data**
External	Optimus	Tested app. code

7.1.2.2 Operation phase

7.1.2.2.1 Deployment preparation

- **Current deployment configuration and history**

The Application controller is integrated into the Decide Framework and will output its information to the DECIDE DevOps UI. The Application Controller will receive information on the deployment configuration from OPTIMUS in order to store this data and make it viewable to the user.

Table 10. Integration requirements of the Application Controller

Inputs		
Internal/External	**Origin**	**Data**
External	OPTIMUS	Deployment schema

Outputs		
Internal/External	**Destination**	**Data**
internal	Integration module/ DevOps UI	Current deployment and history

7.2 DECIDE ARCHITECT (KR2)

7.2.1 Description

Key Result 2 encompasses the ARCHITECT tool, which consists of a catalogue of architectural patterns. These patterns serve the optimization, development and deployment of applications to become multi-cloud aware. The idea is to provide the developers with a set of patterns to apply to their code.

The figure below shows the workflow that will give support to the listed functionalities:

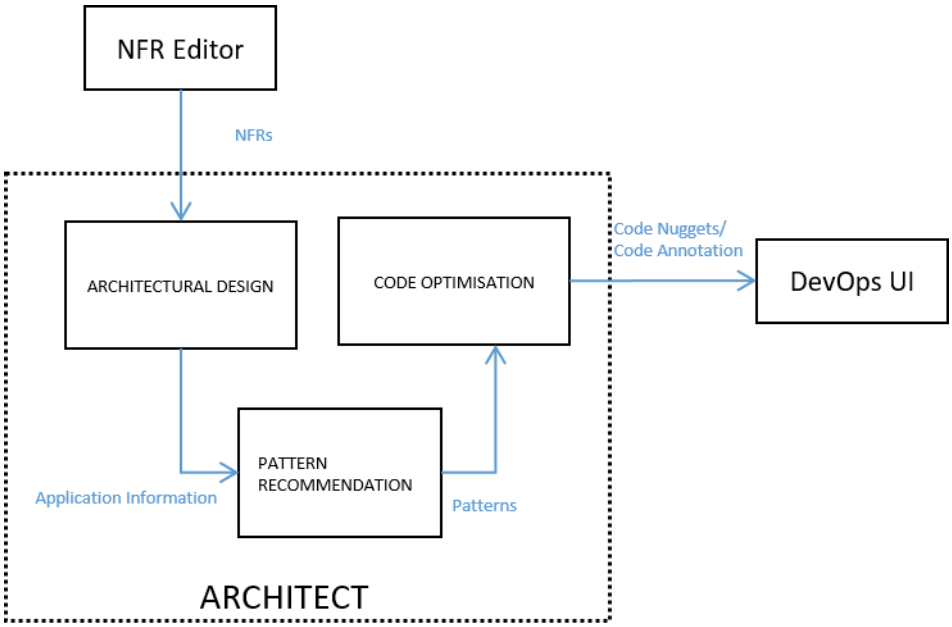

Figure 3. KR2's workflow diagram

The table below provides a summary of all the functionalities that the KR gives support to.

Table 11. KR2 Functionalities

Phase	Sub-phase	Functionality
Development phase	Design	Architectural design
Development phase	Design	Pattern recommendation
Development phase	Optimization	Code Optimization

7.2.2 Integration requirements

In the ARCHITECT, the developer is presented with a set of architectural patterns that are bound to the NFRs previously selected. The output of the architectural patterns should be seamlessly integrated with the DECIDE DevOps UI and the developer should be able to integrate this output into his/her code.

7.2.2.1 Development Phase

7.2.2.1.1 Design

- **Architectural Design**

During the architectural design phase, the developer is provided with a form or a wizard that elicits specific information in order to make the next step and recommend patterns.

Table 12. Integration requirements of Architectural design

Inputs		
Internal/External	**Origin**	**Data**
External	NFR Editor	NFRs

Outputs		
Internal/External	**Destination**	**Data**
Internal	Pattern Recommendation	Application Information

- **Pattern recommendation**

Pattern recommendation in the design phase will select a number of patterns that are applicable to the NFRs entered by the user as well as additional information entered by the user regarding application properties.

Table 13. Integration requirements of Pattern recommendation

Inputs			Outputs		
Internal/External	**Origin**	**Data**	**Internal/External**	**Destination**	**Data**
Internal	Architectural Design	Information Application	Internal	Code Optimization	Patterns

7.2.2.1.2 Optimization

- **Code Optimization**

Depending on the selected NFRs specific code optimizations are suggested as patterns for the developer to introduce into the code.

Table 14. Integration requirements of Code optimization

Inputs			Outputs		
Internal/External	**Origin**	**Data**	**Internal/External**	**Destination**	**Data**
Internal	Pattern Recommendation	Patterns	External	DECIDE DevOps UI	Optimization patterns (code nuggets, code annotation)

7.3 DECIDE OPTIMUS (KR3)

7.3.1 Description

OPTIMUS deployment simulation tool will be responsible for evaluating and optimizing the non-functional characteristics from the developer's perspective considering a set of provided cloud resources alternatives. OPTIMUS, working with the continuous deployment supporting tool (DECIDE ADAPT), will provide the best possible deployment application topology, based on the non-functional requirements set by the developer, automating the provisioning and selection of deployment scripts for multi-cloud applications.

The main functionalities in OPTIMUS are:

- **Multi-cloud application classification.** This functionality will include the profiling and classification of the components that form the multi-cloud application, and also the profiling of the nodes and communications involved in its deployment. This classification will be based on the information provided by the developer and the information stored in the general applications profiling repository.
- **Theoretical deployment generation.** Once the classification is made, and the NFRs gathered, it will perform a process where it will obtain a theoretical schema for the deployment. This schema will be composed of generic types of CSPs, associated to the types set to the micro services. With these generic types of CSPs suitable for the components, a request will be

made to the corresponding service of ACSmI. This functionality requires the "CSP modelling" functionality to be available.

- **Simulation.** The combination of the different possibilities of deployment, taking into account the theoretical deployment and the sorted list of CSPs (from ACSmI) that suit in it, will be ranked in order to select the best of them. The Schema with the needed information for the CONTROLLER will be built and shown to the developer to confirm it.

Table 15. KR2 Functionalities

Phase	Sub-phase	Functionality
Development	Pre-deployment	Application (nodes and communication included) profiling/classification
Development	Pre-deployment	Theoretical deployment generation
Development	Pre-deployment	CSP modelling[1]
Development	Pre-deployment	Simulation (deployment)

The figure below shows the workflow that will support the listed functionalities:

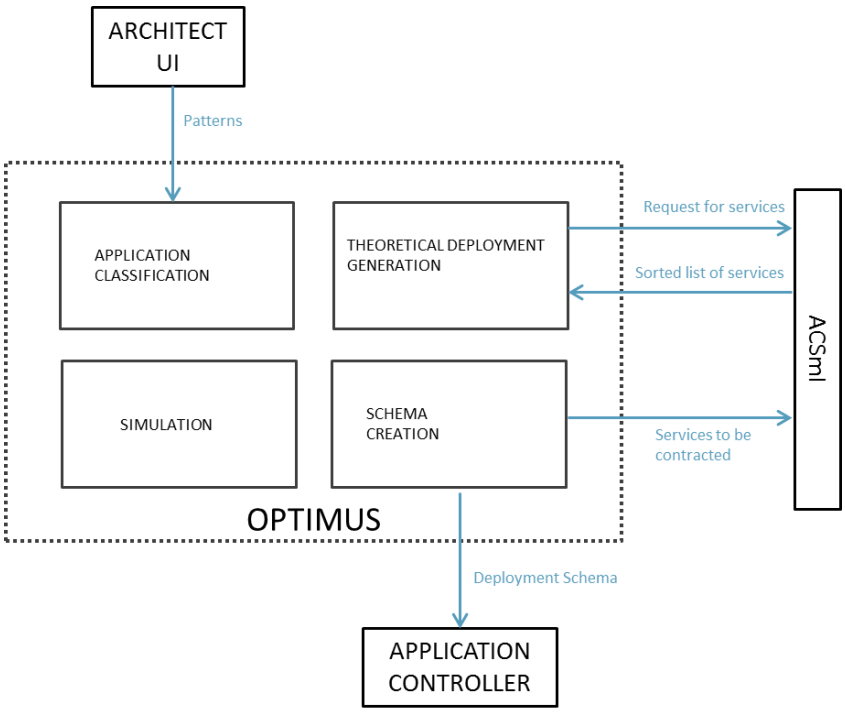

Figure 4. KR3's workflow diagram

[1] This functionality is a requirement to be covered before starting the development of the multi cloud application. It is a previous activity to be done at WP3 but the CSP profile will be used by other KRs, but it is not a tool itself.

7.3.2 Integration requirements

7.3.2.1 Development

7.3.2.1.1 Pre-deployment

- **Multi-cloud application classification.**

The input for this classification task will come from the developer (the UI will show information to complete and confirm), and it will be matched with the information in the "General apps Profiling" repo. The output will be loaded into "Apps classification" repo. This information could be part of the app description and it will be completed during the whole life cycle.

The "General apps profiling" repo contains the system knowledge about types of multi-cloud applications, and the characteristics associated to each of those types. Also, it could contain information about nodes and communication profiling. This information should be related to the defined CSP types (ACSmI).

Table 16. Integration requirements of Multi-cloud application classification

Inputs		
Internal/External	**Origin**	**Data**
External	General apps profiling	General information about apps

Outputs		
Internal/External	**Destination**	**Data**
External	Apps classification repo	App classification data

- **Theoretical deployment generation.**

The "Deployment Types" repo will contain information about the microservices types and the CSPs in which they could be theoretically deployed. The maintenance of this repo will be performed by the Deployment types management module. At this point of the project, it could be static information but later it should be updated with data provided by ACSmI.

Taking as input the multi-cloud application classification (microservices) and the Apps NFRs, the theoretical deployment generation component will access the "Deployment Types" repo to obtain the set of CSPs that can be used for its deployment. Once it has all the information, it will create a list of possible CSPs for each microservice, and assign some sort of score for each option.

Table 17. Integration requirements of Theoretical deployment generation

Inputs		
Internal/External	**Origin**	**Data**
Internal	Apps classification repo	App classification data
External	Apps NFRs Repo	App NFRs
External	Dep. Types Repo	Types of theoretical deployments
External	ACSmI	Sorted list of services

Outputs		
Internal/ External	**Destination**	**Data**
Internal	Theoretical deployments management	Theoretical deployments
Internal	Simulation	Possibilities for deployment
External	OPTIMUS	Requirements of the services

- **Simulation**

The input for the combination process will be the information about different possibilities for the deployment. The algorithm will perform a combination of all these possibilities, using the different CSPs that can be used for each micro service. These combinations will have a score for sorting them. An output with the best deployment will be created.

The best option for the deployment will be selected and confirmed by the developer, and sent (as schema) to the controller, and, eventually, to ACSmI for contracting the services involved. It has to be decided when this contracting action will be performed, either at this point or when the CONTROLLER sends the script to ADAPT.

Table 18. Integration requirements of the Service discovery module

Inputs		
Internal/External	**Origin**	**Data**
Internal	Theoretical deployments management	Possibilities for deployment

Outputs		
Internal/External	**Destination**	**Data**
External	CONTROLLER	Best Schema for app deployment
External	ACSmI	Services to be contracted

7.4 DECIDE ACSmI (KR4)

7.4.1 Description

Advanced Cloud Service (meta-) intermediator (ACSmI) will provide means to assess continuous real time verification of the cloud services non-functional properties fulfilment and legislation compliance enforcement. ACSmI will also provide a cloud services store where developers can easily access centrally negotiated deals of compliant and accredited applications developed by the software sector.

There are six main functionalities [3]:

1. **Endorse a cloud service into the ACSmI.** ACSmI will allow to register services. This can be done by a CSP, a developer or by the ACSmI Administration itself. The registry of each service should cover specific terms for modelling the CSPs and their services. This will allow the discovery of the services from a service catalogue (also named service registry).
2. **Discover & benchmark services.** OPTIMUS will indicate NFRs of the services, and ACSmI will discover the services that exist in the catalogue which comply with the NFRs. Then ACSmI will prioritize the discovered services depending on the level of fulfilment of the NFRs (including the legal ones) expressed by the multi-cloud application. After that, the discovered services will be provided to OPTIMUS as a sorted list, indicating the degree of fulfilment.
3. **Service Contracting.** This functionality will be responsible for handling all the activities related to the contracts establishment with ACSmI. Depending on the type of services and a CSP, ACSmI will manage the contracts in two different ways:
    a. ACSmI will facilitate the service contracting directly between a user and a provider.
    b. ACSmI will manage the contract itself. In this case ACSmI will have mainly two types of contracts. The first one is the contract with service providers, and the other one with the user of the services.
4. **Manage CSPs connectors.** This functionality will enable the management of the different connectors to facilitate the service contracting and monitoring. The feature will provide ADAPT with the required information for the deployment of the multi-cloud application onto different services.
5. **Monitor NFR of CSPs and manage the violation alert mechanisms.** This functionality will monitor SLA (NFRs) of the service offered by CSPs and detect violation of the SLA during the operation of the services. If a violation is detected, an alert to ADAPT will be sent.
6. **Usage monitoring and billing.** Since commercial cloud providers charge users for resource consumption, it is important that these charges are handled by ACSmI accordingly: the users will be charged in the background, provided with usage and billing reports, as well as with periodical invoices. Thus, the resource utilization will be monitored and users will be charged accordingly.

Table 19. KR4 Functionalities

Phase	Sub-phase	Functionality
Development	Pre-deployment	Cloud Services discovery
Operation	Deployment preparation	Manage CSPs connectors
Operation	Deployment preparation	Cloud services contracting
Operation	Application monitoring	CSP monitoring
Operation	Pre-deployment	Create and update the service catalogue into the ACSmI
Operation	Application Monitoring	Monitor the usage of the services

The figure below shows the workflow that will give support to the listed functionalities:

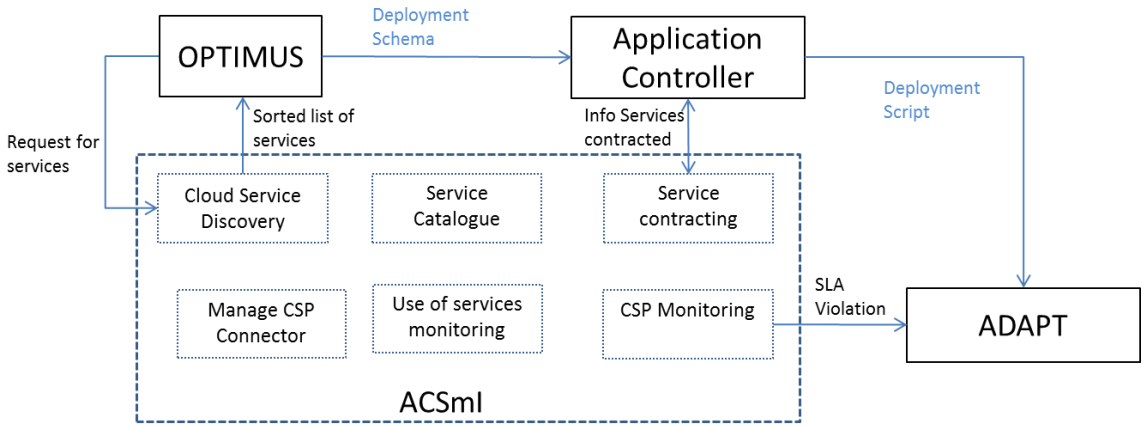

Figure 5. KR4's workflow diagram

7.4.2 Integration requirements

7.4.2.1 Development

7.4.2.1.1 Pre-Deployment

- **Service Discovery and management**

The goal of the CSPs service discovery and management is to support the registration and discovery of services in the ACSmI service catalogue.

The discovery and benchmarking components will take care of the requirements provided by OPTIMUS or by a user of ACSmI and will discover and benchmark the services that match these requirements.

OPTIMUS will provide to this component the information (mainly NFRs) of the services that OPTIMUS requires for carrying out the simulation. ACSmI will provide OPTIMUS with the information of the services that fulfil the NFRs as well as the degree of fulfilment.

Another aspect of the service management is the Service withdrawal of the service catalogue. If this service is operated by any application, this module should inform ADAPT to start a new deployment configuration process.

Table 20. Integration requirements of the Service discovery module

Inputs		
Internal/External	**Origin**	**Data**
External	OPTIMUS	Requirements of the services

Outputs		
Internal/External	**Destination**	**Data**
External	OPTIMUS	Sorted list of services

- **Create and update the service catalogue in the ACSmI**

This functionality aims to govern the service catalogue in order to maintain it as up-to-date as possible. Any changes in the service catalogue should be managed by the so-called service registry governance component.

This component will provide the interface for endorsing any service to the registry. The component will be accessed by the service discovery to look for appropriate services.

Table 21. Integration requirements of the Service catalogue module

Inputs			Outputs		
Internal/External	**Origin**	**Data**	**Internal/External**	**Destination**	**Data**
Internal	Service Discovery	Requirements of the services	Internal	Service Discovery	Services that fulfilled the requirements

7.4.2.2 Operation

7.4.2.2.1 Deployment preparation

- **Cloud services contracting**

The goal of this functionality is to contract the services required by a developer. Thus, OPTIMUS should provide ACSmI with the services to be subcontracted.

Once ACSmI has contracted the services with the CSP, ACSmI provides the information required to start service operation. This information should be provided to the Application Controller.

Table 22. Integration requirements of the Service Contracting module

Inputs			Outputs		
Internal/External	**Origin**	**Data**	**Internal/External**	**Destination**	**Data**
External	OPTIMUS/ Controller	Deployment Schema (info services to be contracted)	External	Controller	Info of the services for being deployed

- **Manage CSPs connectors**

This functionality will allow:

- Contracting services and monitoring them.
- Managing information required by ADAPT for the (re)deployment of a multi-cloud application onto the contracted services.

Table 23. Integration requirements of the CSPs connectors module

Inputs			Outputs		
Internal/External	**Origin**	**Data**	**Internal/External**	**Destination**	**Data**
Internal	Service Contracting	Contracted Services	External	CSP	Data Accessor

7.4.2.2.2 Application monitoring

- **CSP monitoring**

The goal of the CSP monitoring is to detect if the SLA contracted with the CSP is violated. In case SLA violation exists, a message to the ADAPT component should be sent. This message should indicate which service does not comply with the SLA, the type of violation and monitoring metrics if appropriate.

Table 24. Integration requirements of CSP Monitoring

Inputs			Outputs		
Internal/External	**Origin**	**Data**	**Internal/External**	**Destination**	**Data**
Internal	Service Contracting	Contracted Services	External	ADAPT	SLA Violation

- **Service usage monitoring**

The objective of the service monitoring is to bill the service users. For this, the component should communicate with the user registry and the contract registry in order to know which are the services contracted by each user.

Table 25. Integration requirements of the Service usage monitoring module

Inputs			Outputs		
Internal/External	**Origin**	**Data**	**Internal/External**	**Destination**	**Data**
Internal	Service Contracting	Contracted Services	External	Console	Billing data

7.5 DECIDE ADAPT (KR5)

7.5.1 Description

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 Non Functional Properties (NFP) of the Cloud Service Providers (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 cause the OPTIMUS tool to be launched again in order to search for another deployment configuration. Depending on the technological complexity requirement and the requirements initially prioritized 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.

The main functionalities of ADAPT are:

- **Deployment.** ADAPT will support automatic deployment of the user application. The input application is composed by one or more Containers hosting micro services and described by the Application Description Document.
- **(Deployment) Configuration management.** ADAPT will keep track of the endpoints of each micro service composing the application and will update this information upon application reconfiguration and re-deployment.
- **Application MCSLA monitoring.** ADAPT will monitor the application according to its defined (Multi-Cloud) SLA and will identify any related violations.
- **NFR monitoring.** ADAP will monitor the violations to the Non Functional Requirements (NFR) established for the application during the design phase; the actual SLA monitoring will be done by ACSmI and ADAPT will receive the violations.
- **Handle violations.** ADAPT will handle any violation raised either by monitoring the application MCSLA or the CSPs' NFRs. Violation handling may lead both to alerting the operator and to contacting OPTIMUS to obtain a new configuration to re-deploy the application.
- **Application adaptation.** ADAPT will support adaptation of applications in response to violations
- **Application re-deployment.** Application re-deployment will be the main method for enacting adaptation in response to violations.

Table 26. KR5 Functionalities

Phase	Sub-phase	Functionality
Operation phase	Application Deployment	Deployment
Operation phase	Application Deployment	(Deployment) Configuration management
Operation phase	Application Monitoring	Application MCSLA monitoring
Operation phase	Application Monitoring	NFR monitoring
Operation phase	Application Adaptation	Handle violations
Operation phase	Application Adaptation	Application adaptation
Operation phase	Application Adaptation	Application re-deployment

The figure below shows the workflow that will support the listed functionalities:

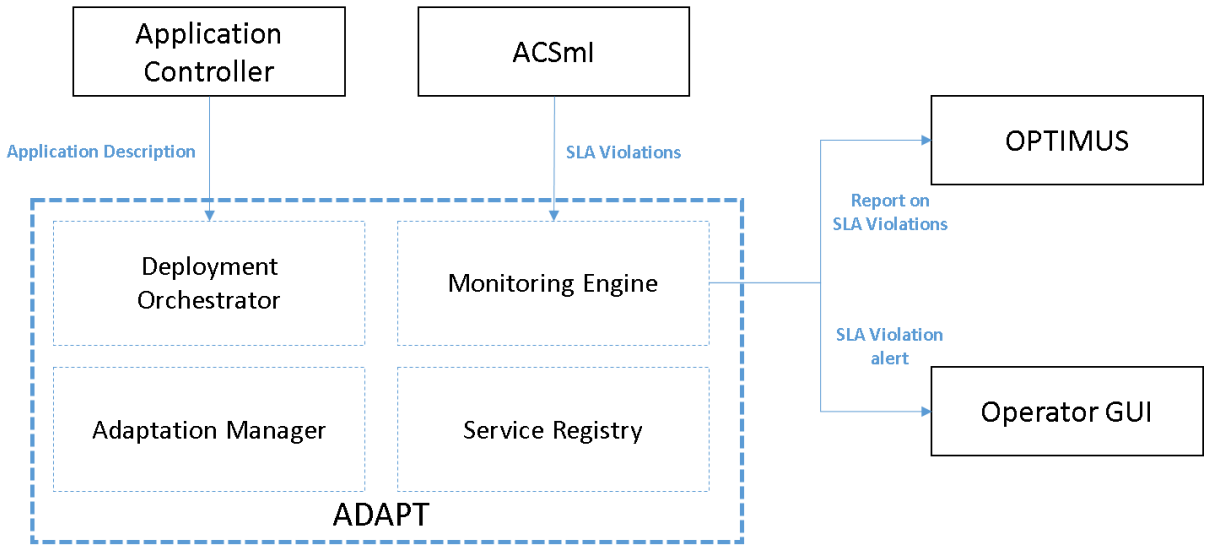

Figure 6. KR5's workflow diagram

7.5.2 Integration requirements

7.5.2.1 Operation

7.5.2.1.1 Deployment

- **Deployment**

The goal of this functionality is to orchestrate the deployment of the application described by the input Application Description. The information needed for deployment can be found in the Deployment Description section of the Application Description (see ***Figure 7*** below).

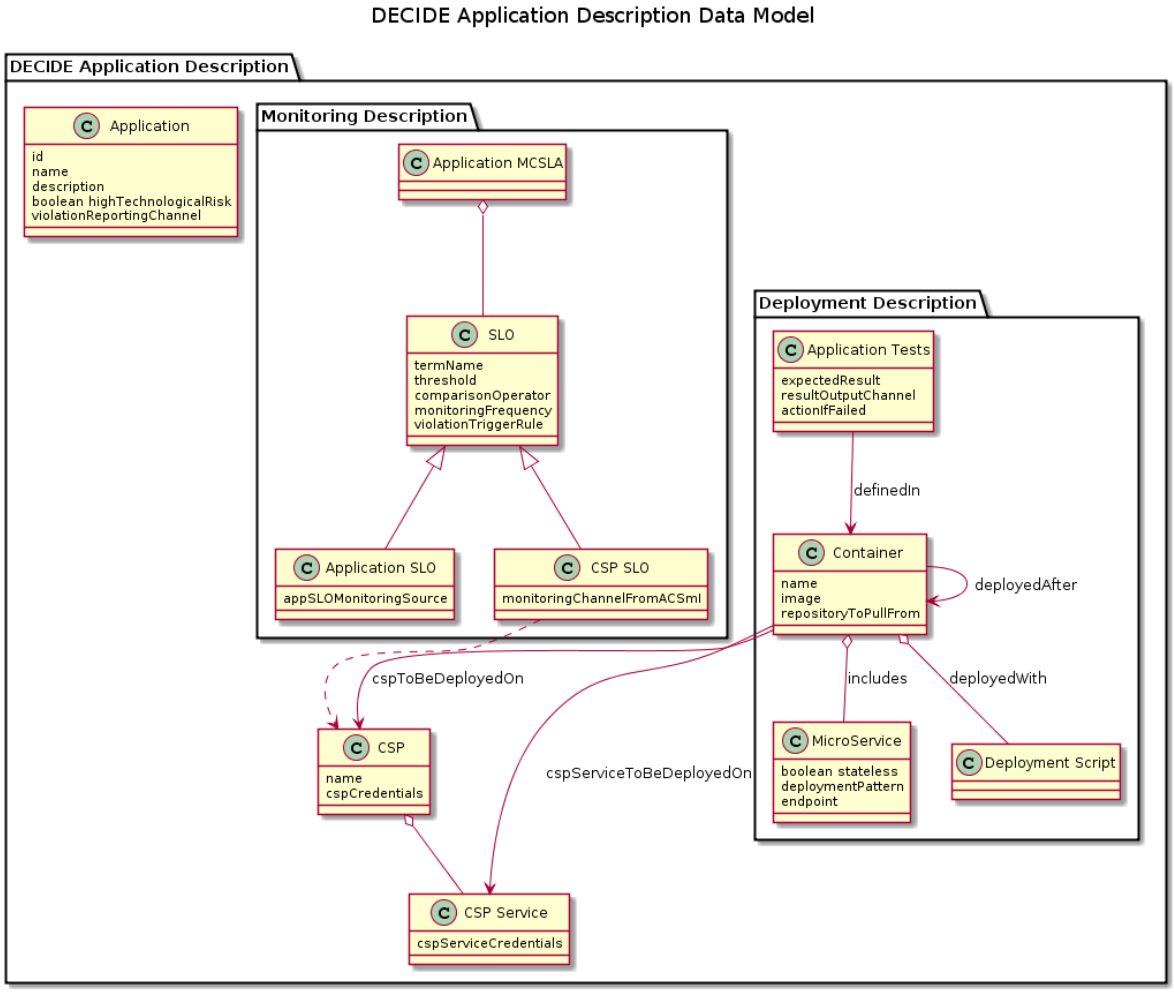

Figure 7. Application Description Data Model

The input for application deployment is included in the Deployment Description section of the Application Description (***Figure 7***). ADAPT must know which containers are part of the application to be deployed, from where their image can be downloaded, to which CSP and which service within that CSP the container should be deployed, and which script can be used for executing the deployment itself. The micro services hosted in each container and their statelessness should be known as well. Optional application tests can be provided in a specific container to assess the application readiness for being switched online to service client requests.

No specific output is planned from the Deployment Orchestrator module other than the exit status of the requested deployment operation.

Table 27. Integration requirements of the Deployment Orchestrator module

Inputs		
Internal/External	**Origin**	**Data**
External	Application Controller	Application Description
External	Application Controller	Application Containers
External	Application Controller	Deployment scripts

Outputs		
Internal/External	**Destination**	**Data**
External	Deployment method caller	Exit status

External	DECIDE Framework Development / Integration module	Optional Application tests

7.5.2.1.2 Deployment

- **(Deployment) Configuration management**

The goal of this functionality is to track the endpoints of each micro service composing the application and to update this information upon application reconfiguration and re-deployment. This allows the dynamic reconfiguration of the service proxy through which each application micro service can be reached independently by its (possibly changing) location.

The application developer should define the REST endpoint of each micro service composing the application.

The proxy prefix to build the corresponding REST endpoint of each micro service will be provided as application configuration information.

Table 28. Integration requirements of the Service Registry module

Inputs		
Internal/External	**Origin**	**Data**
External	DECIDE Framework Development / Integration module	Micro service endpoints

Outputs		
Internal/External	**Destination**	**Data**
Internal	Application	Proxy prefix

7.5.2.1.3 Monitoring

- **Application MCSLA monitoring**

The goal of this functionality is to monitor the application according to its defined (Multi-Cloud) SLA and to identify any related violations.

The Monitoring Engine will need the application's (MC)SLA as an input and will need to know, for each Service Level Objective (SLO), which is the monitoring source from which to retrieve metrics data. In some cases the metrics can be collected by the ADAPT components themselves, in other cases an application probe will need to be provided as part of the application and the monitoring source will indicate its endpoint.

The ADAPT Monitoring Engine will provide a REST API to get the monitored metrics and will provide alert notifications for each SLA violation to all registered clients.

Table 29. Integration requirements of the Monitoring Engine module

Inputs		
Internal/External	**Origin**	**Data**
External	DECIDE Framework	(MC)SLA
External	DECIDE Framework	Monitoring sources

Outputs		
Internal/External	**Destination**	**Data**
Internal / External	Registered clients	SLA violations

7.5.2.1.4 Monitoring

- **NFR monitoring**

The goal of this functionality is to monitor the violations of the Non Functional Requirements (NFR) established for the application during the design phase; the actual SLA monitoring will be done by ACSmI and ADAPT will receive the violations.

The ADAPT Monitoring Engine will provide alert notifications for each SLA violation to all registered clients and will provide a file to OPTIMUS with the violations/metrics registry to "optimize" the next simulation.

Table 30. Integration requirements of the Monitoring Engine module

Inputs		
Internal/External	**Origin**	**Data**
External	ACSmI	SLA Violations

Outputs		
Internal/External	**Destination**	**Data**
Internal / External	Registered clients	SLA violations
External	OPTIMUS	Report on NFR / SLA violations

7.5.2.1.5 Adaptation

- **Handle violations**

The goal of this functionality is to handle any violation raised by monitoring either the application MCSLA, or the CSPs' NFRs. Violation handling may lead both to alerting the operator and to contacting OPTIMUS to obtain a new configuration to re-deploy the application.

NFR / (MC)SLA violations are collected by the Monitoring Engine itself. To handle a violation according to requirements, the Monitoring Engine needs to know the level of technological risk of the application.

Violations for a low technological risk application will result in triggering the simulation of a new deployment configuration in OPTIMUS. The ADAPT Monitoring Engine will also alert the operator through some UI.

Table 31. Integration requirements of the Monitoring Engine module

Inputs		
Internal/External	**Origin**	**Data**
Internal	Monitoring Engine	NFR/(MC)SLA violations
External	OPTIMUS	Level of technological risk

Outputs		
Internal/External	**Destination**	**Data**
External	Operator GUI	SLA violation alert
External	OPTIMUS	Trigger new deployment configuration

7.5.2.1.6 Adaptation

- **Application adaptation / Application re-deployment**

The goal of this functionality is to adapt applications in response to violations. Application re-deployment will be the main strategy for enacting adaptation in response to violations. Re-deployment will be equivalent to the initial deployment, and Optional application tests can be provided in a specific container to assess the application readiness for being switched online to service client requests.

The input for application re-deployment is the same as that needed for initial application deployment: the information included in the Deployment Description section of the Application Description.

As for the initial deployment, output from the Deployment Orchestrator will be the exit status of the requested deployment operation.

Table 32. Integration requirements of the Deployment Orchestrator module

Inputs		
Internal/External	**Origin**	**Data**
External	Application Controller	Application Description
External	Application Controller	Application Containers
External	Application Controller	Deployment scripts
External	DECIDE Framework Development / Integration module	Optional Application tests

Outputs		
Internal/External	**Destination**	**Data**
External	Deployment method caller	Exit status

8 Conclusions

DECIDE is composed by different modules that need to interact among themselves. In order to achieve the proper integration of each component, their data needs to be analysed. This document has explored the data exchanges of each KR, both internal data exchanges and with other KRs.

The document has also focused on the needs of the DECIDE framework (KR1), to obtain the requirements it must fulfil to support the DECIDE workflow, the DevOps principles, and to be validated by the use cases.

Lastly, the appendix contains an analysis of DevOps, to understand its basic principles, and a comparison of tools to be integrated in the KR1.

The information included in this deliverable will serve as input for the DECIDE architecture, of which a first version will be found in deliverable D2.4 and a final version in deliverable D2.5.

It is important to remark that this document is the first part of a series of two. The second part, D2.2, will be released by month 23 and will update the present deliverable with details that will be available as the project progresses.

9 References

[1] I. Ul Haq and E. Schikuta, "Aggregation patterns of service level agreements," in *Proceedings of the 8th International Conference on Frontiers of Information Technology* , Islamabad, Pakistan, December 2010.

[2] *ISO 19086-1:2016. Information technology. Cloud computing. Service level agreement (SLA) framework. Part 1: Overview and concepts,* 2016.

[3] DECIDE Consortium, *DECIDE D5.1 - ACSmI requirements and technical design,* 2017.

[4] D. Chapman, Introduction to DevOps on AWS, 2014.

[5] New Relic, Navigating DevOps. What it is and why it matters to you and your business, New Relic Inc., 2017.

[6] Puppet, DORA, "2016 State of DevOps Report," 2016.

[7] G. Rushgrove, Get started with DevOps: A guide for IT managers, Puppet, 2016.

[8] R. Seroter, Exploring the entire DevOps Toolchain for (Cloud) Teams, InfoQ, 2014.

[9] DECIDE Consortium, *DECIDE, H2020 Proposal #731533,* 2016.

[10] Opensource, «What are microservices?,» [En línea]. Available: https://opensource.com/resources/what-are-microservices. [Último acceso: 2017].

[11] M. Rouse, "Techtarget. Microservices," January 2017. [Online]. Available: http://searchmicroservices.techtarget.com/definition/microservices. [Accessed April 2017].

Appendices

Appendix A. DevOps

DevOps is a portmanteau of "software *dev*elopment" and "IT *op*erations". It is used to refer to practices that motivate the collaboration, communication and integration between software developers and IT professionals while automating the process of software delivery and infrastructure changes. DevOps does not have a "formal" definition. It can be described as a philosophy, cultural change and paradigm shift, and has the goal of establishing an environment where building, testing and releasing software can happen rapidly, frequently and more reliably.

DevOps finds its origins in Enterprise Management Systems (EMS) and Agile development. The relationship with EMS comes from the fact that many of the people involved in the initial definition of DevOps were system administrators, who brought the main EMS best practices to DevOps, such as configuration management, system monitoring, automated provisioning and the toolchain approach.

The concept of DevOps is also closely related to Agile software development. Agile came as an alternative to the waterfall development methodology (where the whole set of requirements was defined before starting development work), and it emphasizes the need for collaboration between business users and developers. With Agile, the focus on software development shifted towards smaller and quicker releases, in order to respond faster to changes. This philosophy started to extend down the chain towards infrastructure and adopted the name DevOps. Where Agile was about collaboration between business users and developers, DevOps focuses on the collaboration between developers, operators and security teams. [4] [5]

A.1. Benefits of DevOps

Many reputable sources report important benefits achieved with DevOps. These reports should be interpreted with caution, since the benefits brought by adopting a DevOps approach vary according to the type of organization that implements them.

In any case, it seems that there are tangible benefits in applying DevOps. Some of these benefits, however, are not easy to measure. Saving employees time means giving them more space to think, experiment, be creative and innovate. The value of this increased freedom to innovate is hard to quantify, but it can be more significant than any cost savings.

Having said that, DevOps can also provide measurable benefits. According to a Puppet report, the potential savings from eliminating excess rework range from around $4M for small, high-performing organization, up to almost $4B for large medium-performing, organizations.

DevOps can also help to reduce downtime. Potential savings from this reason can reach $4.5M for medium performers.

The same report also mentions that high-performing DevOps organizations see deployments 200 more frequent, 24 times faster recovery times and three times lower change failure rates.

DevOps can also help to keep applications secure, with high performers spending up to 50% less time remediating security issues than low performers. [5] [6]

A.2. DevOps Principles

As mentioned before, DevOps is not a process or a standard. It can be defined as a culture, which consists in a series of principles or recommendations. These principles can be grouped under four main pillars: [7]

Culture: above all, DevOps means a cultural shift. It is not just a set of tools and practices, it is about establishing priorities and expectations, and how those priorities and expectations are pursued.

Automation: this is a key concept for DevOps. Everything that can be automated must be automated. Not having to worry about common and repetitive way frees time to dedicate to higher level work.

Measurement: feedback is an important part of agile and lean practices. Feedback is obtained by measuring, and with a DevOps-oriented mind, everything that moves in production should be measured. These measurements should be then shared with the widest possible audience.

Sharing: nowadays, organizations are complex and software teams are formed by people with different skills and specialized knowledge. These people must work together in order to be efficient, which is more easily achieved by sharing. As mentioned above, defining metrics and exposing them to everyone can be greatly beneficial for the organization.

These four pillars can be delved into, which results in the principles that should govern a DevOps approach:

1. Small but frequent updates

Small but frequent updates allow organizations to innovate faster for their clients. Generally, these updates are more incremental than the occasional updates done under traditional practices.

Smaller updates also reduce the risk on each implementation. They help teams to solve errors quicker, since these smaller updates allow them to identify the last implementation that provoked the error.

Even though the frequency and size of the updates vary, organizations that follow a DevOps model implement updates much more frequently than those that follow a traditional model.

2. Use microservices architecture

A microservices architecture divides big and complex systems into smaller and independent projects. Applications are divided in several individual components (services) and each service fulfils just one purpose or task and it is operated independently of the other services and the general application.

This architecture reduces the costs associated to update applications. When a service is assigned to small and agile team, organizations can move forward quickly.

3. Continuous integration

Continuous integration is software development practice that consists in periodically combining the changes in code in a central repository, after which versions and automatic tests are executed.

The main goals of the continuous integration are to improve software quality, to find and fix errors faster and to reduce the time it takes to validate and publish new software updates.

4. Continuous delivery

With continuous delivery, changes in code are created, tested and prepared automatically, to be delivered to the production phase. It widens the concept of continuous integration since it implements all changes in code on a test and/or production environment after the creation phase.

When continuous delivery is implemented correctly, developers will always have an artefact available for implementation, which has been subject to a standardized testing process.

5. Continuous deployment

The main goal of continuous deployment is to allow for the automatic deployment of production-ready code. This deployment will take place in a quick, frequent and reliable way.

Continuous deployment is closely related to continuous delivery. The main difference is that continuous deployment references production deployments.

6. Infrastructure automation (Infrastructure as code)

Infrastructure as code is a practice by which infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration.

This paradigm allows developers and administrators to manage infrastructure programmatically, as opposed to manually configure and adjust resources.

Thus, engineers can interact with infrastructure with code-based tools and treat said infrastructure similarly to how they treat application code. As they are defined by code, both infrastructure and servers can be implemented quickly with standardized patterns, updated with the latest revisions and versions or duplicated in a repeatable way.

7. Monitoring, use of registries, issue tracking

Organizations monitor metrics and logs to study how applications and infrastructure performance affects the experience that the final user has with their product.

By gathering, categorizing and analysing the data generated by applications and infrastructure, organizations can understand how changes and updates affect users, which provides information about the root cause of unexpected problems.

Active monitoring is becoming increasingly important, since services must be available 24/7 as update frequency increases.

Alert creation and real-time data analysis also helps organizations to monitor their services in a proactive way.

8. Communication and cooperation

The increase in communication and cooperation within an organization is one of the key cultural aspects in DevOps. The use of DevOps tools and the automation of the software delivery process promotes cooperation, since it physically gathers the workflows and responsibilities of the DevOps teams.

Furthermore, these teams stablish solid cultural rules that revolve around sharing information and facilitating communication, by means of chat applications, project and issue tracking systems and wikis.

This approach fosters the communication between development and operation teams, and even among other teams, such as marketing and sales. This allows all departments of the organization to better align with projects and goals.

9. Planning

Teams that plan common goals have a better comprehension of the dependencies, can see bottlenecks before they appear and can overcome priority conflicts. Regardless of whether a tool oriented to "blackboard" technologies, such as Kanban, is used or not, what matters the most is to leverage the living assets, not static plans.

Special attention should be paid to divide objects into manageable tasks that can be solved quickly. Static plans that are updated weekly and distributed by mail are not the best option. A better alternative would be to use shared planning tools, which allow to easily see each team member's progress in real time, and to hold conversations between different teams in a collaborative way.

10. Security

In a DevOps environment, attention to security is crucial. Infrastructure and company assets must be protected, and, when problems arise, they must be addressed quickly and effectively.

[4] [8]

A.3. Extended DevOps Principles

DECIDE action presents an approach that tries to extend the current known DevOps principles with the objective of covering the complete SDLC. The objective is to cover phases or stages that are specially needed in the context of multi-cloud software development life cycle. With this in mind, DECIDE extends the DevOps principles with these new ones:

1. Continuous architecting

The current DevOps principles cover the continuous delivery and continuous integration of the application, also the continuous update. Usually, these principles are focused on the implementation rather than on the initial design or architecting of the software. When dealing with complex multi-cloud applications, the design principles and architectural patterns to apply or use when designing the application are quite important. DECIDE action includes (and supports) this principle to support the first phases of the SDLC with the application of best practices and design principles for multi-cloud applications. [9]

2. Continuous pre-deployment

Before deploying multi-cloud applications (understood as applications whose components are split and deployed into different clouds) a deployment configuration selection activity has to be considered.

In a multi-cloud scenario the selection of the resources/services where to deploy the different components is a complex process and as such has to be considered in the SDLC and SOLC. That is the reason why DECIDE action includes the continuous pre-deployment as one of the principles of the extended DevOps approach we want to support. To be able to implement this new activity prior to actual deployment, DECIDE provides mechanisms to analyze alternative cloud deployment scenarios and their impact in the NFR of the application (e.g. security, performance), in the multi-cloud application SLA (MCSLA) as well as in the application costs, suggesting the developers and operators the best cloud deployment alternatives through the simulation of the behaviour of the application under stressful conditions and the cloud resources and cloud nodes. [9]

3. Continuous proactive adaptation

The classical DevOps principles include the continuous integration and continuous deployment. These principles have to do with the integration of changes/updates in the software and deploying these new versions of the software. It can be considered a type of adaptation triggered by the developer who wants to integrate updates and deploy them continuously.

DECIDE proposes to go one step further and to include a new phase/process in the SOLC: the continuous proactive adaptation. The continuous proactive adaptation comprises two main activities: on the one hand the monitoring of the multi-cloud application SLA (and indirectly the SLAs of the underlying cloud resources) and on the other hand the semi-automatic adaptation and redeployment into new cloud services when needed based on the assessment of this continuous monitoring. [9]

Appendix B. Tools analysis

B.1. Phase: Implementation

B.1.1. Tool Name: Eclipse

B.1.1.1 Description

Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins, including: Ada, ABAP, C, C++, COBOL, D, Fortran, Haskell, JavaScript, Julia, Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby (including Ruby on Rails framework), Rust, Scala, Clojure, Groovy, Scheme, and Erlang. It can also be used to develop documents with LaTeX (via a TeXlipse plug-in) and packages for the software Mathematica. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++, and Eclipse PDT for PHP, among others. Eclipse supports a rich selection of extensions, adding support for Python via pydev, Android development via Google's ADT, JavaFX via e(fx)clipse, JavaScript, jQuery, and many others at the Eclipse Marketplace. Valable is a Vala plug-in for Eclipse.

B.1.1.2 Open Source

Yes.

B.1.1.3 Operating System

Linux, MacOS, Solaris, Windows.

B.1.1.4 Free

Yes.

B.1.1.5 Licence

EPL.

B.1.1.6 Reference

https://www.eclipse.org/

B.1.1.7 Usability (ease of use)

Eclipse provides the Rich Client Platform (RCP) for developing general purpose applications.

B.1.1.8 Extendibility

The Eclipse Web Tools Platform (WTP) project is an extension of the Eclipse platform with tools for developing Web and Java EE applications.

Eclipse supports a rich selection of extensions, adding support for Python via pydev, Android development via Google's ADT, JavaFX via e(fx)clipse, JavaScript, jQuery, and many others at the Eclipse Marketplace. Valable is a Vala plug-in for Eclipse.

B.1.1.9 Functionality supported

IDE.

B.1.1.10 Languages supported

Ada , C/C++, D , Fortran, Groovy, Haskell, Java , JavaScript , Lua , Perl, PHP, Ruby, Scala , Tcl.

B.1.1.11 Language implemented

Java, ANSI C, C++,JSP, Bourne shell ,perl, php, sed.

B.1.1.12 Community behind it (in case of OS)

Eclipse Foundation, independent from IBM.

https://eclipse.org/org/foundation/

B.1.2. Tool Name: Netbeans

B.1.2.1 Description

NetBeans is a software development platform written in Java. The NetBeans Platform allows applications to be developed from a set of modular software components called modules. Applications based on the NetBeans Platform, including the NetBeans integrated development environment (IDE), can be extended by third party developers. The NetBeans IDE is primarily intended for development in Java, but also supports other languages, in particular PHP, C/C++ and HTML5.NetBeans is cross-platform and runs on Microsoft Windows, Mac OS X, Linux, Solaris and other platforms supporting a compatible JVM. The NetBeans Team actively support the product and seek feature suggestions from the wider community. Every release is preceded by a time for Community testing and feedback.

B.1.2.2 Open Source

Yes.

B.1.2.3 Operating System

Windows, Mac OS X, Linux, Solaris.

B.1.2.4 Free

Yes.

B.1.2.5 Licence

CCDL/GPL.

B.1.2.6 Reference

https://netbeans.org/

B.1.2.7 Functionality supported

The NetBeans Platform is a framework for simplifying the development of Java Swing desktop applications. The NetBeans IDE bundle for Java SE contains what is needed to start developing NetBeans plugins and NetBeans Platform based applications; no additional SDK is required.

B.1.2.8 Languages supported

D, Fortran, Groovy, Java, Javascript, Perl, Php, Python, Ruby, Scala.

B.1.2.9 Language implemented

Java.

B.1.2.10 Community behind it (in case of OS)

Oracle (Sun Microsystems).

B.2. Phase: Integration

B.2.1. Tool Name: Ant

B.2.1.1 Description

Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications. Ant can also be used effectively to build non-Java applications, for instance C or C++ applications. More generally, Ant can be used to pilot any type of process which can be described.

B.2.1.2 Open Source

Yes.

B.2.1.3 Operating System

Cross platform.

B.2.1.4 Free

Yes.

B.2.1.5 Licence

Apache 2.0.

B.2.1.6 Reference

http://ant.apache.org/

B.2.1.7 Functionality supported

Automatic Software Built processes.

B.2.1.8 Languages supported

Java.

B.2.1.9 Language implemented

Java, XML.

B.2.1.10 Community behind it (in case of OS)

Apache Software Foundation.

B.2.2. Tool Name: Maven

B.2.2.1 Description

Apache Maven is a build automation tool used primarily for Java projects. Maven addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies. Contrary to preceding tools like Apache Ant, it uses conventions for the build procedure, and only exceptions need to be written down. An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins. It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven 2 Central Repository, and stores them in a local cache. This local cache of downloaded artefacts can also be updated with artefacts created by local projects. Public repositories can also be updated. Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other languages. The Maven project is hosted by the Apache Software Foundation.

B.2.2.2 Open Source

Yes.

B.2.2.3 Operating System

Cross platform.

B.2.2.4 Free

Yes.

B.2.2.5 Licence

Apache 2.0.

B.2.2.6 Reference

http://maven.apache.org/

B.2.2.7 Functionality supported

Automatic Software Built processes and Project Management.

B.2.2.8 Languages supported

Java.

B.2.2.9 Language implemented

Java.

B.2.2.10 Community behind it (in case of OS)

Apache Software Foundation.

B.2.3. Tool Name: GitLab

B.2.3.1 Description

GitLab is a web-based Git repository manager with wiki and issue tracking features, using an open source license, developed by GitLab Inc. The software was written by Dmitriy Zaporozhets and Valery Sizov from Ukraine. The code is written in Ruby. Later, some parts have been rewritten in Go. As of December 2016, the company has 150 team members and more than 1400 open source contributors. It is used by organisations such as IBM, Sony, Jülich Research Center, NASA, Alibaba, Invincea, O'Reilly Media, Leibniz-Rechenzentrum (LRZ) and CERN.

B.2.3.2 Open Source

Yes.

B.2.3.3 Operating System

Cross platform.

B.2.3.4 Free

Commercial.

B.2.3.5 Licence

Open Core, Commercial.

B.2.3.6 Reference

http://about.gitlab.com

B.2.3.7 Functionality supported

Web-based Git repository manager.

B.2.3.8 Language implemented

Ruby, Go.

B.2.3.9 Community behind it (in case of OS)

GitLab Inc.

B.2.4. Tool Name: Subversion

B.2.4.1 Description

Apache Subversion (often abbreviated SVN, after its command name svn) is a software versioning and revision control system distributed as open source under the Apache License. Software developers use Subversion to maintain current and historical versions of files such as source code, web pages, and documentation. Its goal is to be a mostly compatible successor to the widely used Concurrent Versions System (CVS). The open source community has used Subversion widely: for example in projects such as Apache Software Foundation, Free Pascal, FreeBSD, GCC, Mono and SourceForge. CodePlex offers access to Subversion as well as to other types of clients. Subversion was created by CollabNet Inc. in 2000, and is now a top-level Apache project being built and used by a global community of contributors.

B.2.4.2 Open Source

Yes.

B.2.4.3 Operating System

Cross platform.

B.2.4.4 Free

Yes.

B.2.4.5 Licence

Apache 2.0.

B.2.4.6 Reference

http://subversion.apache.org/

B.2.4.7 Functionality supported

Software versioning and revision control system.

B.2.4.8 Languages supported

Python, Perl, Java, Ruby.

B.2.4.9 Language implemented

C.

B.2.4.10 Community behind it (in case of OS)

Apache Software Foundation.

B.2.5. Tool Name: Git

B.2.5.1 Description

Git is a version control system (VCS) for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for software development, but it can be used to keep track of changes in any files. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows. Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Its current maintainer since 2005 is Junio Hamano. As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version tracking abilities, independent of network access or a central server. Like the Linux kernel, Git is free software distributed under the terms of the GNU General Public License version 2.

B.2.5.2 Open Source

Yes.

B.2.5.3 Operating System

POSIX: Linux, Windows, macOS, Solaris, BSD.

B.2.5.4 Free

Yes.

B.2.5.5 Licence

GNU GPL v2.

B.2.5.6 Reference

https://git-scm.com/

B.2.5.7 Functionality supported

Software versioning and revision control system.

B.1.2.1. Languages supported

Java (JGit), JScript (JS-Git), Ruby, Python, and Haskell.

B.1.2.2. Language implemented

C, Shell, Perl, Tcl , Python.

B.1.2.3. Community behind it (in case of OS)

Hosted on GitHub, responsible developer Junio Hamano.

B.2.6. Tool Name: Jenkins

B.2.6.1 Description

Jenkins is an open source automation server written in Java. Jenkins helps to automate the non-human part of the whole software development process, with now common things like continuous integration, but by further empowering teams to implement the technical part of a Continuous Delivery. It is a server-based system running in a servlet container such as Apache Tomcat. It supports SCM tools including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase and RTC, and can execute Apache Ant, Apache Maven and sbt based projects as well as arbitrary shell scripts and Windows batch commands. The creator of Jenkins is Kohsuke Kawaguchi. Released under the MIT License, Jenkins is free software. Builds can be triggered by various means, for example by commit in a version control system, by scheduling via a cron-like mechanism and by requesting a specific build URL. It can also be triggered after the other builds in the queue have completed. Jenkins functionality can be extended with plugins.

B.2.6.2 Open Source

Yes.

B.2.6.3 Operating System

Cross platform.

B.2.6.4 Free

Yes.

B.2.6.5 Licence

MIT.

B.2.6.6 Reference

https://jenkins.io/

B.2.6.7 Extendibility

Plug-ins.

B.2.6.8 Functionality supported

Continuous integration Tool.

B.2.6.9 Languages supported

Android, C/C++, Java, Python, Ruby.

B.2.6.10 Language implemented

Java.

B.2.6.11 Community behind it (in case of OS)

Jenkins Community.

B.2.7. Tool Name: Docker

B.2.7.1 Description

Docker is an open-source project that automates the deployment of applications inside software containers. Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in. Docker provides an additional layer of abstraction and automation of operating-system-level virtualization on Windows and Linux. Docker uses the resource isolation features of the Linux kernel such as cgroups and kernel namespaces, and a union-capable file system such as OverlayFS and others to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.

The Linux kernel's support for namespaces mostly isolates an application's view of the operating environment, including process trees, network, user IDs and mounted file systems, while the kernel's cgroups provide resource limiting, including the CPU, memory, block I/O, and network. Since version 0.9, Docker includes the libcontainer library as its own way to directly use virtualization facilities provided by the Linux kernel, in addition to using abstracted virtualization interfaces via libvirt, LXC (Linux Containers) and systemd-nspawn.

B.2.7.2 Open Source

Yes.

B.2.7.3 Operating System

Linux, Windows.

B.2.7.4 Free

Yes.

B.2.7.5 Licence

Apache 2.0.

B.2.7.6 Reference

www.docker.com

B.2.7.7 Functionality supported

Automates the deployment of applications inside software containers.

B.2.7.8 Language implemented

Java.

B.2.7.9 Community behind it (in case of OS)

Apache Software Foundation.

B.2.8. Tool Name: Portainer

B.2.8.1 Description

Portainer is a simple management solution for Docker. Portainer is an open-source lightweight management UI which allows you to easily manage your Docker host or Swarm cluster

It consists of a web UI that allows you to easily manage your Docker containers, images, networks and volumes. Portainer provides a detailed overview of Docker and allows you to manage containers, images, networks and volumes. It is easy to deploy, one Docker command away from running Portainer anywhere. It is compatible with the *standalone Docker* engine and with *Docker Swarm*.

Portainer adds a security layer on top of Docker with authentication, multiple user management and the ability to define restrict access to some resources.

B.2.8.2 Open Source

Yes.

B.2.8.3 Operating System

Linux, Windows,iOs.

B.2.8.4 Free

Yes.

B.2.8.5 Licence

Apache 2.0.

B.2.8.6 Reference

http:www.portainer.io

B.2.8.7 Functionality supported

It manages Docker containers, images, networks and volumes via Dashboard.

It allows to deploy containers from a template list that includes: MySQL, MariaDB, PostgreSQL, Drupal, Jenkins, Odoo, Redmine, Magento.

B.2.8.8 Language implemented

Go.

B.2.8.9 Community behind it (in case of OS)

Community (Gitter, Slack) and Commercial Support.

B.2.9. Tool Name: Nagios Core

B.2.9.1 Description

Nagios is a free and open source computer-software application that monitors systems, networks and infrastructure. Nagios offers monitoring and alerting services for servers, switches, applications and services. It alerts users when things go wrong and alerts them a second time when the problem has been resolved. Ethan Galstad and a group of developers originally wrote Nagios as NetSaint. As of 2015 they actively maintain both the official and unofficial plugins.The Linux kernel's support for namespaces mostly isolates an application's view of the operating environment, including process trees, network, user IDs and mounted file systems, while the kernel's cgroups provide resource limiting, including the CPU, memory, block I/O, and network. Since version 0.9, Docker includes the libcontainer library as its own way to directly use virtualization facilities provided by the Linux kernel, in addition to using abstracted virtualization interfaces via libvirt, LXC (Linux Containers) and systemd-nspawn.

B.2.9.2 Open Source

Yes.

B.2.9.3 Operating System

Cross Platform.

B.2.9.4 Free

Yes.

B.2.9.5 Licence

GNU GPL V2.

B.2.9.6 Reference

www.nagios.org

B.2.9.7 Extendibility

Plugin.

B.2.9.8 Functionality supported

Network Monitoring.

B.2.9.9 Language implemented

C.

B.2.9.10 Community behind it (in case of OS)

Nagios Plugin Team.

B.2.10. Tool Name: Pfsense

B.2.10.1 Description

PfSense is an open source firewall/router computer software distribution based on FreeBSD. It is installed on a physical computer or a virtual machine to make a dedicated firewall/router for a network and has been noted for its reliability and offering a range of features. It can be configured and upgraded through a web-based interface, and requires no knowledge of the underlying FreeBSD system to manage. pfSense is commonly deployed as a perimeter firewall, router, wireless access point, DHCP server, DNS server, and as a VPN endpoint. pfSense supports installation of third-party packages like Snort or Squid through its Package Manager. As of 2016 pfSense is described by servethehome.com as the "gold standard" for open source network appliances in its buyer guides.

B.2.10.2 Open Source

Yes.

B.2.10.3 Operating System

Cross Platform, can be installed on hardware with x86 or x86-64 architecture.

B.2.10.4 Free

Yes.

B.2.10.5 Licence

Apache 2.0.

B.2.10.6 Reference

https://www.pfsense.org/

B.2.10.7 Functionality supported

Berkeley Software Distribution.

B.2.10.8 Language implemented

C.

B.2.10.9 Community behind it (in case of OS)

Electric Sheep Fencing LLC (ESF).

B.2.11. Tool Name: Zabbix

B.2.11.1 Description

Zabbix is enterprise open source monitoring software for networks and applications, created by Alexei Vladishev. It is designed to monitor and track the status of various network services, servers, and other network hardware. Zabbix uses MySQL, PostgreSQL, SQLite, Oracle or IBM DB2 to store data. Its backend is written in C and the web frontend is written in PHP. Zabbix offers several monitoring options: Simple checks can verify the availability and responsiveness of standard services such as SMTP or HTTP without installing any software on the monitored host. A Zabbix agent can also be installed on UNIX and Windows hosts to monitor statistics such as CPU load, network utilization, disk space, etc. As an alternative to installing an agent on hosts, Zabbix includes support for monitoring via SNMP, TCP and ICMP checks, as well as over IPMI, JMX, SSH, Telnet and using custom parameters. Zabbix supports a variety of near-real-time notification mechanisms, including XMPP. As an alternative to installing an agent on hosts, Zabbix includes support for monitoring via SNMP, TCP and ICMP checks, as well as over IPMI, JMX, SSH, Telnet and using custom parameters. Zabbix supports a variety of near-real-time notification mechanisms, including XMPP.

B.2.11.2 Open Source

Yes.

B.2.11.3 Operating System

Cross Platform.

B.2.11.4 Free

Yes.

B.2.11.5 Licence

GNU GPL V2.

B.2.11.6 Reference

https://www.zabbix.org/

B.2.11.7 Functionality supported

Network Monitoring.

B.2.11.8 Language implemented

C.

B.2.11.9 Community behind it (in case of OS)

Zabbix and Zabbix Comunity.

B.2.12. Tool Name: Sonarqube

B.2.12.1 Description

SonarQube (formerly Sonar) is an open source platform for continuous inspection of code quality. Supports languages: Java (including Android), C/C++, Objective-C, C#, PHP, Flex, Groovy, JavaScript, Python, PL/SQL, COBOL, Swift, etc.

B.2.12.2 Open Source

Yes.

B.2.12.3 Operating System

Cross Platform.

B.2.12.4 Free

Yes.

B.2.12.5 Licence

GNU GPL.

B.2.12.6 Reference

www.sonarqube.org

B.2.12.7 Extendibility

Integrates with Eclipse, Visual Studio and IntelliJ IDEA development environments through the SonarLint plugins , Integrates with external tools: LDAP, Active Directory, GitHub.

B.2.12.8 Functionality supported

Source Code Testing.

B.2.12.9 Languages supported

Java (including Android), C/C++, Objective-C, C#, PHP, Flex, Groovy, JavaScript, Python, PL/SQL, COBOL, Swift.

B.2.12.10 Language implemented

Java, Ruby.

B.2.12.11 Community behind it (in case of OS)

Sonarsource (https://www.sonarsource.com/).

B.3. Phase: Testing

B.3.1. Tool Name: Junit

B.3.1.1 Description

JUnit is a set of libraries created by Erich Gamma and Kent Beck that are used in programming to do unit tests of Java applications. JUnit is a set of classes (framework) that allows to execute the execution of Java classes in a controlled way, to be able to evaluate if the operation of each one of the methods of the class behaves as expected. That is, depending on some input value the expected return value is evaluated; If the class meets the specification, then JUnit will return that the class

method successfully passed the test; In case the expected value is different than the method returned during execution, JUnit will return a failure in the corresponding method. JUnit is also a means of controlling the regression tests needed when a part of the code has been modified and it is desired to see that the new code meets the above requirements and that its functionality has not been altered after the new modification. The framework itself includes ways to see the results (runners) that can be in text, graphic (AWT or Swing) or as an Ant task. Currently development tools such as NetBeans and Eclipse have plug-ins that allow the generation of the necessary templates for the creation of the tests of a Java class is done automatically, making it easier for the programmer to focus on the test and the expected result, and leaving to the tool the creation of the classes that allow to coordinate the tests.

B.3.1.2 Open Source

Yes.

B.3.1.3 Operating System

Cross platform.

B.3.1.4 Free

Yes.

B.3.1.5 Licence

GPL.

B.3.1.6 Reference

http://junit.sourceforge.net/

B.3.1.7 Extendibility

JUnit alternatives have been written in other languages including:

Actionscript (FlexUnit), Ada (AUnit), C (CUnit), C# (NUnit), C++ (CPPUnit, CxxTest), Coldfusion (MXUnit), Erlang (EUnit), Eiffel (Auto-Test) - JUnit inspired getest (from Gobosoft), which led to Auto-Test in Eiffel Studio.

Fortran (fUnit, pFUnit), Delphi (DUnit), Free Pascal (FPCUnit), Haskell (HUnit), JavaScript (JSUnit), Microsoft .NET (NUnit), Objective-C (OCUnit), OCaml (OUnit) Perl (Test::Class and Test::Unit), PHP (PHPUnit), Python (PyUnit), Qt (QTestLib) , R (RUnit), Ruby (Test::Unit).

B.3.1.8 Functionality supported

Unit testing framework.

B.3.1.9 Languages supported

Java (see extensions).

B.3.1.10 Language implemented

Java.

B.3.1.11 Community behind it (in case of OS)

http://junit.org/junit4/

B.3.2. Tool Name: Mockito

B.3.2.1 Description

Mockito is an open source testing framework for Java released under the MIT License. The framework allows the creation of test double objects (mock objects) in automated unit tests for the purpose of Test-driven Development (TDD) or Behaviour Driven Development (BDD).In software development there is an opportunity of ensuring that objects perform the behaviours that are expected of them. One approach is to create a test automation framework that actually exercises each of those behaviours and verifies that it performs as expected, even after it is changed. However, the requirement to create an entire testing framework is often an onerous task that requires as much effort as writing the original objects that were supposed to be tested. For that reason, developers have created mock testing frameworks. These effectively fake some external dependencies so that the object being tested has a consistent interaction with its outside dependencies. Mockito intends to streamline the delivery of these external dependencies that are not subjects of the test. A study performed in 2013 on 10,000 GitHub projects found that Mockito is the 9th most popular Java library.

B.3.2.2 Open Source

Yes.

B.3.2.3 Operating System

Cross platform.

B.3.2.4 Free

Yes.

B.3.2.5 Licence

MIT.

B.3.2.6 Reference

https://github.com/mockito/mockito

B.3.2.7 Functionality supported

Testing Framework for Java.

B.3.2.8 Languages supported

Java.

B.3.2.9 Language implemented

Java.

B.3.2.10 Community behind it (in case of OS)

http://site.mockito.org/

B.3.3. Tool Name: Arquillian

B.3.3.1 Description

Arquillian Eclipse is a JBoss Tools component that makes it simple to create and maintain Arquillian tests. It provides adding Arquillian artifacts to a project as well as creating, validating and running Arquillian JUnit tests. The mission of the Arquillian project is to provide a simple test harness that developers can use to produce a broad range of integration tests for their Java applications (most likely enterprise applications). A test case may be executed within the container, deployed alongside the code under test, or by coordinating with the container, acting as a client to the deployed code. Arquillian defines two styles of container, remote and embedded. A remote container resides in a separate JVM from the test runner. Its lifecycle may be managed by Arquillian, or Arquillian may bind to a container that is already started. An embedded container resides in the same JVM and is mostly likely managed by Arquillian. Containers can be further classified by their capabilities. Examples include a fully compliant Java EE application server (e.g., GlassFish, JBoss AS, Embedded GlassFish), a Servlet container (e.g., Tomcat, Jetty) and a bean container (e.g., Weld SE). Arquillian ensures that the container used for testing is pluggable, so the developer is not locked into a proprietary testing environment. At the core, Arquillian provides a custom test runner for JUnit and TestNG that turns control of the test execution lifecycle from the unit testing framework to Arquillian. From there, Arquillian can delegate to service providers to setup the environment to execute the tests inside or against the container. An Arquillian test case looks just like a regular JUnit or TestNG test case with two declarative enhancements. Since Arquillian works by replacing the test runner, Arquillian tests can be executed using existing test IDE, Ant and Maven test plugins without any special configuration. Test results are reported just like you would expect. Using Arquillian is no more complicated than basic unit testing. Three aspects of an Arquillian test case are: Container: a runtime environment for a deployment. Deployment: the process of dispatching an artifact to a container to make it operational. Archive: a packaged assembly of code, configuration and resources.

B.3.3.2 Open Source

Yes.

B.3.3.3 Operating System

Cross platform.

B.3.3.4 Free

Yes.

B.3.3.5 Licence

Apache 2.0.

B.3.3.6 Reference

http://arquillian.org/

B.3.3.7 Functionality supported

Testing of Java microservices.

B.3.3.8 Languages supported

Java.

B.3.3.9 Language implemented

Java.

B.3.3.10 Community behind it (in case of OS)

Arquillian Community.

B.3.4. Tool Name: TestNG

B.3.4.1 Description

TestNG is a testing framework for the Java programming language created by Cédric Beust and inspired by JUnit and NUnit. The design goal of TestNG is to cover a wider range of test categories: unit, functional, end-to-end, integration, etc., with more powerful and easy-to-use functionalities.

B.3.4.2 Open Source

Yes.

B.3.4.3 Operating System

Cross platform.

B.3.4.4 Free

Yes.

B.3.4.5 Licence

Apache 2.0.

B.3.4.6 Reference

http://testng.org/doc/

B.3.4.7 Functionality supported

Testing Platform for Java.

B.3.4.8 Languages supported

Java.

B.3.4.9 Language implemented

Java.

B.3.4.10 Community behind it (in case of OS)

TestNG.org

B.4. Comparison table

Tool Name	Open Source	Operating System	Free	License	Usability (ease of use)	Functionality Supported	Languages Supported	Language Implemented
Eclipse	Yes	Linux, MacOS, Solaris, Windows	Yes	EPL	Eclipse provides the Rich Client Platform (RCP) for developing general purpose applications.	IDE	Ada , C/C++, D , Fortran, Groovy, Haskell, Java , JavaScript , Lua , Perl, PHP, Ruby, Scala , Tcl.	Java, ANSI C, C++,JSP, Bourne shell ,perl, php, sed.
Netbeans	Yes	Windows, Mac OS X, Linux, Solaris	Yes	CCDL/GPL		The NetBeans Platform is a framework for simplifying the development of Java Swing desktop applications. The NetBeans IDE bundle for Java SE contains what is needed to start developing NetBeans plugins and NetBeans Platform based applications; no additional SDK is required.	D, Fortran, Groovy, Java, Javascript, Perl, Php, Python, Ruby, Scala	Java
Ant	Yes	Cross platform	Yes	Apache 2.0		Automatic Software Built processes	Java	Java, XML
Maven	Yes	Cross platform	Yes	Apache 2.0		Automatic Software Built processes and Project Management	Java	Java
Gitlab	Yes	Cross platform	Commercial and Community	MIT		Web-based Git repository manager		Ruby, Go

Tool Name	Open Source	Operating System	Free	License	Usability (ease of use)	Functionality Supported	Languages Supported	Language Implemented
Subversion	Yes	Cross platform	Yes	Apache 2.0		Software versioning and revision control system	Python, Perl, Java, Ruby	C
Git	Yes	POSIX: Linux, Windows, macOS, Solaris, BSD	Yes	GNU GPL v2		Software versioning and revision control system	Java (JGit) , JScript (JS-Git) , Ruby, Python, and Haskell	C, Shell, Perl, Tcl, Python
Jenkins	Yes	Cross platform	Yes	MIT		Continuous integration Tool	Android, C/C++, Java, Python, Ruby	Java
Docker	Yes	Linux, Windows, Macos	Yes	Apache 2.0		Automates the deployment of applications inside software containers		Java
Portainer	Yes	Cross platform	Yes	Apache 2.0		It manages Docker containers, images, networks and volumes via Dashboard. It allows to deploy containers from a template list that includes MySQL, MariaDB, PostgreSQL, Drupal, Jenkins, Odoo, Redmine, Magento		Go
Nagios Core	Yes	Cross platform	Yes	GNU GPL V2		Network Monitoring		C

Tool Name	Open Source	Operating System	Free	License	Usability (ease of use)	Functionality Supported	Languages Supported	Language Implemented
Pfsense	Yes	Cross Platform, can be installed on hardware with x86 or x86-64 architecture	Yes	Apache 2.0		Opensource firewall/router software		FreeBSD
Zabbix	Yes	Cross platform	Yes	GNU GPL V2		Network Monitoring		C
sonarqube	Yes	Cross platform	Yes	GNU GPL		Source Code Testing	Java (including Android), C/C++, Objective-C, C#, PHP, Flex, Groovy, JavaScript, Python, PL/SQL, COBOL, Swift.	Java, ruby
Junit	Yes	Cross platform	Yes	GPL		Unit testing framework		Java
Mockito	Yes	Cross platform	Yes	MIT		Testing Framework for Java	Java	Java
Arquillian	Yes	Cross platform	Yes	Apache 2.0		Testing Framework	Java	Java
TestNG	Yes	Cross platform	Yes	Apache 2.0		Testing Platform for Java	Java	Java