SeaClouds Project

D5.1.2 – Integrated Platform

Project Acronym	SeaClouds
Project Title	Seamless adaptive multi-cloud management of service-based applications
Call identifier	FP7-ICT-2012-10
Grant agreement no.	610531
Start Date	1st October 2013
Ending Date	31st March 2016

Work Package	WP5 Integration, infrastructure delivery and GUI
Deliverable code	D5.1.2
Deliverable Title	Final design of the User Interface
Nature	Integrated Platform
Dissemination Level	Public
Due Date:	M19
Submission Date:	8th of May 2015
Version:	1.0
Status	Final
Author(s):	Elisabetta Di Nitto (Polimi), Román Sosa (ATOS), Marc Oriol (UPI), Simone Zenzaro (UPI), Javier Cubo (UMA), Andrea Turli (Cloudsoft), Diego Pérez (Polimi), Dionysis Athanasopoulos (Polimi), Jose Carrasco (UMA)
Reviewer(s)	Ernesto Pimentel (UMA), Francesco D'Andria (ATOS)

Dissemination Level

Project co-funded by the European Commission within the Seventh Framework Programme		
	Public	X
	Restricted to other programme participants (including the Commission)	
	Restricted to a group specified by the consortium (including the Commission)	
	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	8/04/2015	ToC and distribution of work	Elisabetta Di Nitto
0.2	17/04/2015	Section 2, 3 and 4 mainly, and initial version in Sections 5 and 6	Diego Pérez, Elisabetta Di Nitto, Román Sosa, Marc Oriol, Javier Cubo, Andrea Turli, Dionysis Athanasopoulos
0.3	22/04/2015	Added SLA Agreement	Román Sosa
0.4	23/04/2015	Updates to Sections 2, 3 and 5. Writing of Section 7	Elisabetta Di Nitto
0.5	24/04/2015	Updates to Section 5	Marc Oriol, Jose Carrasco
0.6	26/04/2015	Revision of the whole document	Elisabetta Di Nitto
0.7	28/04/2015	Revision of Executive Summary, Sections 3 and 5, Modifications in the example application	Javier Cubo, Jose Carrasco, Diego Perez, Marc Oriol, Simone Zenzaro, Dionysis Athanasopoulos
0.8	28/04/2015	Updates to Section 5.4	Román Sosa
0.9	29/04/2015	Checking and suggestions in Sections 2, 3, 5, creation of Sections 1 and 8, and revision of Section 1	Javier Cubo, Elisabetta Di Nitto, Diego Pérez, Marc Oriol, Simone Zenzaro, Jose Carrasco
0.10	30/04/2015	Updates to Section 5	Elisabetta Di Nitto, Diego Perez
0.11	04/05/2015	Final review by the authors	Elisabetta Di Nitto, Marc Oriol, Diego Perez, Javier Cubo
0.12	06/05/2015	Comments by reviewers	Ernesto Pimentel, Francesco D'Andria, Javier Cubo
1.0	08/05/2015	Final changes based on reviewers' comments	Elisabetta Di Nitto, Diego Perez

Table of Contents

List of Figures

List of Tables

Executive Summary

This deliverable is the first integrated platform developed within the SeaClouds project. This document aims at accompanying the software prototype by offering information about: i) the released and integrated components and their interactions; ii) the way the resulting integrated platform can be installed by a user; iii) the way a user can exploit such platform to compile, starting from an Abstract Application Model (AAM), the Deployable Application Model (DAM), and can then deploy, monitor and check the SLA (Service Level Agreement) of an example application. This document provides also an updated version of the integration plan originally defined in Deliverable D5.1.1.

1 Introduction

1.1 Scope and outcome of the Deliverable

This deliverable is constituted by the first version of the SeaClouds Integrated Platform plus the following elements:

1. This accompanying document that aims at guiding users of the SeaClouds Integrated Platform through the identification of the main components of the current platform, the relationships between them and through their download, deployment, installation and execution phases.
2. The artifacts that are needed to deploy and execute the platform (see Section 4).
3. An application example, together with the artifacts needed to describe it (AAM and ADP in the TOSCA specification, and DAM in the TOSCA and CAMP format), the cloud offers that match the application, the associated monitoring rules and the SLA service (see Section 5).

Since the project is pursuing a fully open source approach, all software is released with an Apache 2.0 license and has been made available since the beginning of the development on github. Thus, all above material can be downloaded from the SeaClouds Platform github repository https://github.com/SeaCloudsEU. The current document is made available on the same repository and will be continuously updated to constitute a live documentation while the SeaClouds Platform will be evolved.

The SeaClouds Integrated Platform follows the architecture defined in Deliverable D2.4 [1] with some simplifications that will be addressed in the next releases (see Sections 3 and 7).

1.2 Structure of the document

This document has the following structure:

- Section 2 describes the current status of the SeaClouds Integrated Platform in comparison with the integration plan foreseen at M8, and also mentioning the first prototype developed at M12.
- Section 3 describes the components that currently belong to the SeaClouds Integrated Platform and the interaction among them.
- Section 4 provides an overview of the procedure to be followed in order to download, deploy and run the SeaClouds Integrated Platform.
- Section 5 shows how the user can exploit the Integrated Platform to deploy an example application.
- Section 6 presents the tools and practices for continuous integration and quality assurance that we have followed for delivering the SeaClouds Integrated Plaform.

- Section 7 presents the updated integration plan that will lead us toward the finalization of the Integrated Platform.
- Finally, Section 8 provides some conclusions.

1.3 List of Acronyms

Here we list the different acronyms that will be used in this document.

Acronym	Definition
SaaS	Software-as-a-Service
PaaS	Platform-as-a-Service
IaaS	Infrastructure-as-a-Service
QoS	Quality of Service
QoB	Quality of Business
SLA	Service Level Agreement
GUI	Graphical User Interface
API	Application Programming Interface
AAM	Abstract Application Model
DAM	Deployable Application Model
ADP	Abstract Deployment Plan
URI	Uniform Resource Identifier
YAML	YAML Ain't Markup Language
XML	eXtensible Markup Language
REST	Representation state transfer
LAM	Live Application Model

Table 1: Acronyms

2 Foreseen integration plan at M8 and current situation

The current section shortly provides an overview of the history of the SeaClouds Integrated Platform from the initial integration plan defined at M8 to the first proof of concept demonstrated during the review at M12 and described in D3.1 [2] and D4.1 [3] (with a general overview also presented in D5.4.1 [4]) to the current situation at M19.

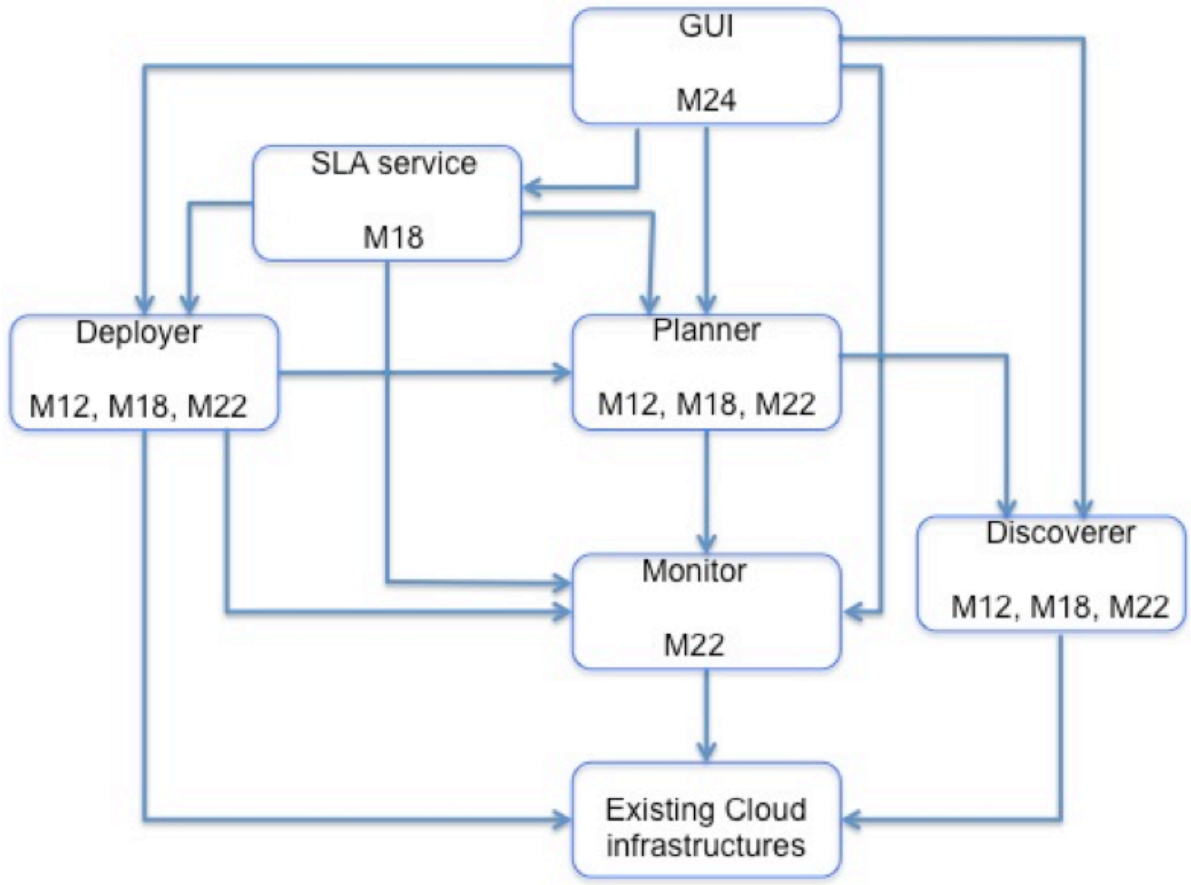

Figure 1: Release and integration plan defined at M8

Figure 1 describes the SeaClouds release and integration plan as it has been defined at Project Month M8 in Deliverable D5.1.1 [5]. According to the plan, at M18, the integrated platform should have included the following components:

- The *Discoverer* in charge of supporting the discovery of cloud services to be used in the resource allocation phase.
- The *Planner* supporting the design of a multi-cloud application and the definition of the corresponding resource allocation plan.
- The *Deployer* in charge of deploying the application according to the allocation plan defined by the Planner.
- The *SLA Service* in charge of verifying the fulfillment of Service Levels Agreements.

With respect to this original plan we have decided to:

- Give priority to the GUI runtime with respect to the Discoverer as the GUI is needed to support users in properly managing the lifecycle of a multi-cloud application. The

SeaClouds *GUI* is divided in two parts, the design-time user interface dedicated to support the *design of an application topology* and the *Dashboard* supporting the management of deployment and the interaction with the runtime components. The current SeaClouds Integrated Platform includes the Dashboard while the design-time GUI is under development.

- Offer a full-fledged *Monitoring Platform* in order to prepare the stage for the next step in the project that concerns the development of the reconfiguration and replanning actions presented in Deliverables D4.3 [6] and D4.4 [7]. To this end, the first proof of concept prototype we demonstrated during the review was already including the simple *Monitor* component presented in D4.1. The current SeaClouds Platform encapsulates the monitoring platform developed as part of another European project, MODAClouds (www.modaclouds.eu). As already discussed in other deliverables, this platform supports monitoring in a multi-cloud context. It allows collection of data to be customized depending on the specific application at hands. Moreover it offers a monitoring rule language that supports the definition of conditions on the monitoring data that can indicate the presence of a problem and the implementation of reaction actions in response to this situation.

As mentioned above, initial versions of the *Planner*, *Deployer*, *and Monitor* supported by the initial design and implementation of the *Dashboard* have been already presented during M12 review, also with an initial version of the *SLA Service* component. The SeaClouds Integrated Platform includes consolidated versions of all these components. The functionality offered by this platform is described in the next section together with the corresponding components.

3 Integrated SeaClouds components and interaction among them

3.1 Overview

The currently available SeaClouds Integrated Platform offers the following functionality:

- *Planning functionality*, that is, starting from an Abstract Application Model (AAM), which at the moment is assumed to be defined outside the platform, the Planner is able to define a Deployable Application Model (DAM) that includes information on the optimal cloud resources to be used for the application at hands.
- *Deployment functionality*, that is, based on the DAM, the Deployer is able to install and run the application exploiting some cloud resources. At the moment the resources available are of IaaS kind while in the next release of the platform we will integrate the mechanisms to support interaction with PaaS. Together with the application, the Deployer installs, when needed, the Data Collectors components that support the monitoring activity.

- *Monitoring functionality*, that is, based on monitoring rules that are assumed, for the moment, to be generated by the application designer or operator, the Monitoring Platform is able to connect to the deployed Data Collectors and is able to acquire data and reason on them. In the next release of the platform the monitoring rules will be automatically generated by the Planner based on the Quality of Service constraints the developer will incorporate into the AAM and on the actual cloud resources that will be selected for the application.
- *SLA management functionality*, that is, based on an SLA that is assumed, for the moment, to be manually generated by the designer, the SLA Service is able to connect to the Monitoring Platform and, through it, it is able to receive the monitoring information needed to identify SLA violations. Such violations are shown through the Dashboard.

The whole set of functionality is orchestrated by the Dashboard as highlighted in Figure 2. This figure simplifies the final architecture presented in D2.4 as it describes the situation of the current integrated platform.

The following subsections describe each of the components of the M19 Integrated Platform highlighting the current dependences to other components (new dependencies will be implemented in the next version of the platform), the external libraries being used, the license associated to the component, the repository, and the kind of API offered by the component.

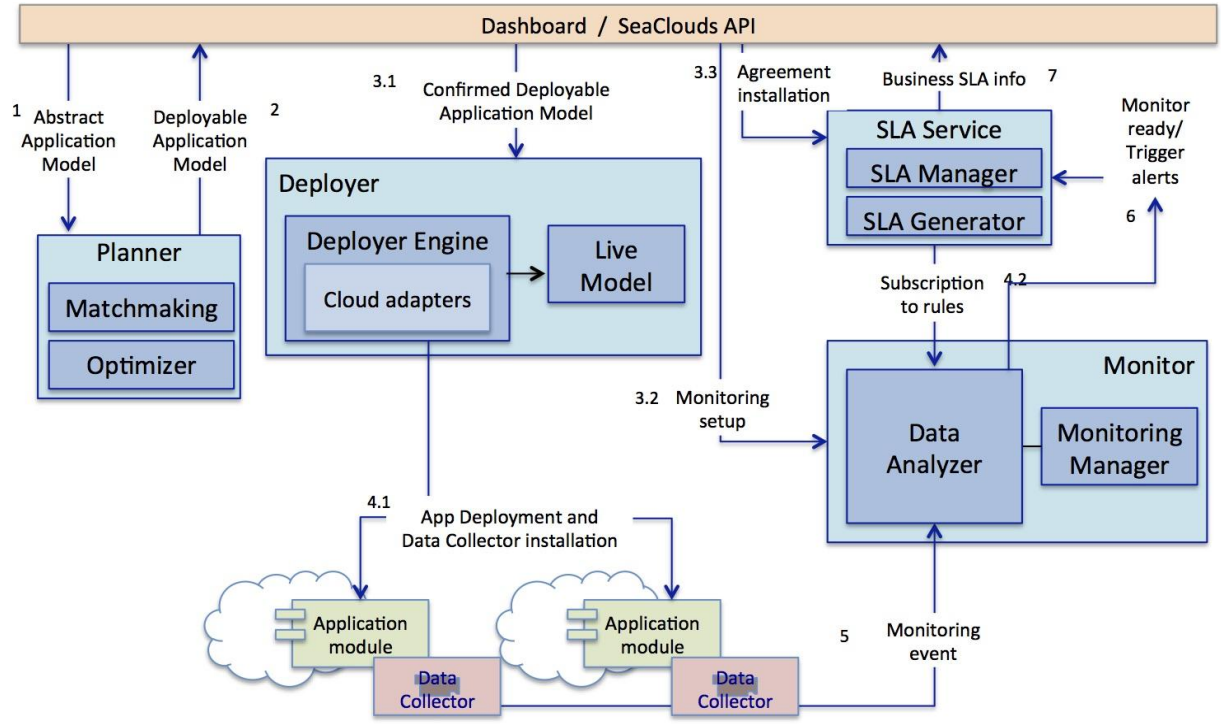

Figure 2: Current SeaClouds integrated platform architecture

3.2 Main SeaClouds Components

All SeaClouds components have been released under the Apache 2.0 license and are available at the following URL https://github.com/SeaCloudsEU/SeaCloudsPlatform. The table below offers information about the programming language adopted for developing the component, the external libraries that have been used (if any), the dependencies with other components (we assume that a component depends on another if it uses the interface of the second component), the software interface used by the component. A short description of each component is detailed in the subsections below.

Component name	Programming language	External libraries used	Dependencies with other components	Offered software interfaces
Dashboard	HTML5, JavaScript	Bootstrap library[1] Angular JS[2]	Planner, Deployer, Monitor, SLA Service	REST API
Planner	Java	Apache Tomcat, json-simple 1.1.1, slf4j-api (v 1.6.6) logger	Matchmaker, Optimizer, Dashboard	REST API
Deployer	Java	Apache Brooklyn[3]	Planner, Monitor	REST API
SLA Service	Java	jersey-1.18, spring-3.2.4, jpa-2.0-api, jackson-2.4.5, qos-models-2.4	Monitor, in particular, addObserver and sendMonitoringRuleI nstalled functions	REST API
Monitor	Java	monitoring-manager.jar[4] data-collector-1.3-snapshot.jar[5] fuseki-server.jar[6] rsp-services-csparql.jar[7]	Deployer, SLA Service, and Dashboard	REST API

[1] http://getbootstrap.com
[2] https://angularjs.org
[3] https://brooklyn.incubator.apache.org/
[4] https://github.com/deib-polimi/modaclouds-monitoring-manager
[5] https://github.com/imperial-modaclouds/imperial-modaclouds-mvn-repo/tree/master/releases/imperial/modaclouds/monitoring
[6] http://archive.apache.org/dist/jena/binaries
[7] https://github.com/deib-polimi/rsp-services-csparql

Dashboard

The main goal of the SeaClouds Dashboard is to provide a simple interface to the application administrator. The Dashboard is, in fact, focused on supporting the deployment and execution of multi-cloud applications. It is going to be integrated with the design-time GUI described in Deliverable D5.2.2 [8], more focused on the front-end. Thanks to this integration, it will support the whole lifecycle of an application, starting from the definition of its structured to the discovery and allocation of cloud resources and then to deployment and execution.

The Dashboard Component is a Web application that runs on any browser. It relies on the REST API to interact with the other components.

Planner

The planner is in charge of generating deployment plans from the Abstract Application Model. To do so, the planner invokes first the matchmaker, which finds a list of suitable cloud offerings for each module of the Abstract Application Model. And then, to the optimizer, which generates a list of abstract deployment plans using optimization problem techniques

The planner is also in charge of generating replans if a violation of the QoS/SLA occurs, using the same components. Its stable version is available under the sub-repository https://github.com/SeaCloudsEU/SeaCloudsPlatform/tree/master/planner.

The planner is organized in the two main sub-components that are described below.

Matchmaker: The Matchmaker is in charge of matching the different modules of the Abstract Application Model (AAM) with cloud offerings for both IaaS and PaaS.

Optimizer: The Optimizer creates an optimization problem whose solution is a set of ADP, that is, the combination of cloud resources and number of instances that satisfy the best the application requirements. To create the optimization problem *constraints,* it uses the modules in the AMM, their relationships and their QoS properties, the set of suitable cloud offers for each module provided by matchmaker. To create the *objective* function it uses the application QoS requirements. To find (sub-)optimal solutions of the optimization problem, it implements search-based algorithms. Besides being part of the Planner, the Optimizer can also be used in isolation through its REST API.

Deployer

The Deployer is composed of several elements (more details in Deliverable D4.1 [3]). The main element is the Deployer Engine, which receives a Deployable Application Model (DAM) through its Deployer API and executes it. As the Deployer Engine is cloud-agnostic, it is able to deploy applications on different cloud providers using multiple Cloud Adapters (PaaS and IaaS levels), which are part of the Deployer component itself. The DAM contains the necessary information to deploy an application over a set of cloud providers (locations). More specifically, the DAM describes the application topology detailing the application

components, the relationships, the dependencies, features, constraints, the target providers, etc. Therefore, the DAM includes an embedded deployment plan for carrying out the deployment based on the modules dependencies and relations.

The Deployer processes the DAM and uses the necessary operations that allow to manage the target locations and the cloud resources in an homogeneous way. Then, the application modules are distributed and the relationships are established, archiving the desired application behaviour.

The Deployer provides the necessary operations or methods, through the REST API, allowing the post-deployment management like changing the application status (e.g., stop, pause, restart) or performing entity-specific actions (e.g., scale up/down). Moreover, the Deployer maintains the model of the current application deployment status called Live Application Model (LAM).

SLA Service

The SLA Service represents the component responsible for generating and storing the formal documents describing electronic agreements between the parties involved in SeaClouds: customers, application providers and cloud providers. At runtime, the component is in charge of supervising that all the agreements are respected.

The SLA Service exposes the basic functionalities of handling providers, templates and agreements, and searching for violations and penalties, through a REST interface. It also offers the possibility to push events, such as violations and penalties, to subscribed components. The code of this component is available here https://github.com/SeaCloudsEU/sla-core.

Monitoring Platform

SeaClouds monitoring platform encapsulates and extends the functionality offered by MODAClouds monitoring platform [9]. The latter platform uses four core components, the Monitoring Manager, the Knowledge Base, the Data Analyser, and one or more Data Collectors. Further details about these components are provided in [10].

To exploit the functionality offered by SeaClouds monitoring platform, the SeaClouds components (e.g., Planner) should interact with the internal components of the monitoring platform, a.k.a., MODAClouds components. To reduce the coupling between the external and the internal components, we adopt the mediator design pattern. Based on this pattern, the interaction between components and the necessary logic is encapsulated by a mediator component, which we call Controller.

3.3 TOSCA YAML Object Model

Besides the main components described in the previous section, the SeaClouds Platform includes a library, the TOSCA YAML Object Model, that encapsulates all various SeaClouds models (i.e. cloud offerings, AAM, ADP, DAM and Live Model) and is going to be used by almost

all SeaClouds components to interact with these models and transform them into a TOSCA YAML specification (and vice versa).

More specifically, the library is written in Java and is available in the following sub-repository https://github.com/SeaCloudsEU/tosca-parser. It is released, as all other parts of the SeaClouds platform under the Apache 2.0 license and uses the following pre-existing libraries Guava 18.0and Snake-YAML 1.15.

4 How to get and install the SeaClouds Integrated Platform

SeaClouds project has a Continuous Integration chain in place (Section 6 details it). This allows to have all the binaries produced by each software component of SeaClouds to be always available from https://oss.sonatype.org/content/groups/public/eu/seaclouds-project/.

The consortium has identified Apache Brooklyn as the tool to easily deploy SeaClouds. We currently support deployments against [Bring Your Own Nodes (BYON)] and to all the IaaS provider supported by Apache jclouds[8].

In the following subsections we show how it is possible to deploy the SeaClouds platform both on a local computer and on the cloud.

4.1 Local Deployment

The deployment of SeaClouds on a local computer is supported to allow users experimenting with the platform.

To simplify the creation of the nodes needed to deploy SeaClouds, a convenient Vagrantfile has been created for the end-users. Make sure you have Vagrant[9] and Apache Brooklyn[10] installed, then:

```
cd $HOME
git clone git@github.com:SeaCloudsEU/seaclouds-distribution.git
cd seaclouds-distribution
./setup
```

Please make sure you have configured BROKLYN_HOME at least in the current terminal.

```
vagrant up
```

This spins up a virtual environment, made up of 2 VMs, which are accessible at

[8] http://jclouds.org
[9] https://www.vagrantup.com/
[10] https://brooklyn.incubator.apache.org/

`192.168.100.10` and `192.168.100.11`.

Start Apache Brooklyn
nohup $BROOKLYN_HOME/bin/brooklyn launch &

This starts up your instance of Apache Brooklyn on your workstation, accesible at http://localhost:8081.
Please double-check in nohup.out the correct url.

Finally, copy and paste SeaClouds blueprint[11] to deploy the SeaClouds platform on the 2 VMs created by Vagrant previously.

4.2 Launching in the clouds

The previous deployment option has to be considered non-production ready: it is a great way to start with SeaClouds with no effort and get familiar with the main concepts. Of course, deploy SeaClouds on the cloud is more interesting if an organization wants to support it in production. By simply editing the location pre-specified on the seaclouds blueprint, it'd be possible to deploy SeaClouds against any IaaS provider supported by Apache Jclouds

For example, instead of:

```
location:
  byon:
    user: vagrant
    privateKeyFile: ~/git/seaclouds/seaclouds-distribution/seaclouds_id_rsa
    hosts:
    - 192.168.100.10
    - 192.168.100.11
```

one could instead use:

```
location: jclouds:softlayer:ams01
```

To provision the 2 hosts on demand on the IBM SoftLayer cloud provider in the datacenter in Amsterdam.

[11] https://github.com/SeaCloudsEU/seaclouds-distribution/blob/master/seaclouds.yaml

5 How to use the SeaClouds Integrated Platform: an example

The software application here considered for illustrating SeaClouds behavior is brought from Apache Brooklyn deploying tutorial [11]. This application is part of *Brooklyn blueprint* and provides enough complexity to exemplify the current Seaclouds capabilities. In the following subsections we describe the functionality and structure of the application (Section 5.1), then we define the corresponding AAM (Section 5.2) and show how, starting from this AAM, the matchmaking and optimization process works (Section 5.3), then we describe the monitoring rules and the SLA associated to the application (Section 5.4), the DAM associated to the application (Section 5.5) and, finally, the way the application is executed and monitored (Section 5.6).

5.1 Description of the Application Example

The application example implements a simple web chat room. Concretely, users can send messages providing their name and the message text. These messages are stored in a database and they are shown to all the chat users. A user can leave the room and, when she/he eventually returns, can still see the previously sent messages. The software architecture of the application consists of the following types of modules.

- A web interface consisting of three different web pages: a *welcome* page, a page that lists links to the provided application functionality, and a *chat* page to interact with the business logic.
- An external message database.

Web interface is packaged in a *.war* (**W**eb application **AR**chive) file `chat-webApplication.war`[12] which uses an external database to store and retrieve students' information.

The application deployment executes on top of application server (e.g., Tomcat 7) to deploy `chat-webApplication.war` and requires the Message Database module for its data persistence. In turn, Message Database module requires a MySQL database management system.

Figure 3 shows the architectural topology of the application example.

12 http://search.maven.org/remotecontent?filepath=io/brooklyn/example/brooklyn-example-hello-world-sql-webapp/0.6.0/brooklyn-example-hello-world-sql-webapp-0.6.0.war

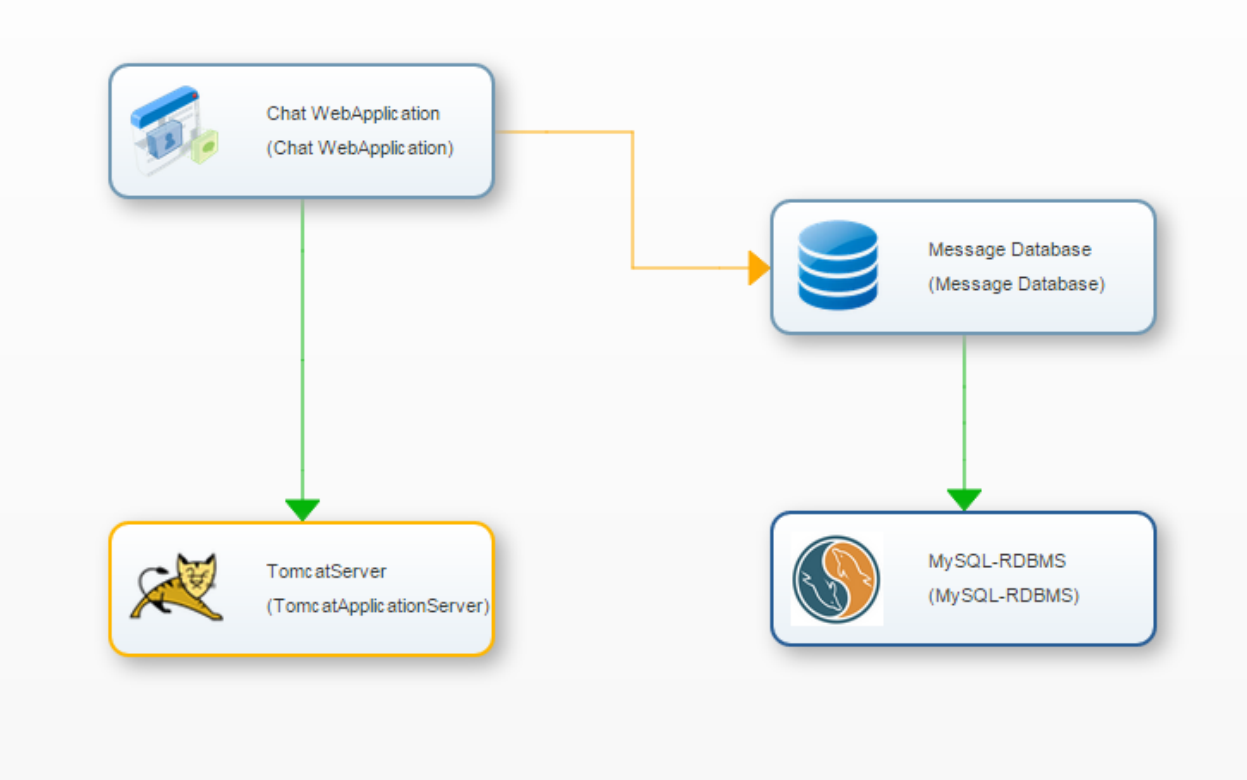

Figure 3: Application Chat topology

The application example is a preexisting one. Without using SeaClouds, someone willing to deploy it on some cloud would need to go through the following steps:

- Select the cloud services to be used, assess that they fit the needs of the application, and acquire these services from some cloud provider.
- Assuming that we have selected a IaaS cloud, start the virtual machines.
- Configure and start an application server.
- Configure and start a database management system.
- Set up the Message database.
- Configure the application to use the remote database.
- Deploy `chat-webApplication.war` on the application server.

The goal of SeaClouds with respect to this application example is to simplify all above work by automatizing most of the steps or guide non-expert users where needed. Moreover, the goals are also to i) ensure that the selected cloud resources are the optimal ones, given the characteristics of the application and the trade-off between service characteristics and cost; ii) monitor the performance of the application and make sure that, given the SLA offered by the selected cloud provider, any violation is communicated to the operator.

In the next release of the SeaClouds platform, we will also tackle selection and usage of PaaS services as well as the possibility to change the application deployment to deal with violations of the SLA or of generic QoS parameters.

Application requirements

In order to demonstrate the ability of SeaClouds to manage application with technical and quality requirements, we assume that the example application has the following requirements:

- The database is MySQL 5.0 and needs 50GB of size.
- The application server has to be able to execute Java.
- The application availablity should be higher than 99.8%.
- The application expected response time is lower than 2 seconds for an arrival rate of 50 messages per minute.
- The chat owner organisation expects to spend less than 200 Euros per month for executing the application on a cloud.

For reasoning over response time requirements, we also provide the following information that are assumed to be acquired by sturying the behavior of the application: each message sent through the application GUI produces, on average, two queries to the database; in the testing environment a request took in average 50ms to execute the code in the web interface and 30ms to execute a query to the database; the testing environment was composed of virtual machines of type `hp_cloud_services.2xl`.

5.2 Definition of the Abstract Application Model

Based on the Application Example described previously, we define the Abstract Application Model in TOSCA YAML. While in the next release of the SeaClouds platform the user will be guided through the definition of the AAM, at the moment, he/she has to define it manually. The syntax used for the AAM TOSCA YAML is defined in D3.2 [12]. Below we provide a short overview of such syntax and the present the AAM of the application example.

AAM TOSCA YAML syntax

The Abstract Application Model (AAM) is structured on two layers: The *Deployment layer* and the *Logic Layer*.

The Deployment layer defines the modules of the application topology, which are represented as `node_template`. A `node_template` is of a specific `node_type`. If the module is a component to deploy, the `node_type` identifies the type of the component (e.g. `seaclouds.nodes.Deploy.MySQL`). If the module is the abstract service where these

components are to be deployed, the `node_type` identifies the type of the service (in case of a generic IaaS, `seaclouds.nodes.Compute`).

For `node_templates` of type `seaclouds.nodes.Deploy.*`, textual information (e.g. credentials) are represented as `attributes`, whereas resources (e.g. configuration files) are represented as `artifacts`. These components are required to be deployed in a service host, which is represented as a `host` in the `requirements`.

The logic layer is composed of the functionalities of the modules and the dependencies between them. The functionalities are expressed also as `node_template` using the `seaclouds.nodes.Logic` type. The Logic nodes must have a host named requirement which refers to the module implementing the functionality being described. The dependency from functionalities provided by other modules in the application can be expressed using requirement linking to other logic nodes with a relationship of the type `seaclouds.relationships.Uses`. The Uses relationship has a property `average_usage_count` which defines how many times the target functionality needs to be used in average to provide the functionality being described.

In `Logic` nodes the user can also define two properties: `qos_requirements` and `qos_info`, that respectively specify the QoS requirements for the given functionality and the benchmark information needed to compute them.

In Table 4 we depict the grammar of a node_template as defined in TOSCA YAML Simple Profile specification [13]. Some of the elements are not yet exploited in the current version of the platform, but will be considered in the future.

```
<node_template_name>:
  type: <node_type_name>
  description: <node_template_description>
  properties:
    <property_definitions>
  attributes:
    <attribute_definitions>
  requirements:
    <requirement_definitions>
  capabilities:
    <capability_definitions>
  interfaces:
    <interface_definitions>
  artifacts:
    <artifact_definitions>
```

Table 2: node_template grammar

The AAM for the application example

Below we depict the AAM of the example. The `node_templates` `java_ee_server` and `db` define the computational services that the user writing the AAM is willing to acquire for the application. MySQL server is being hosted on the `db` computational server as reported by the specification of the third `node_template` (`mysql_server`) under the requirements label, while the `java_ee_server` is hosting a Tomcat installation as indicated by the fourth note_template (`tomcat_server`). `mysql_server` and `tomcat_server` also include the configuration information and the artifacts that are needed for their installation. `db.query` and `java_ee_server.operation` define the logic layer of the application. They include the QoS information that we have defined in Section 5.1 as well as the QoS requirements that the user defines for the application. Finally, the `relationship_template` defines the connection between `java_ee_server.operation` and `db.query` and highlights the fact that every time the application (`java_ee_server.operation`) uses the database db.query, it performs two queries at a time (see the value for `average_usage_count`).

```
tosca_definitions_version: tosca_simple_yaml_1_0_0

topology_template:
  node_templates:

    java_ee_server:
      type: seaclouds.nodes.Compute

    db:
      type: seaclouds.nodes.Compute
      properties:
        disk_size: 50 GB

    mysql_server:
      type: seaclouds.nodes.deploy.MySQL
      properties:
        name: some_name
        user: some_user
        password: some_password
        version: 5.5.37
      artifacts:
        - db_create: files/db_create.sql
          type: tosca.artifacts.File
      requirements:
        - host: db

    tomcat_server:
      type: seaclouds.nodes.deploy.tomcatServer
      properties:
```

```
        version: 7.0.53
      artifacts:
        - war: chat-webApplication.war
      requirements:
        - host: java_ee_server

    db.query:
      type: seaclouds.nodes.Logic
      properties:
        qos_info:
          execution_time: 30 ms
          benchmark_platform: hp_cloud_services.2xl
      requirements:
        - host: db

    java_ee_server.operation:
      type: seaclouds.nodes.Logic
      properties:
        qos_info:
          execution_time: 50 ms
          benchmark_platform: hp_cloud_services.2xl
        qos_requirements:
          response_time: 2 sec
          availability: 0.998
          cost: 200 euros_per_month
          workload: 50 req_per_mins
      requirements:
        - host: java_ee_server
        - dependence_to_query:
            node: db.query
            relationship: java_ee_server.query.db


relationship_templates:
  java_ee_server.query.db:
    type: seaclouds.relationships.Uses
      properties:
          average_usage_count: 2
```

5.3 Matchmaking and optimization

The objective of the matchmaking and optimization steps is to define a proper allocation of resources for the AAM. The following subsections describe the outcome of these two steps in the case of the application example.

5.3.1 Matchmaking process

The Matchmaker is in charge of identifying a list of candidate cloud services for each of the abstract services defined in the AAM. Particularly, it returns a map of <abstract service, list of cloud offerings> where each of the cloud offerings in the list satisfies the technical and quality requirements of the abstract service defined in the AAM. To do so, the matchmaker executes the rules that identify the semantic comparison between properties (e.g. if the AAM requires a IaaS with storage features of at least 50 GB, all IaaS that can provide 50 GB or more satisfy the requirement). These rules are applied when comparing the properties between the AAM and the cloud offerings. While at the moment the possible cloud offerings are predefined in a configuration file, in the next version of the integrated platform, they will be returned by the Discovery component that, in turn, will interact with initiatives such as Cloud Harmony to acquire them.

The following set of cloud offerings fulfills the requirements of one of the abstract services of our application example (particularly, we show the case of java_ee_server). The actual data in the offering description is intended to be just an example and they are currently not real information from the cloud providers as at the moment we lack the integration with real information sources like Cloud Harmony. The cloud offerings specification is given in TOSCA YAML. Each offer is expressed in terms of a node_template with a type that corresponds to the specific cloud service and a set of properties that describe the offering.

```
tosca_definitions_version: tosca_simple_yaml_1_0_0

topology_template:

    node_templates:

      aws-ec2:us-west-2:
        type: seaclouds.nodes.Compute.Amazon
        properties:
          num_cpus: 4
          availability: 0.98
          cost: 0.928 usd_per_hour
          performance: 62 ecb


      seaclouds-hpcloud-region-b:
        type: seaclouds.nodes.Compute.HP
        properties:
          num_cpus: 6
          disk_size: 1 TB
          scaling_vertical: auto
          availability: 0.998
          cost: 0.07 usd_per_hour
          performance: 3 ecb
```

```
hp_cloud_services.2xl:
  type: seaclouds.nodes.Compute.HP
  properties:
      region: 'seaclouds.types.Locations.NV.US.LasVegas'
      load_balancing: false
      scaling_horizontal: 'no'
      storage_file_system: 'ext4'
      disk_type: 'sata'
      local_storage: 470 GB
      mem_size: 30 GB
      num_cpus: 8
      availability: 0.995
      cost: 3.41 usd_per_hour
      performance: 53 ecb


c1.xlarge:
  type: seaclouds.nodes.Compute.Amazon
  properties:
      region: 'seaclouds.types.Locations.AM.US.OR.Portland'
      operating_system: 'seaclouds.types.os.linux.ubuntu'
      num_cpus: 8
      mem_size: 7 GB
      disk_type: 'sata'
      local_storage: 2 TB
      cost: 5.52 usd_per_hour
      performance: 116 ecb
      availability: 0.9995
```

5.3.2 The optimization process

The cloud offerings proposed by the Matchmaker is given to the Optimizer together with the AAM. The Optimizer performs the optimization process implementing search-based algorithms guided by meta-heuristics and provides a list of Abstract Deployment Plans (ADP).

The Optimizer associates each application module in the AAM of type "`seaclouds.nodes.Compute`" with an actual IaaS cloud offer. Additionally, it includes the appropriate initial number of replicas to deploy for each module in order to satisfy both availability and response time requirements. This is done for each of the ADP generated.

The following YAML code provides an instance of ADP of our application example assuming that the suitable options provided by the Matchmaker for `db` module are the same as the illustrated in Section 5.3.1 for `java_ee_server` module. This case illustrates that the search-based algorithm within the Optimizer has found `aws-ec2:us-west-2` as the most suitable IaaS option to use as host for `java_ee_server` module, and `seaclouds-hpcloud-region-b` as the most suitable option for `db` module. In both cases the ADP sets the number of instances needed to fulfill the both availability and response time requirements under the expected workload to one.

```
tosca_definitions_version: tosca_simple_yaml_1_0_0

topology_template:
  node_templates:

    java_ee_server:
      type: seaclouds.nodes.Compute.Amazon
        properties:
          location: aws-ec2:us-west-2
          num_instances: 1

    db:
      type: seaclouds.nodes.Compute.HP
        properties:
          num_instances: 1
          location: seaclouds-hpcloud-region-b
          disk_size: 50 GB



  mysql_server:
      type: seaclouds.nodes.deploy.MySQL
      properties:
        name: some_name
        user: some_user
        password: some_password
        version: 5.5.37
      artifacts:
        - db_create: files/db_create.sql
          type: tosca.artifacts.File
      requirements:
        - host: db

    tomcat_server:
      type: seaclouds.nodes.deploy.tomcatServer
      properties:
        version: 7.0.53
      artifacts:
        - war: chat-webApplication.war
      requirements:
        - host: java_ee_server

    db.query:
      type: seaclouds.nodes.Logic
      requirements:
        - host: db

    java_ee_server.operation:
      type: seaclouds.nodes.Logic
      properties:
          qos_requirements:
          response_time: 2 sec
          availability: 0.998
```

```
          cost: 200 euros_per_month
          workload: 50 req_per_mins
      requirements:
        - host: java_ee_server
        - dependence_to_query:
            node: db.query
            relationship: java_ee_server.query.db


relationship_templates:
  java_ee_server.query.db:
    type: seaclouds.relationships.Uses
      properties:
          average_usage_count: 2
```

These ADP work as basis for the generation of a DAM, since the final DAM includes additional concepts as the account information to access the selected cloud as well as some configuration information.

Implementation note

At the time in which this deliverable is being written, there is a syntactical mismatch between the language understood by the Matchmaker and the one supported by the Optimizer. These results in the fact that the AAM and the ADP manipulated by the Optimizer for the reference application example are the ones listed below. Of course, the mismatch is purely syntactical and is being fixed by the SeaClouds team.

AAM understood by the Optimizer

```
tosca_definitions_version: tosca_simple_yaml_1_0
node_templates:

  Chat_WebApplication:
    type: seaClouds.nodes.WebApplication.Java
    properties: {version: 7}
    requirements: {host: tomcat_server, database_endpoint: mysql_server}
    QoSpropertiesPOC:
      executionTimeMeasuredInPOC: hp_cloud_services.2xl
      executionTimePOC: 50.0
      OpProfilePOC: {mysql_server: 2.0}


  mysql_server:
    type: seaclouds.nodes.Deploy.MySQL
    properties: {version: 5.5.37}
    requirements:
      host: seaclouds.nodes.Compute
      constraints:
```

```
        localStorage: {greater_or_equal: 50}
        suitableServices:
          - aws-ec2:us-west-2
          - seaclouds-hpcloud-region-b
          - hp_cloud_services.2xl
          - c1.xlarge
    QoSpropertiesPOC:
      executionTimeMeasuredInPOC: hp_cloud_services.2xl
      executionTimePOC: 30.0

  tomcat_server:
    type: seaclouds.nodes.deploy.tomcatServer
    capabilities:
      host: seaclouds.nodes.WebApplication.Java
      version: 7.0.53
    requirements:
      host: seaclouds.nodes.Compute
      constraints:
        suitableServices:
          - aws-ec2:us-west-2
          - seaclouds-hpcloud-region-b
          - hp_cloud_services.2xl
          - c1.xlarge
QoSrequirementsPOC:
  availabilityPOC: 0.998
  responseTimePOC: 2000.0
  workloadPOC: 50.0
  costPOC: 200.0
```

Cloud Offer description format understood by the Optimizer

```
---
tosca_definitions_version: tosca_simple_yaml_1_0_0
node_templates:

  aws-ec2:us-west-2:
    type: seaclouds.nodes.Compute.Amazon
    properties:
      cpuCores: 4
      availabilityPOC: 0.98
      costPOC: 0.928
      performancePOC: 62

  seaclouds-hpcloud-region-b:
    type: seaclouds.nodes.Compute.HP
    properties:
      cpuCores: 6
      disk_size: 1 TB
      scaling_vertical: auto
      availabilityPOC: 0.998
      costPOC: 0.07
      performancePOC: 3

  hp_cloud_services.2xl:
    type: seaclouds.nodes.Compute.HP
    properties:
      region: 'seaclouds.types.Locations.NV.US.LasVegas'
```

```
      load_balancing: false
      scaling_horizontal: 'no'
      storage_file_system: 'ext4'
      disk_type: 'sata'
      local_storage: 470 GB
      mem_size: 30 GB
      cpuCores: 8
      availabilityPOC: 0.995
      costPOC: 3.41
      performancePOC: 53

  c1.xlarge:
    type: seaclouds.nodes.Compute.Amazon
    properties:
      region: 'seaclouds.types.Locations.AM.US.OR.Portland'
      operating_system: 'seaclouds.types.os.linux.ubuntu'
      cpuCores: 8
      mem_size: 7 GB
      disk_type: 'sata'
      local_storage: 2 TB
      costPOC: 5.52
      performancePOC: 116
      availabilityPOC: 0.9995

  latencyExternalPOC: 200.0
  latencyInternalPOC: 2.0
```

ADP produced by the Optimizer

```
tosca_definitions_version: tosca_simple_yaml_1_0
node_templates:

  Chat_WebApplication:
    type: seaClouds.nodes.WebApplication.Java
    properties: {version: 7}
    requirements: {host: tomcat_server, database_endpoint: mysql_server}
    QoSpropertiesPOC:
      executionTimeMeasuredInPOC: hp_cloud_services.2xl
      executionTimePOC: 50.0
      OpProfilePOC: {mysql_server: 2.0}

  mysql_server:
    type: seaclouds.nodes.Deploy.MySQL
    properties: {version: 5.5.37}
    requirements:
      host: aws-ec2:us-west-2
      instancesPOC: 1
    QoSpropertiesPOC: {
       executionTimeMeasuredInPOC: HP.compute.standard.medium, executionTimePOC: 30.0}

  tomcat_server:
    type: seaclouds.nodes.deploy.tomcatServer
```

```
capabilities: {host: seaclouds.nodes.WebApplication.Java, version: 7.0.53}
requirements:
  host: seaclouds-hpcloud-region-b
  instancesPOC: 1
QoSrequirementsPOC: {
   availabilityPOC: 0.998, responseTimePOC: 2000.0, workloadPOC: 50.0,
   costPOC: 200.0}
```

5.4 Definition of Monitoring Rules and SLA

Another step in the interaction of SeaClouds components is related to the definition of the required monitoring rules and SLA. In particular, the Dashboard interacts with the Monitor and the SLA Service for defining the monitoring rules and the SLA, respectively.

Concerning the definition of monitoring rules, they generally specify the monitored entities (e.g., virtual machines), the kind of collected data (e.g., the application response time), the aggregation way of collected data, monitoring actions, and under which conditions they will be performed (logical expressions applied on monitoring data). The information included in a monitoring rule is modeled by the XML schema, presented in [7]. Based on this model, the monitoring rules defined for checking the delivered quality of the application example, are presented in the following of the section.

SLA agreements are formal documents describing electronic agreements between the parties involved in SeaClouds: customers, application providers and cloud providers. In the scope of the project, they do not aim at representing a contractual relationship.

SLA agreements describe the service that is delivered, its functional and non-functional properties, and the duties of each party involved. The SLA agreements in SeaClouds follow the schema defined in the WS-Agreement specification [14], an open standard specifying a language and a protocol for creating SLAs.

An SLA agreement, based on this WS-Agreement schema, is presented in the following section.

Monitoring Rules for the Application Example

Based on the previously specified requirements of the application, the metrics, required for checking the delivered quality by the application, are the response time and the availability of the application.

To define the target monitoring entities of these rules, we adopt the ones included in the MODAClouds monitoring ontology[13]. In particular, the main entities of the MODAClouds monitoring ontology are the *CloudProvider*, the *VM*, the *Internal Component*, and the *Method* entities. The first entity represents a cloud provider that hosts the whole or a part of an application. The second entity represents a virtual machine and is further characterized by the number of CPUs, reserved by this virtual machine. An *Internal Component* entity represents the whole or a part of an application. Finally, a *Method* entity is used for capturing the notion of a method of the source code of an application.

The monitoring rules for the application example are provided in Table 3. In detail, the first rule calculates the average response time of the application methods. In this rule, it is used the entity *Method* of MODAClouds monitoring ontology (*monitoredTarget class="Method"*), the *Average* as an aggregation function (*aggregationFunction name="Average"*), and the *Method* as a grouping class, based on which the aggregation will be performed (*groupingClass= "Method"*). It is also specified a proper condition over the value of the average response time, which checks whether this value is lower than two seconds.

In a similar vein, the second rule calculates the availability of the application by using the *Internal Component* entity of the MODAClouds monitoring ontology (*monitoredTarget class="InternalComponent"*). Recall that this kind of entity can represent the whole application. The specified condition checks whether the application availability is greater than 0.998. Also, observe that the availability metric needs some further parameter values (e.g., samplingTime, retryPeriod, retryTime, etc.) for collecting data. In the current version of SeaClouds platform, default values are used, but in the next version of the platform, these parameters will be specified by the end-user.

```
<monitoringRule id="avgRespTime" label="avgRespTime" timeWindow="30" timeStep="30">
 <collectedMetric metricName="ResponseTime">
   <parameter name="samplingProbability">1</parameter>
 </collectedMetric>
 <monitoredTargets>
   <monitoredTarget class="Method"/>
 </monitoredTargets>
 <metricAggregation aggregateFunction="Average" groupingClass="Method"/>
 <condition> METRIC &lt; 2 </condition>
 <actions>
   <action name="OutputMetric">
      <parameter name="metric">ResponseTimeViolation</parameter>
      <parameter name="value">METRIC</parameter>
      <parameter name="resourceId">ID</parameter>
   </action>
 </actions>
</monitoringRule>
```

[13] https://github.com/deib-polimi/modaclouds-qos-models/blob/master/doc/user-manual.md#actions

```
<monitoringRule id="availabilityRule" label="availabilityRule" timeWindow="30" timeStep="30">
  <collectedMetric metricName="AppAvailable">
   <parameter name="samplingTime">60</parameter>
   <parameter name="retryPeriod">5</parameter>
   <parameter name="retryTimes">0</parameter>
   <parameter name="port">8080</parameter>
   <parameter name="path">/index.html</parameter>
 </collectedMetric>
 <monitoredTargets>
   <monitoredTarget class="InternalComponent" type="chat-webApplication"/>
 </monitoredTargets>
 <condition> METRIC &gt; 0.998 </condition>
 <actions>
   <action name="OutputMetric">
     <parameter name="metric">AppAvailabilityViolation</parameter>
     <parameter name="value">METRIC</parameter>
     <parameter name="resourceId">ID</parameter>
   </action>
 </actions>
</monitoringRule>
```

Table 3: Monitoring rules for the application example

SLA for Web Chat Case Study

Given the specified requirements of the application and the monitoring rules obtained from them, an SLA Agreement between the application provider (acting as a provider) and a generic user of the application (acting as a customer) is supplied in Table 4.

An agreement may have several elements and attributes. In this case, the most important elements are:

- *context*: contains the interested parties and the role of each one. The Context/ServiceProvider element contains "AgreementResponder" or "AgreementInitiator" depending on who initiates the agreement. Usually, the service customer is the agreement initiator, so the service provider is the agreement responder.
- *service properties*: these elements contains the variables used in the service level objectives.
- *guarantee terms*: these are the terms that express the service level objectives of the application. In this case, the two service level objectives are a response time less than 2 and an availability of a 99% at least. The SLA Service leverages on the monitoring rules, so the output metrics of the monitoring rules are specified here.

According to WS-Agreement, the custom service level is a domain-specific expression. A simple expression syntax has been developed in SeaClouds to express the needed non functional requirements.

```
<wsag:Agreement                    xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
xmlns:sla="http://sla.atos.eu">
  <wsag:Name>ChatAppRoom</wsag:Name>
  <wsag:Context>
    <wsag:AgreementInitiator>client</wsag:AgreementInitiator>
    <wsag:AgreementResponder>seaclouds</wsag:AgreementResponder>
    <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
    <sla:Service xmlns:sla="http://sla.atos.eu">chatroom</sla:Service>
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceProperties wsag:Name="NonFunctional" wsag:ServiceName="default">
        <wsag:VariableSet>
          <wsag:Variable wsag:Name="ResponseTime" wsag:Metric="xs:double">
            <wsag:Location></wsag:Location>
          </wsag:Variable>
          <wsag:Variable wsag:Name="AppAvailable" wsag:Metric="xs:double">
            <wsag:Location></wsag:Location>
          </wsag:Variable>
        </wsag:VariableSet>
      </wsag:ServiceProperties>
      <wsag:GuaranteeTerm wsag:Name="ResponseTimeGT">
        <wsag:ServiceLevelObjective>
          <wsag:KPITarget>
            <wsag:KPIName>ResponseTime</wsag:KPIName>
            <wsag:CustomServiceLevel>{"constraint":    "ResponseTimeViolated",    "qos"    :
"ResponseTime LT 2"}</wsag:CustomServiceLevel>
          </wsag:KPITarget>
        </wsag:ServiceLevelObjective>
      </wsag:GuaranteeTerm>
      <wsag:GuaranteeTerm wsag:Name="AppAvailableGT">
        <wsag:ServiceLevelObjective>
          <wsag:KPITarget>
            <wsag:KPIName>AppAvailable</wsag:KPIName>
            <wsag:CustomServiceLevel>{"constraint":    "AppAvailableViolated",    "qos"    :
"AppAvailable GT 0.998"}</wsag:CustomServiceLevel>
          </wsag:KPITarget>
        </wsag:ServiceLevelObjective>
      </wsag:GuaranteeTerm>
    </wsag:All>
  </wsag:Terms>
</wsag:Agreement>
```

Table 4: SLA for Web Chat

5.5 Definition of the Deployable Application Model

TOSCA YAML DAM

The ADP, the monitoring rules, and the SLA are reviewed by the user through the Dashboard. The user can then generate from these a DAM that assembles together all data produced by the Matchmaker and the Optimizer, points to the monitoring rules and SLA URI and, finally, includes in the specification information concerning the user credentials to be used to access the selected cloud services if they are provided by the user through the Dashboard. The DAM resulting from our application example is the one reported below:

```
tosca_definitions_version: tosca_simple_yaml_1_0_0

topology_template:
  node_templates:

    java_ee_server:
      type: seaclouds.nodes.Compute.Amazon
        properties:
          location: aws-ec2:us-west-2
          num_instances: 1

    db:
      type: seaclouds.nodes.Compute.HP
        properties:
          num_instances: 1
          location: seaclouds-hpcloud-region-b
          disk_size: 50 GB

    mysql_server:
      type: seaclouds.nodes.Deploy.MySQL
      properties:
        name: some_name
        user: some_user
        password: some_password
        version: 5.5.37
      artifacts:
        - db_create: files/db_create.sql
          type: tosca.artifacts.File
      requirements:
        - host: db

    tomcat_server:
      type: seaclouds.nodes.deploy.tomcatServer
      properties:
        version: 7.0.53
      artifacts:
        - war: chat-webApplication.war
      requirements:
        - host: java_ee_server

    cloud_credentials_java_ee_server:
      type: seaclouds.nodes.credentials
      properties:
        identity: ABCDEFGHIJKLMNOPQRST
        credential: s3cr3tsq1rr3ls3cr3tsq1rr3ls3cr3tsq1rr3l
      requirements:
        - host: java_ee_server

    cloud_credentials_db:
      type: seaclouds.nodes.credentials
      properties:
        identity: ABCDEFGHIJKLMNOPQRST
        credential: s3cr3tsq1rr3ls3cr3tsq1rr3ls3cr3tsq1rr3l
      requirements:
        - host: db

    db.query:
```

```
      type: seaclouds.nodes.Logic
      requirements:
        - host: db

    java_ee_server.operation:
      type: seaclouds.nodes.Logic
      properties:
          qos_requirements:
          response_time: 2 sec
          availability: 0.998
          cost: 200 euros_per_month
          workload: 50 req_per_mins
      requirements:
        - host: java_ee_server
        - dependence_to_query:
            node: db.query
            relationship: java_ee_server.query.db

relationship_templates:
  java_ee_server.query.db:
    type: seaclouds.relationships.Uses
      properties:
          average_usage_count: 2

    monitoring_rules:
      type: seaclouds.nodes.Monitoring
      artifacts:
        - monitoring_rules: <URI>
          type: tosca.artifacts.File

    slas:
      type: seaclouds.nodes.SLA
      artifacts:
        - sla_artifacts: <URI>
          type: tosca.artifacts.File
```

CAMP YAML DAM

The TOSCA YAML DAM defined in the previous section contains the information needed for deploying our application.

Since our first prototype of Deployer exploits the CAMP syntax for the DAM, the current version of the SeaClouds integrated platform requires a manual conversion from TOSCA to CAMP to be performed at this stage (such manual conversion is performed according to the rules defined in [15]). In the next release of the platform such gap will be covered and the Deployer will accept DAM in the TOSCA format.

The Deployer receives a DAM, which specifies the application to deploy, its distribution and orchestration, and follows the instruction to deploy the application using the services indicated.

Then, the Deployer initializes the monitoring components and maintains the management of the application.

As described in the SeaClouds Architecture [1], in the Deployer component, different engines could be used to deploy the application. In our first solution, Apache Brooklyn is used as Deployer engine, to accomplish the heterogeneous management of the cloud providers. Given this choice, the DAM definition is based on the YAML Blueprint specification of Brooklyn [16].

In the following subsections we provide a short overview of the CAMP-based syntax of the DAM, a definition of the mapping between CAMP and TOSCA constructs (this mapping constitutes the basis for the future development of the new version of the Deployer), and the CAMP YAML DAM of our reference application.

CAMP-based DAM schema

Below, we outline the CAMP-based DAM schema and provide an overview of the main components needed to describe the application and its management.

name: name of application
location: a location specification element as a string or map.

services: this block contains the entities which compose the application.
- serviceType/service: service reference
name: human readable name of entity.
id: id of entity.
location: location (provider service) where the entity will be deployed (**target cloud provider**).
config: (features and requirements of the entity).
children: a list of child specifications which will be configured as children of this entity.
policies: list of policy specifications which add behavior to the entity.
enrichers: enrichers of this entity.
initializers: values needed to configure this entity (key values).

The schema shows the elements needed by the Deployer in order to deploy and manage the application modules over the target providers. A detailed description of every element in the DAM is presented in deliverable D4.1 [3].

Mapping between TOSCA and CAMP DAM

The mapping between the TOSCA and CAMP DAMs is performed according to the rules defined in this section.
The `services` in CAMP YAML are represented by means of the `node_templates` in TOSCA YAML. In our example, the Tomcat service in CAMP YAML is represented by the node_templates `java_ee_server` (which defines the machine) and the `tomcat_server` (which defines the application). Analogously, MySQL in CAMP YAML is represented by the node_templates `db` (which defines the machine) and `mysql_server` (which defines the application).

The `type` of the `service` in CAMP YAML is mapped to the `type` of the `node_template` in TOSCA YAML that describes the application (i.e. `tomcat_server.type` and `mysql_server.type`).

The `location` of the `service` in CAMP YAML identifies the cloud service where the application should be deployed, which in the TOSCA YAML is in the `location` of `node_templates` defining the machine.
The credential information for the cloud service is represented by `identity` and `credential` inside the `services` node in CAMP YAML. In TOSCA YAML it is represented also as `identity` and `credential`, but in a new `node_template` of type `seaclouds.nodes.credentials`. In our example, these are `cloud_credentials_db` and `cloud_credentials_java_ee_server`.

The artifacts to deploy, are defined in `wars.root` (for the tomcat application) and `creationScriptUrl` (for MySQL) and are specified inside `brooklyn.config` in CAMP YAML. In TOSCA YAML, this is represented as `war` and `db_create` respectively, which are specified inside `artifacts`.

The `minDisk`, which represents the amount of disk required, is defined in `provisioning.properties` in CAMP YAML. This corresponds to the `disk_size` property in the `properties` of the `node_template` of the machine.

Finally, to perform the binding of the different components (in our example, the Tomcat server should connect to the dabatase), the CAMP YAML document defines the following instruction:

```
brooklyn.example.db.url: >
      $brooklyn:formatString(
            "jdbc:%s%s?user=%s&password=%s",
```

```
            component("mysql_server").attributeWhenReady("datastore.url"),
            "some_name",
            "some_user",
            "some_password"
      )
```

In TOSCA YAML, this binding is defined in properties of the component to be bound to. In our example, the properties of MySQL, include the `name`, the `user` and the `password`.
The rest of CAMP elements (e.g. enrichers) are not required in the current example, and their mapping with TOSCA is still to be defined.

Resulting CAMP YAML DAM

Given the mapping explained in the previous section, the DAM expressed in the CAMP YAML format is the one below:

```
name: Application
services:
- type: brooklyn.entity.webapp.tomcat.TomcatServer
  id: tomcat_server
  name: My TomcatServer
  location:
    jclouds:aws-ec2:
    identity: ABCDEFGHIJKLMNOPQRST
    credential: s3cr3tsq1rr3ls3cr3tsq1rr3ls3cr3tsq1rr3l
  brooklyn.config:
       wars.root: chat-webApplication.war
       java.sysprops:
       brooklyn.example.db.url: >
       $brooklyn:formatString(
              "jdbc:%s%s?user=%s&password=%s",
              component("mysql_server").attributeWhenReady("datastore.url"),
              "some_name",
              "some_user",
              "some_password"
       )

- type: brooklyn.entity.database.mysql.MySqlNode
  id: mysql_server
  name: My SQL Server
  location: seaclouds-hpcloud-region-b

  brooklyn.config:
       creationScriptUrl: files/db_create.sql
       provisioning.properties:
         minDisk: 50g
```

Let us discuss the main differences between CAMP and TOSCA models that could hamper the association of concepts between models. Regarding the connection of `tomcat_server` with `mysql_server`, in CAMP model the database connection for the web component is configured in the TomcatServer entity using a String "`jdbc:...`", which is generated at run-time. The reason is that, for establishing the connection, `tomcat_server` waits until `mysql_server` has been deployed and has an IP address. After that moment, it is able to request the database IP, which corresponds to the first parameter (i.e., `component("mysql_server").attributeWhenReady("datastore.url")`) of the connection String. Finally, `tomcat_server` completes the aforementioned connection String using the connection credentials provided by the `db` module (i.e., username and password).

We can also see that, while TOSCA model includes four nodes -named `java_ee_server`, `db` `tomcat_server` and `mysql_server`- CAMP model includes only two -`tomcat_server` and `mysql_server`. The reason is that CAMP aggregates the software and the infrastructure information in a single node (e.g., see that field `location` in `mysql_server` node includes also the information of the `db` infrastructure), while TOSCA model dedicates a node for each concept.

Besides the Deployer, also the Monitoring platform exploits the DAM to generate an application-specific instance of the MODAClouds monitoring ontology (described earlier in Section 5.4) in JSON format.

MODAClouds-Based Deployment Model of the Application Example

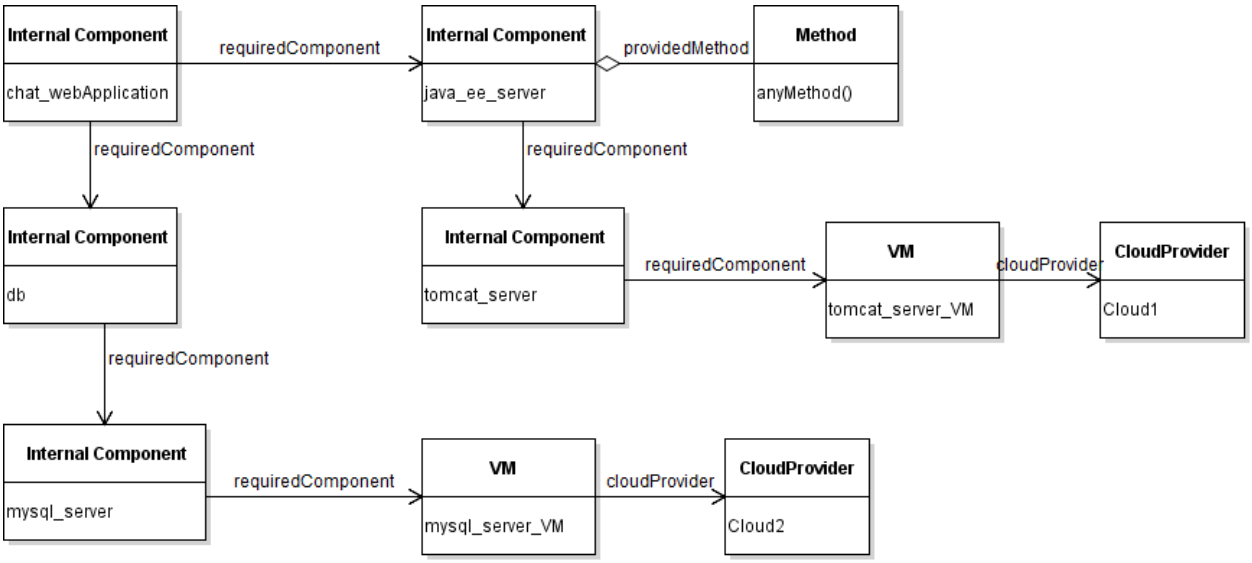

Figure 4: UML diagram of MODAClouds-based deployment model for the application example

The DAM is used at runtime also by the Monitoring platform. The monitoring internal representation of such DAM is obtained by instantiating the MODAClouds monitoring ontology. The resulting monitoring deployment model for the application example is provided in Figure 4

as a UML diagram. In this model, the business logic of the application is represented by the *Internal Component java_ee_server*, which provides multiple methods (in the figure, we do not provide specific method names). The business logic of the application is available through an application server, represented by the *Internal Component tomcat_server*. Also, the application requires the *Internal Component db*, which represents the database, used by the application for creating and storing the application data. The database is available through a database server, represented by the *Internal Component mysql_server*. Both application server and database are executed by using a VM of the machines of a *CloudProvider*.

The construction of the monitoring deployment model is obtained from the DAM according to the following rules: The type of a module can be used for identifying the kind of the corresponding MODAClouds entity. For instance, the *type seaclouds.nodes.deploy.MySQL* in TOSCA YAML DAM corresponds to an *Internal Component* MODAClouds entity. Additionally, the hosting requirements can be used for identifying the relationships between MODAClouds entities. For instance, the *requirements host: java_ee_server* inside the module *tomcat_server* in TOSCA YAML DAM corresponds to the relationship *requiredComponent* between the *Internal Components java_ee_server* and *tomcat_server* in the MODAClouds deployment model (see Figure 4).

The full specification of this mapping is going to be delivered in the next version of the integrated SeaClouds platform together with an automatic format translation. In the current version of the platform, the monitoring component has to be initialized with the deployment model presented above and defined in the following JSON format. The values of the fields (e.g., *mysql_server_VM_ID*), which are not present in TOSCA YAML DAM, are set as default values.


```
{"cloudProviders":[{"id":"Cloud2"},{"id":"Cloud1"}],

"vMs":[{"numberOfCPUs":6,"cloudProvider":"Cloud2","type":"mysql_server_VM","id":"mysql_server_VM_ID"},{"numberOfCPUs":0,"cloudProvider":"Cloud1","type":"tomcat_server_VM","id":"tomcat_server_VM_ID"}],

"internalComponents":[{"requiredComponents":["mysql_server_VM_ID"],"providedMethods":[],"type":"mysql_server","id":"mysql_server_ID"},{"requiredComponents":["tomcatServer_ID","java_ee_server_ID","DB1"],"providedMethods":["anyMethod_ID"],"type":"java_ee_server","id":"java_ee_server_ID"},{"requiredComponents":["tomcatServer_ID"],"providedMethods":[],"type":"DB","id":"DB1"},{"requiredComponents":["tomcat_server_VM_ID"],"providedMethods":[],"type":"tomcat_server","id":"tomcatServer_ID"},{"requiredComponents":[],"providedMethods":[],"type":"chat_webApplication","id":"chat_webApplication_ID"}],"methods":[{"type":"anyMethod","id":"anyMethod_ID"}]}
```

5.6 Executing and monitoring the application

After the application deployment, the execution and the monitoring of the application is started. In particular, during the application execution, the installed data collectors send data to the monitoring platform and especially, to the Data Analyzer component of the monitoring platform. In turn, the monitoring platform forwards these data to the Deployer and the Dashboard. This last one retrieves raw monitoring data through the Monitor and visualizes them.

In the next version of the SeaClouds Integrated Platform, the Deployer will use these data to evaluate its management policies. If a policy is violated, then the Deployer will try to fix this by repairing the deployment plan. If a repairing activity is not possible, then the Deployer will decide and trigger the re-planning of the current application via an interaction with the Monitor.

Finally, the Monitor uses the raw data for evaluating its monitoring rules. In particular, the Monitor processes and filters raw data based on the definition of the previously defined monitoring rules. For instance, the evaluation of the first monitoring rule, which measures the average response time of the application methods (see Section 5.4), will result in the following behavior. The data collector, which is in charge of monitoring the response time of each application method, sends monitoring data whenever each application method is called. The Data Analyzer component of the monitoring platform retrieves these data, partitions the data per method, and for each method, computes every 30 seconds the average response time of the method of the last 30 seconds of data (timeWindow=”30” timeStep=”30”). An example of the calculated average response time (in miliseconds) for a method of the application example is presented below.


```
{"http://www.modaclouds.eu/rdfs/1.0/monitoringdata#metric":[{"type":"literal","value":"Respo
nseTimeViolation" } ],

"http://www.modaclouds.eu/rdfs/1.0/monitoringdata#timestamp":[{"type":"literal","value" :
"1429028681315", "datatype": "http://www.w3.org/2001/XMLSchema#integer" } ],

"http://www.modaclouds.eu/rdfs/1.0/monitoringdata#value":[{"type:"literal",
"value":"47.0e0","datatype": "http://www.w3.org/2001/XMLSchema#double" } ],

"http://www.modaclouds.eu/rdfs/1.0/monitoringdata#resourceId":[{"type":"literal","value":"ch
at_webApplication-anyMethod" } ] }
```


In case the value of the calculated average response time is not lower than two seconds, then a violation of this rule is performed and an alert is triggered to the SLA service, which has

previously subscribed to such events. In this case, the SLA service receives the violation and stores it as QoS violation of this SLA agreement. In this way, a posterior analysis of the fulfillment of the agreed service could be performed.

In the next release of the integrated platform, a more complex definition of policies will be available, allowing the description of business penalties to apply when a guarantee term is violated.

6 Update on tools and practices for continuous integration and quality assurance

To ensure a good level of QA a completely free Continuous Integration (CI) and Continuous Distribution has been set up.

We identified travis-ci.org [https://travis-ci.org/SeaCloudsEU/SeaCloudsPlatform] as CI system to build the source code hosted on github [https://github.com/SeaCloudsEU/]. For each new commit against https://github.com/SeaCloudsEU/SeaCloudsPlatform in whatever branch of that repository, a new build on travis-ci is triggered. Basically, by configuring the github repository with a .travis.yml file it is now possible to run a build plus the unit tests defined for that repository, for any branch the developers want to use.

Additionally, as SeaClouds platform is built using java language, the build will also store all the resulting artifacts of a green build to a central repository. In fact, we agreed on distributing the artifacts generated from the source code, like jar file, war file, etc., to a well-know public managed maven repository hosted by Sonatype. This means that all the SNAPSHOTs created by a successful build will be automatically pushed to the `eu.seaclouds-project` space at SONATYPE snapshot repository [https://oss.sonatype.org/content/repositories/snapshots/eu/seaclouds-project/] as it is free for opensource projects like ours.

The above workflow based on free managed services (github, travis, sonatype) can be consumed by 2 main stakeholders: SeaClouds developers that can manually get access to an always up-to-date, tested and tracked set of binaries of the project.

As SeaClouds platform artifacts are published in a public easily accessible repository, SeaClouds consortium agreed in using Apache Brooklyn as the Deployer of the SeaClouds platform itself. By using most of the mechanism that we are developing for the SeaClouds end-users, SeaClouds team will be able to deploy in a number of different environment the SeaClouds platform by deploying the SeaClouds blueprint.

7 Updated integration plan

The next five months of the project will be dedicated to the production of the Final Integrated platform. This will include all functions foreseen for the SeaClouds project. In particular:

- The design-time GUI and the Discovery component will be released and integrated with the other components.
- The Deployer is being extended in the following main directions: i) to be able to handle TOSCA-based DAMs, ii) to extract from the DAM the information needed by the Monitoring Platform and the SLA Service and to pass this to them, and, most important iii) to be able to support PaaS-based deployment and iv) to support repairing of the application.
- The Planner is being extended with the application replanning feature and with the possibility of producing a more complete DAM that includes, besides the basic resource allocation information, also the monitoring rules and SLA associated to an application.

These extensions will lead to the architecture of Figure 5 and will greatly simplify the work of developers and operators of multi-cloud applications as they will be able to manage from a single GUI the whole life-cycle of their applications.

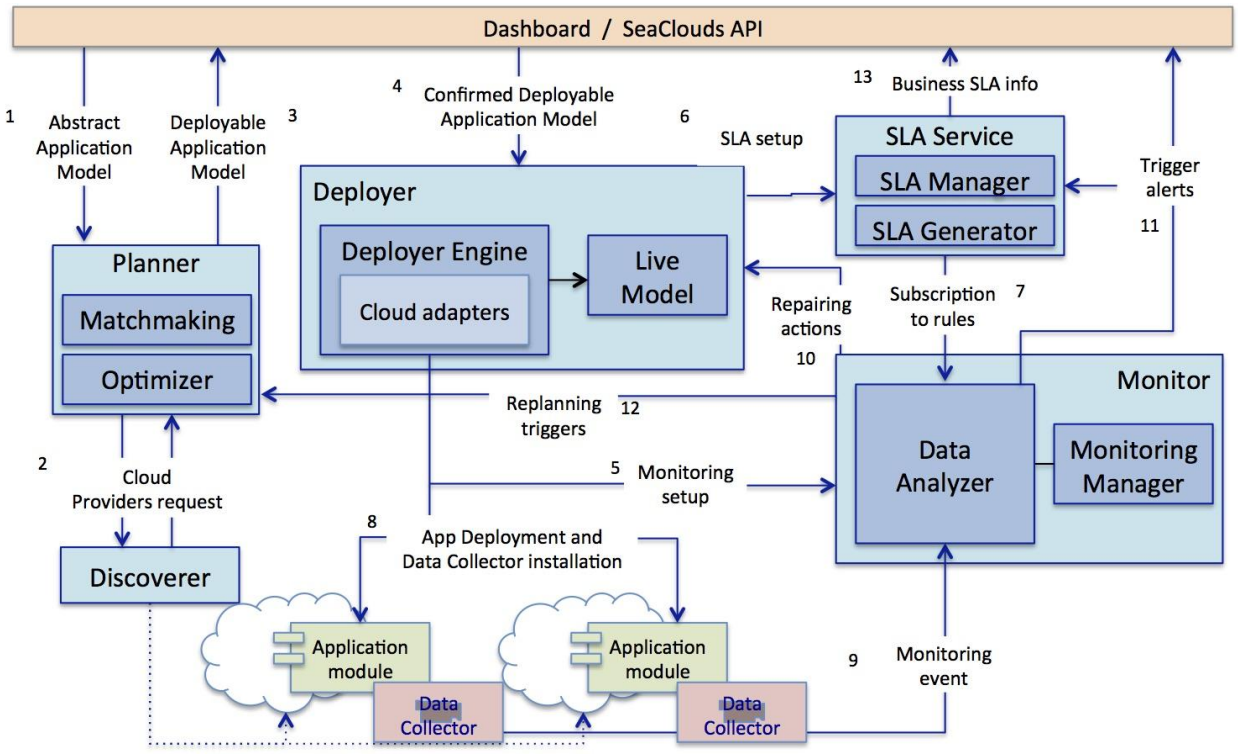

Figure 5: SeaClouds final integrated platform

8 Conclusions

The SeaClouds Integrated Platform is currently available in the form described in this deliverable but it will be continuously updated from now till the end of the project, following the continuous integration approach we have adopted. To reflect such evolution, the current document will be made available online as a live document and continuously updated in all its parts.

References

[1]. SeaClouds Project. Deliverable D2.4 Final SeaClouds Architecture (SeaClouds Consortium), To be published, 2015.

[2]. SeaClouds Project. Deliverable D3.1 Discovery, design and orchestration functionalities: Fiest specification (SeaClouds Consortium), http://seaclouds-project.eu/deliverables/SEACLOUDS-D3.1-Discovery_Design_and_Orchestration_Functionalities%20_First_Specification.pdf, 2014.

[3]. SeaClouds Project. Deliverable D4.1 Definition of the multi-deployment and monitoring strategies (SeaClouds Consortium), http://seaclouds-project.eu/deliverables/SEACLOUDS-D4.1_Definition_of_the_multi-deployment_and_monitoring_strategies.pdf, 2014.

[4]. SeaClouds Project. Deliverable D5.4.1 Initial version of the s/w platform (SeaClouds Consortium), http://seaclouds-project.eu/deliverables/SEACLOUDS-D5.4.1-Initial_version_of_sw_platform.pdf, 2014.

[5]. SeaClouds Project. Deliverable D5.1.1 Definition of the software developing environment (SeaClouds Consortium), http://seaclouds-project.eu/deliverables/SeaClouds-D5.1.1-Definition_of_the_software_developing_environment.pdf, 2014.

[6]. SeaClouds Project. Deliverable D4.3 Design of the run-time reconfiguration process (SeaClouds Consortium), To be published, 2015.

[7]. SeaClouds Project. Deliverable D4.4. Dynamic QoS Verification and SLA Management Approach (SeaClouds Consortium), To be published, 2015.

[8]. SeaClouds Project. Deliverable D5.2.2 Final design of the user interface (SeaClouds Consortium), To be published, 2015.

[9]. Model-driven approach for design and execution of applications on multiple clouds, Project is partially Funded by European Commission Grant no. FP7-ICT-2011-8-318484, 2013-2015, https://github.com/deib-polimi/modaclouds-monitoring-manager/wiki

[10]. SeaClouds Project. Deliverable D4.5. Unified Dashboard and Revision of Cloud API (SeaClouds Consortium), March 2015.

[11]. Apache brooklyn deployment blueprint https://brooklyn.incubator.apache.org/v/latest/start/blueprints.html

[12]. SeaClouds Project. Deliverable D3.2. Discovery, design and orchestration functionalities (SeaClouds Consortium), To be published, 2015.

[13]. TOSCA Simple Profile in YAML Version 1.0 . http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html

[14]. Web Services Agreement Specification, http://www.ogf.org/documents/GFD.192, Open Grid Forum, 2011.

[15]. Jose Carrasco, Javier Cubo, Ernesto Pimentel. Towards a Flexible Deployment of Multi-cloud Applications Based on TOSCA and CAMP. Advances in Service-Oriented and Cloud Computing, Communications in Computer and Information Science Volume 508, 2015, pp 278-286, 2015.

[16]. Brooklyn YAML Blueprint Reference, http://brooklyncentral.github.io/v/0.7.0-SNAPSHOT/use/guide/defining-applications/yaml-reference.html, CloudSoft, 2014.