



---

## SeaClouds Project

### D5.1.3 – Final 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	1 <sup>st</sup> October 2013
Ending Date	31 <sup>st</sup> March 2016
Work Package	WP5 Integration, infrastructure delivery and GUI
Deliverable code	D5.1.3
Deliverable Title	Final Integrated Platform
Nature	Prototype
Dissemination Level	Public
Due Date:	M24
Submission Date:	20th of October 2015
Version:	1.0
Status	Final
Author(s):	Miguel Barrientos (UMA), Jose Carrasco (UMA), Javier Cubo (UMA), Elisabetta Di Nitto (Polimi), Michele Guerriero (Polimi), Adrián Nieto Pérez (UMA), Marc Oriol (UPI), Diego Pérez (Polimi), Román Sosa (ATOS), Andrea Turli (Cloudsoft), Simone Zenzaro (UPI)
Reviewer(s)	Javier Cubo (UMA)

### 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	26/08/2015	ToC and distribution of work	Elisabetta Di Nitto
0.2	19/09/2015	Sect 2, final integrated platform	Diego Pérez
0.3	29/09/2015	Check contributions and corrections along all the deliverable	Diego Pérez
0.4	14/10/2015	internal review	Javier Cubo
1.0	19/10/2015	Final version	Diego Pérez, Elisabetta Di Nitto

## Table of Contents

Executive Summary.....	4
1 Introduction.....	5
1.1 Scope and outcome of the Deliverable.....	5
1.2 Structure of the document .....	5
1.3 List of Acronyms.....	6
2 Architecture of SeaClouds Final Integrated Platform .....	7
3 How to use the SeaClouds Integrated Platform: an example .....	11
3.1 Description of the Application Example .....	11
3.2 Definition of the application topology and of the Abstract Application Model.....	14
3.3 From AAM to TOSCA DAM.....	19
3.4 Definition of Monitoring Rules and SLA.....	22
3.5 Deployment on IaaS/PaaS.....	26
3.6 Execution, visualization of monitoring data, autoscaling .....	28
4 Conclusions.....	30
References .....	31

## List of Figures

Figure 1: Final Integrated Platform.....	7
Figure 2: Intermediate Integrated Platform presented in D5.1.2 .....	9
Figure 3: Final Architecture presented in [2].....	11
Figure 4: Application Chat topology .....	13
Figure 5: First step of SeaClouds wizard .....	15
Figure 6: Second step of SeaClouds wizard .....	16
Figure 7: SeaClouds Deployer - Deployment process and summary.....	27
Figure 8: Legacy monitoring view with autoscaling status.....	29
Figure 9: Tower4Clouds + Graphite + Grafana monitoring platform .....	29
Figure 10: SLA View showing a violation .....	30

## List of Tables

Table 1: Acronyms.....	6
Table 2: Seaclouds Integrated components description .....	9

## Executive Summary

This deliverable is the final 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 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, check the SLA (Service Level Agreement), and reconfigure an example application.

# 1 Introduction

## 1.1 Scope and outcome of the Deliverable

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

1. This document that aims at guiding users of the SeaClouds Final Integrated Platform through the identification of the main components of the current platform and the relationships between them.
2. An application example, together with the artifacts needed to describe it (AAM and ADP explained in [1] in the TOSCA specification, and DAM also explained in [1] in the TOSCA and CAMP format), the cloud offers that match the application, the associated monitoring rules and the SLA service.

The previous version of the Integrated Platform required some manual steps for supporting the lifecycle of a multi-cloud application, from the definition of the application topology down to the deployment phase. The current version features a complete GUI that supports the users in going through all phases and makes complex details transparent to all of them. Moreover, the Final Integrated Platform supports deployment on PaaS and automatic repair of applications. The platform will be evolved till the end of the project to improve the support to application repair/replan and to address the new requirements that will be raised by its users.

The SeaClouds project is continuing to pursue a fully open source approach not only releasing all software with an Apache 2.0 license on a github repository <https://github.com/SeaCloudsEU>, but also exploiting and contributing to other open source research initiatives such as and Brooklyn<sup>1</sup> and Alien4Clouds<sup>2</sup>.

The SeaClouds Final Integrated Platform follows the architecture defined in Deliverable D2.4 [2].

## 1.2 Structure of the document

This document has the following structure:

- Section 2 describes the SeaClouds Final Integrated Platform and the changes with respect to the architecture in Deliverable D2.4 [2], and the previous version of the Integrated Platform [3].
- Section 3 shows how the user can exploit the Integrated Platform to deploy an example application.

---

<sup>1</sup> <https://brooklyn.incubator.apache.org/>

<sup>2</sup> <http://alien4cloud.github.io>

- Finally, Section 4 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 Architecture of SeaClouds Final Integrated Platform

This section describes the Final Integrated Software Platform, highlighting the updates in the architecture with respect to the intermediate software Platform released at M19 and described in D5.1.2 [3] to bring it closer to the Final Architecture defined in D2.4 [2].

Overall, few modifications have been introduced in the architecture delivered in D5.1.2 to achieve the required functionality of components, whereas some modifications have been introduced to achieve better communication among components and Platform integration.

Figure 1 depicts the architecture of the final SeaClouds integrated platform. To make easier the description of the modifications made in the last months, Figure 2 presents the architecture shown in D5.1.2 [3] and Figure 3 shows the final architecture described in D2.4 [2].

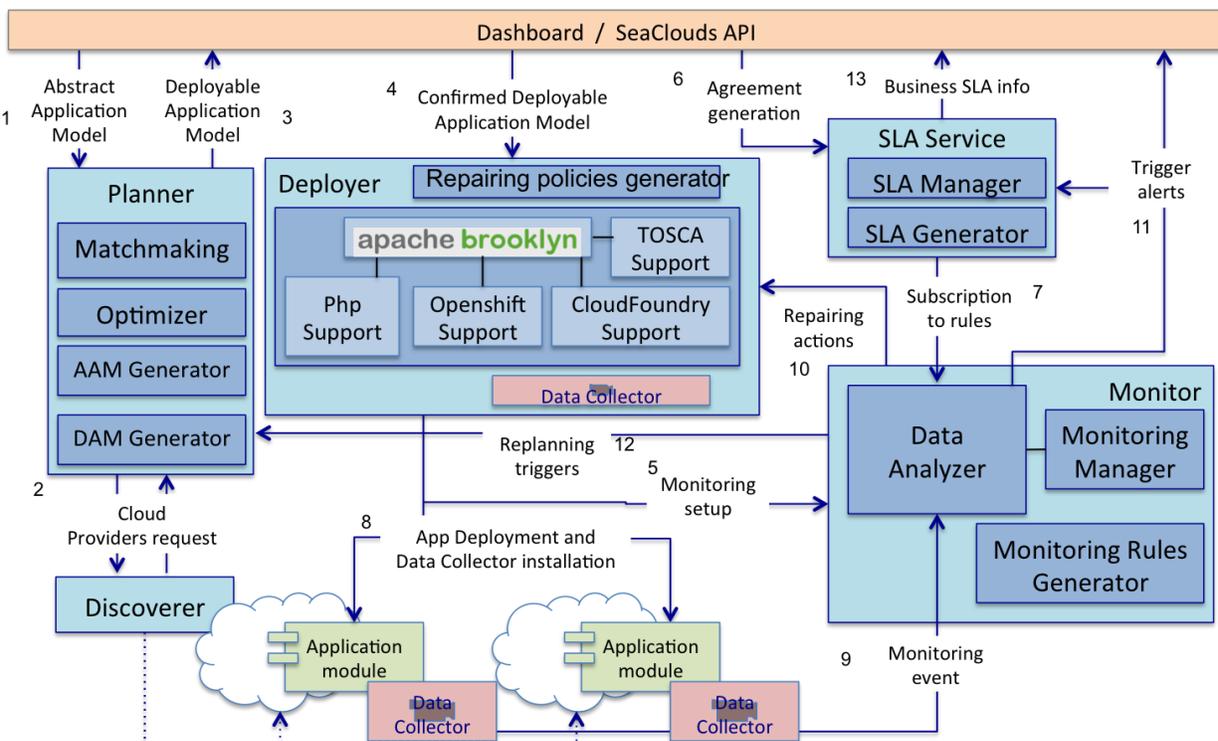


Figure 1: Final Integrated Platform

### SeaClouds components:

Seaclouds components continue to be released under the Apache 2.0 license and are available at the URL <https://github.com/SeaCloudsEU/SeaCloudsPlatform>. Components called *Dashboard*, *Planner*, *Deployer*, *SLA Service* and *Monitor* (incorporating the Tower4Clouds<sup>3</sup> component developed as part of MODAClouds<sup>4</sup>) were already integrated and described in the intermediate integrated platform in

<sup>3</sup> <https://github.com/deib-polimi/tower4clouds/>

<sup>4</sup> <http://www.modaclouds.eu>

D5.1.2. A significant difference can be appreciated in the *Deployer* component in Figure 1. The reason is that now we can give concrete details of the previous *Deployer engine* and *Cloud adapters* modules that were part of the *Deployer* component showed in Figures 2 and 3. These previous modules are replaced with Brooklyn as *Deployer engine*; and by *Php support*, and *Openshift support*, *CloudFoundry support* as *Cloud adapters*. Moreover, a *TOSCA support* module is also added, at present under development, which is in charge of providing Brooklyn with capabilities to deploy application specified in TOSCA and will be eventually included in the official Brooklyn release. Finally, *Live Model* is no longer part of the *Deployer* architectural view since it is information managed internally by Brooklyn. Another component that was not integrated in the intermediate platform, as it does not appear in Figure 2, and now it is part of the final integrated platform is the *Discoverer*.

Table 2 shows again the description of SeaClouds components, highlighting in boldface the differences with respect to their characteristics provided in D5.1.2. We have added an additional column to the table in order to provide a better distinction between the external libraries used by components, their subcomponents and their dependencies with other first-level SeaClouds components.

Component name	Programming language	External libraries used	Subcomponents	Dependencies with other components	Offered software interfaces
Discoverer	Java	Apache Tomcat, JJsonSimple, Alien4Cloud TOSCA parser	Cloud Offerings Spiders	None	REST API
Dashboard	HTML5, JavaScript, Java	Bootstrap library <sup>5</sup> , Angular JS <sup>6</sup> , Dropwizard <sup>7</sup>		Planner (including new dependency with <b>AAMgenerator</b> ), Deployer, Monitor, SLA Service	REST API
Planner	Java	Dropwizard, Alien4Cloud TOSCA parser, SnakeYaml	Matchmaker, Optimizer, <b>AAM generator, DAM generator</b>	<b>Discoverer, Dashboard, Monitor, Deployer</b>	REST API
Deployer	Java	Apache Brooklyn	Deployer Engine (detailed to Brooklyn, Php support, Ppenshift support CloudFoundry support and TOSCA support ), <b>Repairing rules generator</b>	Planner, Monitor	REST API

<sup>5</sup> <http://getbootstrap.com>

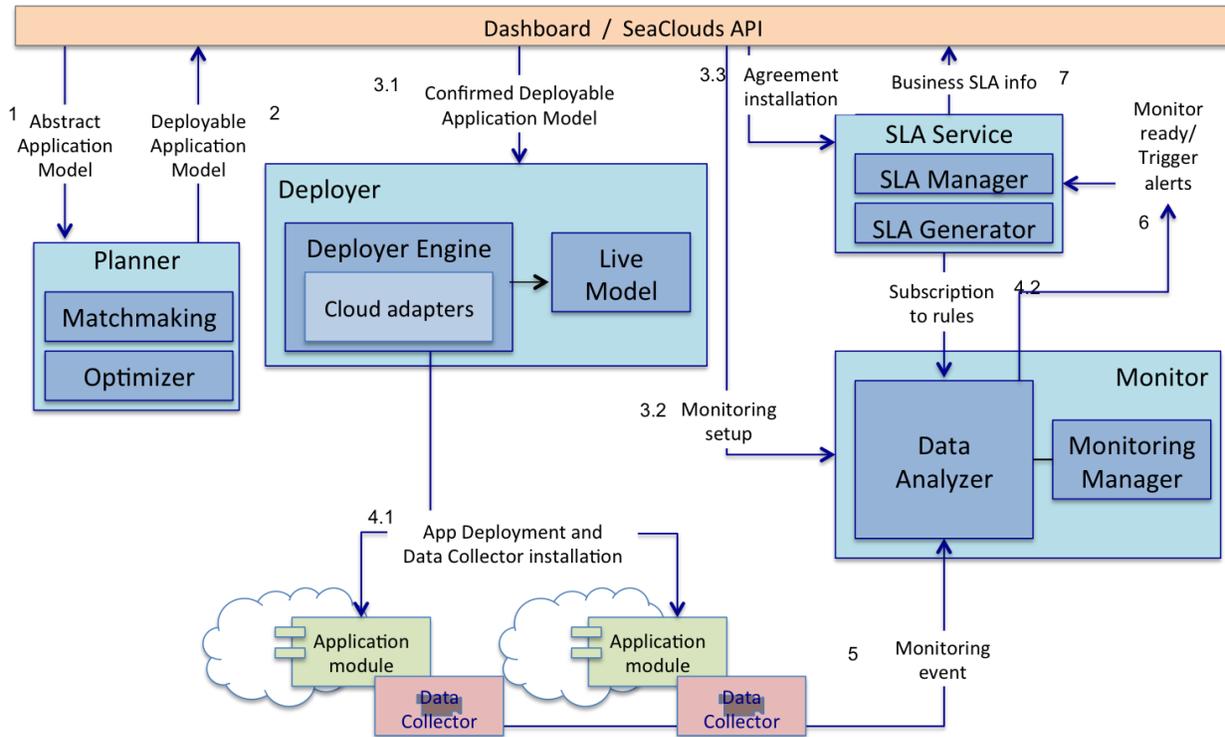
<sup>6</sup> <https://angularjs.org>

<sup>7</sup> <http://www.dropwizard.io/>

SLA Service	Java	jersey-1.18.1, spring-4.1.4, jpa-2.0-api, jackson-2.5.1, qos-models-2.4	SLA manager, SLA generator	Monitor, particular, addObserver and sendMonitoringRule nstalled functions	in REST API
Monitor	Java	fuseki-server.jar <sup>8</sup> , rsp-services-csparql.jar <sup>9</sup> , snakeyaml.jar	Monitoring Manager, Data Collector, Data Analyzer, <b>Monitoring Rules Generator</b>	Deployer, Service, Dashboard	SLA and REST API

**Table 2: Seaclouds Integrated components description**

Looking at Figures 1 and 3 we can see that all the components of the Final Architecture have already been integrated in the platform.



**Figure 2: Intermediate Integrated Platform presented in D5.1.2**

**SeaClouds inter-component communication**

Significant effort has been done in the last months towards the interconnection of components, accepted inputs of each component and produced outputs. This effort has led to the inclusion of new modules in the main architectural components, which do not add new functionality (i.e., component

<sup>8</sup> <http://archive.apache.org/dist/jena/binaries>  
<sup>9</sup> <https://github.com/deib-polimi/rsp-services-csparql>

behavior given in previous D5.1.2 is completely valid), but create decoupled code to manage inputs and outputs that are easier to understand by other developers. Concretely, these new modules are:

- **AAM Generator** in *Planner* component. This module is in charge of creating the AAM in the same TOSCA syntax as the *Planner* understands. The *Dashboard* component imports this module in order to create appropriate AAM models.
- **DAM generator** in *Planner* component. This module is in charge of creating the TOSCA DAM that will be used by the rest of the modules with the information included in the format that is understood by *Dashboard*, *SLA Service*, *Monitor* and *Deployer*.
- **Monitoring rules generator** in *Monitor* component. This module creates the monitoring rules in the syntax that is understood by the Data Analyzer. Since these monitoring rules are included in the DAM, this new module helps the DAM generator module in *Planner* component to create them in the correct syntax.
- **Repairing policy generator** in *Deployer* component. This module creates the information for application repairing (e.g., information for scaling up/down the application or restart single modules) in the syntax understood by the Deployer Engine (i.e., Apache Brooklyn). Since this policy is included in the DAM, this new module helps the DAM generator in *Planner* to create it in the correct syntax.
- **New Data Collector** in *Deployer*. This Data collector is used by the *Monitor* to collect runtime information regarding the application's health. The *Monitor* uses this information to trigger alerts of application malfunction to the *Dashboard* when necessary.

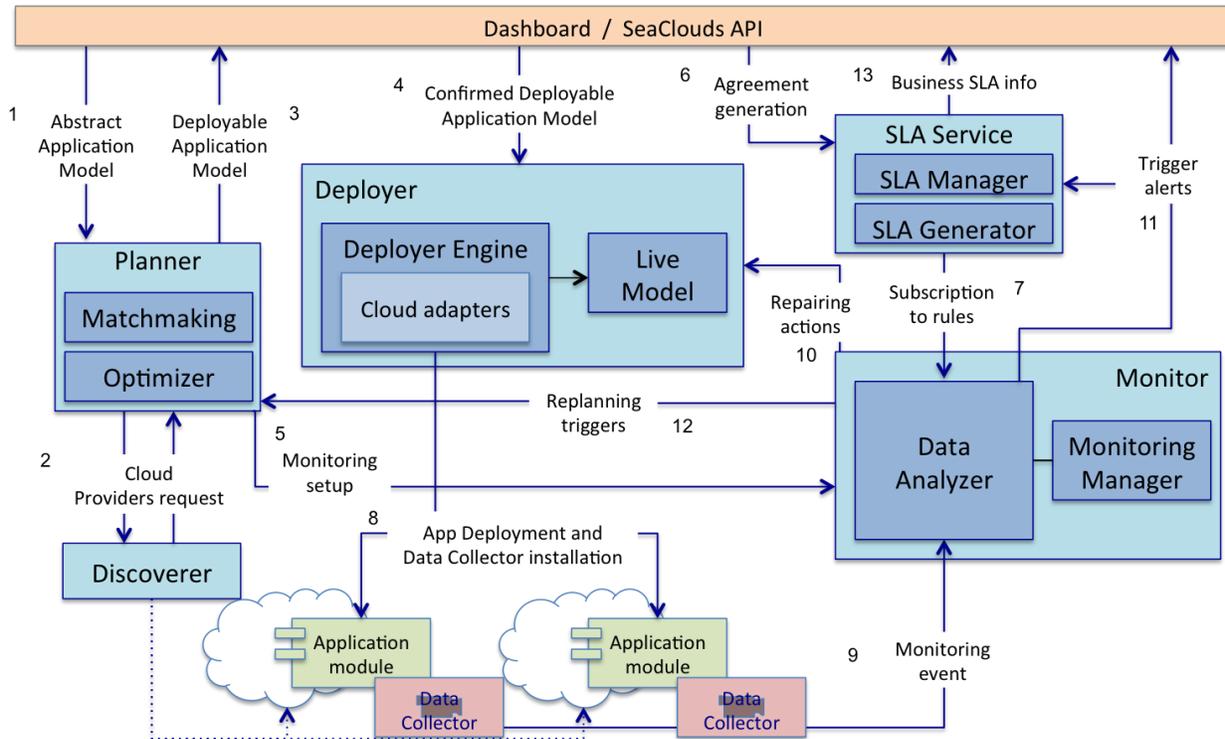


Figure 3: Final Architecture presented in [2]

### 3 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 [4]. This application is part of the *Brooklyn blueprint* and provides enough complexity to exemplify the current Seaclouds capabilities. This document does not describe the installation process of SeaClouds Platform since such process is detailed in SeaClouds Deliverable D5.4.2 [5], which is delivered concurrently with this document. In the following subsections we describe the functionality and structure of the application (Section 3.1), then we describe how the SeaClouds user can define the application topology using the SeaClouds GUI thus defining the corresponding AAM (Abstract Application Model) (Section 3.2). We show how, starting from this AAM, the matchmaking and optimization process works (Section 3.3). After that, we describe the monitoring rules and the SLA associated to the application (Section 3.4) and the deployment of the corresponding application (Section 3.5). Finally, we show how the application is executed, monitored and repaired (Section 3.6).

#### 3.1 Description of the Application Example

The following description has been integrally taken from Deliverable 5.1.2 and is included here to make this deliverable self-contained.

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.

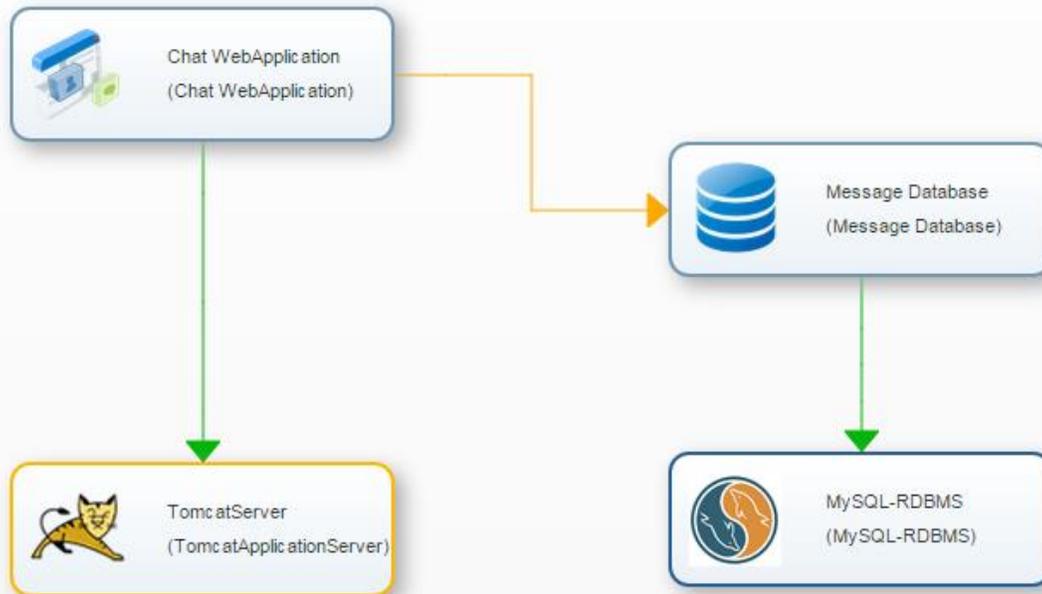
**Web interface is packaged in a .war (Web application ARchive) file `chat-webApplication.war`<sup>10</sup> which uses an external database to store and retrieve students' information.**

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 4 shows the architectural topology of the application example.

---

<sup>10</sup> <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 4: 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 an 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.
- Be aware of the application behavior and, when necessary, perform reconfigurations actions such as scale in/out the application or reboot failing resources.

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 appropriate, given the characteristics of the application and the trade-off between service characteristics and cost; and 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 this release of the SeaClouds platform, we also tackle selection and usage of PaaS services and, partially, 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:

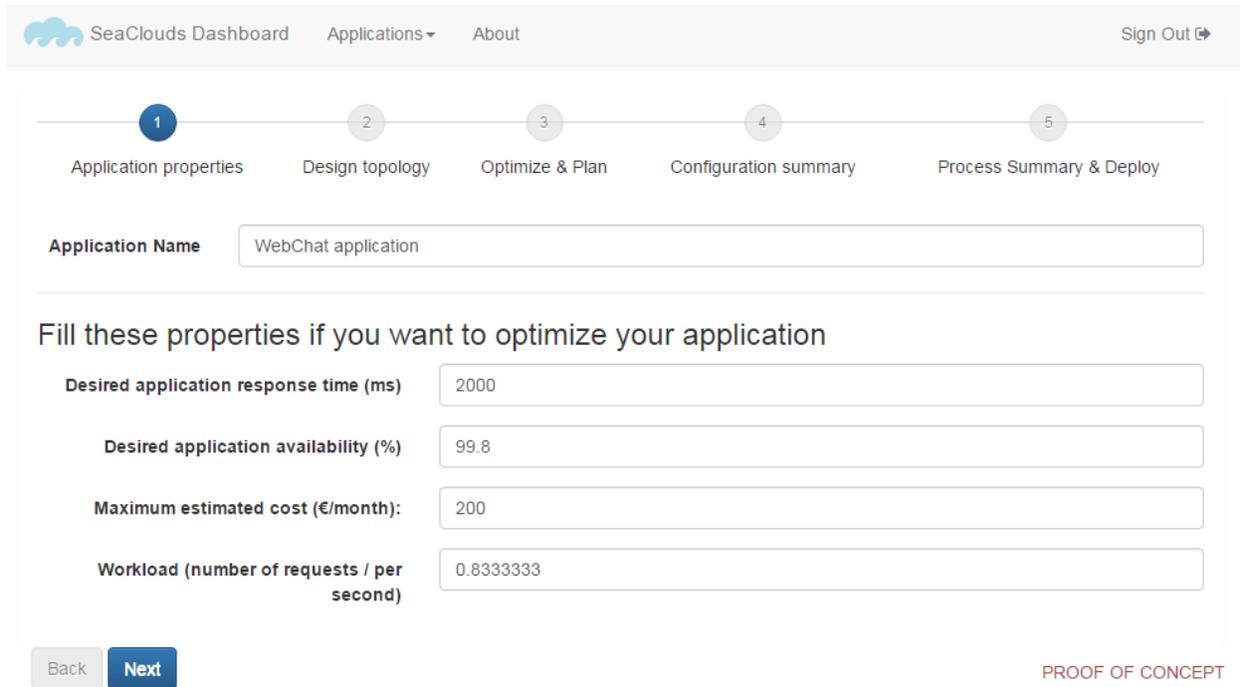
- The database is MySQL 5.0 and needs 50GB of size.
- The application server has to be able to execute Java.
- The application availability should be higher than 99.8%.
- The application expected response time is lower than 2 seconds for an arrival rate of 50 messages per minute.
- The chat owner organisation 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 studying 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.2x1`.

In order to demonstrate the new capability of SeaClouds *Deployer* component to manage application deployments on PaaS, we also impose the requirement “the application server where `chat-webApplication.war` is deployed must be a PaaS”.

### 3.2 Definition of the application topology and of the Abstract Application Model

The application topology is entered by the user through the New Application wizard in the User Interface. The first step in the wizard, shown in Figure 5, asks for the name of the application and a set of properties needed if the user wants to optimize the resources used by the application.



SeaClouds Dashboard Applications About Sign Out

1 Application properties 2 Design topology 3 Optimize & Plan 4 Configuration summary 5 Process Summary & Deploy

Application Name WebChat application

Fill these properties if you want to optimize your application

Desired application response time (ms) 2000

Desired application availability (%) 99.8

Maximum estimated cost (€/month): 200

Workload (number of requests / per second) 0.8333333

Back Next

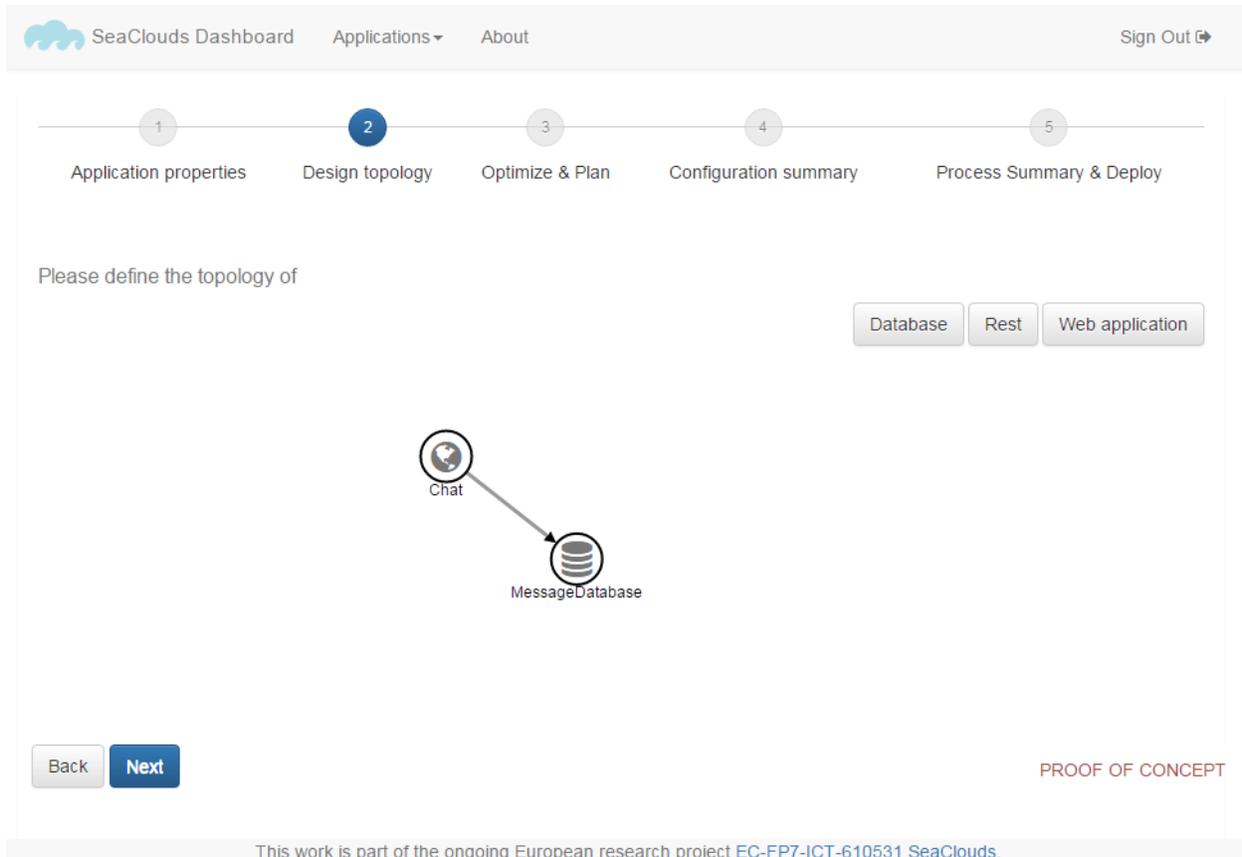
PROOF OF CONCEPT

Figure 5: First step of SeaClouds wizard

The second step in the wizard, illustrated in Figure 6, allows the user to define the components that the application is composed of. For each component, the user has to select:

- The type of component, among web application, database and message queue (other types of services will be added when required).
- Technological requirements: language of the component or type of database, versions, etc.
- QoS requirements for the module.
- Desired infrastructure where to deploy the component on: IaaS or PaaS.
- Dependency relationships between components.

The complete description of types and properties are in deliverable D5.3 "Implementation of the User Interface" [6].



**Figure 6: Second step of SeaClouds wizard**

This wizard step produces a model of the application, in JSON<sup>11</sup> format, shown in Listing 1.

```

{
  "name": "WebChat application",
  "nodes": [
    {
      "name": "Chat",
      "type": "WebApplication",
      "properties": {
        "language": "JAVA",
        "artifact": "http://www.seaclouds.eu/artifacts/chat-webApplication.war",
        "min_version": "6",
        "infrastructure": "platform",
        "container": "webapp.tomcat.TomcatServer",
        "benchmark_rt": "50",
        "benchmark_platform": "hp_cloud_services.2x1"
      }
    },
    {
      "name": "MessageDatabase",
      "type": "Database",
      "properties": {

```

<sup>11</sup> <http://json.org>

```

        "category": "database.mysql.MySqlNode",
        "artifact": "http://www.seaclouds.eu/artifacts/create-message-database.sql",
        "min_version": "5.0",
        "max_version": "5.0",
        "disk_size": "50",
        "infrastructure": "compute"
        "benchmark_rt": "30",
        "benchmark_platform": "hp_cloud_services.2x1"
    }
}
],
"links": [
    {
        "source": "Chat",
        "target": "MessageDatabase",
        "properties": {
            "calls": "2"
        }
    }
],
"application_requirements" {
    "response_time": "2000",
    "availability": "0.998",
    "cost": "200",
    "workload": "50"
}
}

```

**Listing 1: Application model in JSON format**

An improvement on the integration of different SeaClouds components from the delivery of Deliverable 5.1.2 is that the JSON model produced by the wizard is now translated to the TOSCA YAML read by the *Planner* component. Therefore, once the application components have been modeled, the next step in the wizard invokes the *AAM Generator*. This SeaClouds module takes the JSON model as input and generates the AAM, specified in TOSCA, representing the application. The AAM obtained from the webchat JSON model is shown in Listing 2.

```

tosca_definitions_version: tosca_simple_yaml_1_0_0_wd03
description: WebChat application
imports:
- tosca-normative-types:1.0.0.wd03-SNAPSHOT
topology_template:
  node_templates:
    Chat:
      type: sc_req.Chat
      artifacts:
      - war: http://www.seaclouds.eu/artifacts/chat-webApplication.war
        type: tosca.artifacts.File
      requirements:
      - endpoint: MessageDatabase
    MessageDatabase:
      type: sc_req.MessageDatabase
      artifacts:
      - db_create: http://www.seaclouds.eu/artifacts/create-message-database.sql
        type: tosca.artifacts.File

```

```

properties:
  mysql_version:
    constraints:
      - greater_or_equal: '5.0'
      - less_or_equal: '5.0'
node_types:
  sc_req.Chat:
    derived_from: seaclouds.nodes.webapp.tomcat.TomcatServer
    properties:
      java_support:
        constraints:
          - equal: true
      tomcat_support:
        constraints:
          - equal: true
      java_version:
        constraints:
          - greater_or_equal: '6'
      resource_type:
        constraints:
          - equal: platform
  sc_req.MessageDatabase:
    derived_from: seaclouds.nodes.database.mysql.MySqlNode
    properties:
      disk_size:
        constraints:
          - greater_or_equal: '50'
      resource_type:
        constraints:
          - equal: compute
groups:
  operation_Chat:
    members:
      - Chat
    policies:
      - QoSInfo:
          execution_time: 50 ms
          benchmark_platform: hp_cloud_services.2x1
      - dependencies:
          operation_MessageDatabase: '2'
      - QoSRequirements:
          response_time:
            less_than: 2000.0 ms
          availability:
            greater_than: 0.998
          cost:
            less_or_equal: 200.0 euros_per_month
          workload:
            less_or_equal: 50.0 req_per_min
  operation_MessageDatabase:
    members:
      - MessageDatabase
    policies:
      - QoSInfo:
          execution_time: 30 ms
          benchmark_platform: hp_cloud_services.2x1
      - dependencies: {}

```

Listing 2: AAM in TOSCA format

### 3.3 From AAM to TOSCA DAM

The AAM is sent to the *Planner* through the *Dashboard*. The *Planner* interacts then with the *Discoverer*, which provides a list of Clouds Offerings from service providers with information regarding their technical characteristics and QoS information. An example of a cloud offering provided by the *Discoverer* is as shown below. The actual data in the offering description is intended to be just an example and currently it is not real information from the cloud providers.

```
tosca_definitions_version: tosca_simple_yaml_1_0_0_wd03
description:
  template_name:
  template_version: 1.0.0-SNAPSHOT
  template_author:

imports:
  - tosca-normative-types:1.0.0.wd03-SNAPSHOT

topology_template:
  node_templates:
    Microsoft_Azure_Virtual_Machines_G5_us_east:
      type: seaclouds.Nodes.Compute.Microsoft_Azure_Virtual_Machines_G5_us_east
      properties:
        resource_type: compute
        hardwareId: G5
        location: "azure:compute"
        region: "us-east"
        availability: 0.99950
        performance: 537 SPECfp
        country: United States
        city: ASHBURN
        cost: 8.69 USD/hour
        local_storage: 6144
        num_disks: 1
        num_cpus: 32
        ram: 448
        disk_type: ssd
```

The different components of the planner provide the functionalities required to generate the DAM. In particular, 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 requirements of the abstract service defined in the AAM.

The AAM and the list of candidate cloud services for each module is passed to the *Optimizer*. This module creates ADP models, where in each ADP model, each of the application modules in the AAM is associated with one of its candidate cloud offers. The association of application modules with cloud offers intends to satisfy performance, availability and cost QoS requirements of the application; therefore it requires inputs of the performance, availability and cost of each candidate cloud offer. The selection of the candidate offer is expressed in the ADP 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. The

following listing shows an example of ADP. See, for instance, the new `Amazon_EC2_i2_xlarge_ap_china_1` node template that provides the host specification of `MessageDatabase` node template.

```

tosca_definitions_version: tosca_simple_yaml_1_0_0_wd03
description: WebChat application
imports:
- tosca-normative-types:1.0.0.wd03-SNAPSHOT
topology_template:
  node_templates:
    Chat:
      type: sc_req.Chat
      artifacts:
      - war: http://www.seaclouds.eu/artifacts/chat-webApplication.war
        type: tosca.artifacts.File
      requirements:
      - endpoint: MessageDatabase
      - host: pivotal

    MessageDatabase:
      type: sc_req.MessageDatabase
      artifacts:
      - db_create: http://www.seaclouds.eu/artifacts/create-message-database.sql
        type: tosca.artifacts.File
      properties:
        mysql_version:
          constraints:
          - greater_or_equal: '5.0'
          - less_or_equal: '5.0'
      requirements:
      - host: Amazon_EC2_i2_xlarge_ap_china_1

    Amazon_EC2_i2_xlarge_ap_china_1:
      type: seaclouds.Nodes.Compute.Amazon_EC2_i2_xlarge_ap_china_1
      properties:
        resource_type: compute
        hardwareId: i2.xlarge
        location: "aws:ec2"
        region: "ap-china-1"
        performance: 90
        availability: 0.99950
        country: China
        city: BEIJING
        cost: 1.001 USD/hour
        disk_size: 800
        num_disks: 1
        num_cpus: 4
        ram: 30.5
        disk_type: ssd

    pivotal:
      type: seaclouds.nodes.Platform.Pivotal
      properties:
        resource_type: platform
        go_support: true
        java_support: true
        node_support: true
        php_support: true
        python_support: true
        ruby_support: true
        mysql_support: true
        postgresql_support: true
        mongoDB_support: true

```

```

redis_support: true
riak_support: true
dataStax_support: true
neo4j_support: true
pivotalHD_support: true
cost: 0.03 USD_perGB_per_h
java_version: 7

```

node\_types:

sc\_req.Chat:

derived\_from: seaclouds.nodes.webapp.tomcat.TomcatServer

properties:

java\_support:

constraints:

- equal: true

tomcat\_support:

constraints:

- equal: true

java\_version:

constraints:

- greater\_or\_equal: '6'

resource\_type:

constraints:

- equal: platform

sc\_req.MessageDatabase:

derived\_from: seaclouds.nodes.database.mysql.MySqlNode

properties:

disk\_size:

constraints:

- greater\_or\_equal: '50'

resource\_type:

constraints:

- equal: compute

groups:

operation\_Chat:

members:

- Chat

policies:

- QoSInfo:

execution\_time: 50 ms

benchmark\_platform: hp\_cloud\_services.2x1

- dependencies:

operation\_MessageDatabase: '2'

- QoSRequirements:

response\_time:

less\_than: 2000.0 ms

availability:

greater\_than: 0.998

cost:

less\_or\_equal: 200.0 euros\_per\_month

workload:

less\_or\_equal: 50.0 req\_per\_min

- ExpectedQuality: {expectedWorkload: 50.0, expectedAvailability: 0.9994997551225, expectedCost: 190.8, expectedExecutionTime: 0.10079721294968995}

operation\_MessageDatabase:

members:

- MessageDatabase

policies:

- QoSInfo:

```
execution_time: 30 ms
benchmark_platform: hp_cloud_services.2x1
- dependencies: {}
```

**Listing 3: ADP in TOSCA format**

The list of ADPs is sent back to the user who selects the most suitable deployment plan to her interests. The selected ADP is sent again to the *Planner*, in particular, to the *DAM Generator* module. The *DAM Generator* augments the information specified in the ADP and generates a Deployable Application Model (DAM). Thus, the DAM contains the information needed by the SeaClouds *Deployer* to deploy, configure and execute the application. In particular:

- Monitoring rules
- Service Level Agreements
- Credentials

This information is added by different services orchestrated by the *DAM Generator*. In particular the monitoring rules are generated by a service invocation to the new *Monitoring rules generator* module in the *Monitor* component (module previously described in Section 2). The SLA is generated by the *SLA generator* module in the *SLA service* component. Next subsection details the generation of monitoring rules and SLA. Both of them will be XML documents. At present, these documents are attached to the DAM and we are under development of more appropriate solutions, for example, upload the monitoring rules and SLA documents to a document server and include in the DAM only the URLs of their location.

### 3.4 Definition of Monitoring Rules and SLA

This subsection details the generation of the information to be included in the DAM and related to the observation of the application behavior. This information is composed of the **monitoring rules, deployment scripts of data collectors** and the **SLA**.

The *Monitoring Rules Generator* module, which is part of the SeaClouds *Monitor* component, executes as main tasks the generation of monitoring rules and the deployment scripts for data collectors. These two tasks are interrelated because the monitoring rules are based on the metrics specified in the deployment scripts of the data collectors.

The *Monitoring Rules Generator* generates by default a couple of monitoring rules for infrastructural level monitoring (achieved exploiting the Sigar<sup>12</sup> based data collector). In particular monitoring rules for RAM and CPU usage of each application module will be generated by default. Also default application level monitoring is provided exploiting

---

<sup>12</sup> <http://sigar.hyperic.com/>

application level data collectors to measure properties such as the application response time or its throughput. This feature is provided for Java applications customized with `java-app-dc`<sup>13</sup>, available within the Tower4Clouds<sup>14</sup> framework. Non Java applications can bring their own data collector to feed the system with application level metrics. In this case the *Monitoring Rules Generator* also needs to be customized with an additional module to generate specific rules and deployment script for the custom data collector.

The described set of rules is mainly devoted to metrics visualization. A second set of rules enabling the monitoring of the SLAs violations is then generated based on the QoS requirements specified in the ADP.

Summarizing the *Monitoring Rules Generator* will output:

- 1) A set of monitoring rules for the SLA contracting process derived from the QoS requirement specified in the ADP.
- 2) A second set of rules for infrastructural level monitoring (RAM and CPU usage) for each application module when the deployment is performed in IaaS.
- 3) If it is a Java application equipped with the `java-app-dc`, it is also provided a third set of monitoring rules for application level monitoring.
- 4) Deployment script for infrastructural level Data Collector to be attached to every application module

Listing 4 reports the three monitoring rules generated for `Chat` module of the example application.

```
<monitoringRules>
  <monitoringRule id="respTimeRule_Chat" timeStep="10" timewindow="10">
    <monitoredTargets>
      <monitoredTarget type="Chat" class="InternalComponent"/>
    </monitoredTargets>
    <collectedMetric metricName="AvarageResponseTimeInternalComponent"/>
    <actions>
      <action name="OutputMetric">
        <parameter name="metric">AvarageResponseTime_Chat</parameter>
        <parameter name="value">METRIC</parameter>
        <parameter name="resourceId">ID</parameter>
      </action>
    </actions>
  </monitoringRule>

  <monitoringRule id="respTimeSLARule_Chat" timeStep="10" timewindow="10">
    <monitoredTargets>
```

<sup>13</sup> <https://github.com/deib-polimi/tower4clouds/tree/master/data-collectors/java-app-dc>

<sup>14</sup> <https://github.com/deib-polimi/tower4clouds>

```

        <monitoredTarget type="Chat" class="InternalComponent"/>
    </monitoredTargets>
    <collectedMetric metricName="AvarageResponseTimeInternalComponent"/>
    <condition>METRIC &gt; 2000.0</condition>
    <actions>
        <action name="OutputMetric">
            <parametername="metric">
                AvarageResponseTime_Chat_Violation
            </parameter>
            <parameter name="value">METRIC</parameter>
            <parameter name="resourceId">ID</parameter>
        </action>
    </actions>
</monitoringRule>

<monitoringRule id="vmAvailableSLARule_Chat" timeStep="2" timeWindow="2">
    <monitoredTargets>
        <monitoredTarget type="Chat" class="InternalComponent"/>
    </monitoredTargets>
    <collectedMetric metricName="AppAvailable"/>
    <metricAggregation groupingClass="InternalComponent" aggregateFunction="Average"/>
    <condition>METRIC &lt; 0.998</condition>
    <actions>
        <action name="OutputMetric">
            <parameter name="metric">
                AvarageAppAvailability_Chat_Violation
            </parameter>
            <parameter name="value">METRIC</parameter>
            <parameter name="resourceId">ID</parameter>
        </action>
    </actions>
</monitoringRule>
</monitoringRules>

```

**Listing 4: monitoring rules for Chat module of the WebChat application**

The first monitoring rule, which has `respTimeRule_Chat` as ID, belongs to the third set of generated metrics in the previous list. In turn, second and third monitoring rules, which have `respTimeSLARule_Chat` and `vmAvailableSLARule_Chat` as IDs respectively, belong to the first set of metrics and are generated because the application has availability and response time requirements.

After the monitoring rules have been generated, the next step in the production of the DAM document is the generation of the SLA. In this step, the *SLA Generator* module generates an SLA agreement to be assessed by the *SLA Service*. The generated agreement (refer to WS-Agreement [7] specification or deliverable D4.4 [8]) contains guarantee terms associated to the expressed application level constraints, namely application response time and availability. If there were specified actions to be applied in case of violations on these terms, they will be also present in the business part of the respective guarantee terms.

The SLA Service depends on the Monitor component to detect violations of the QoS. Concretely, it relies on the first set of generated monitoring rules mentioned above.

The generated agreement is an XML document following WS-Agreement specification. The agreement for the example application is shown in Listing 5. See that the *SLA Generator* reads the collected metric names `AvarageResponseTimeInternalComponent` and `AppAvailable` assigned in the monitoring rules and uses them as names of their `wsag:Variable`. It also reads from the monitoring rules the action parameters when these metrics are violated (`AvarageResponseTime_Chat_Violation` and `AvarageAppAvailability_Chat_Violation` in the monitoring rules example) to use them as `CustomServiceLevel` in the guaranteed terms of the SLA.

```
<wsag:Agreement xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement">
  <wsag:Name>AAM for SeaClouds Demo V2.0</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">service</sla:Service>
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceProperties wsag:Name="NonFunctional" wsag:ServiceName="default">
        <wsag:VariableSet>
          <wsag:Variable wsag:Name="AvarageResponseTimeInternalComponent" 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="ApplicationResponseTime">
        <wsag:ServiceLevelObjective>
          <wsag:KPITarget>
            <wsag:KPIName>ResponseTime</wsag:KPIName>
            <wsag:CustomServiceLevel>{"constraint": "AvarageResponseTime_Chat_Violation", "qos" :
"AvarageResponseTimeInternalComponent LE 2000"}</wsag:CustomServiceLevel>
          </wsag:KPITarget>
        </wsag:ServiceLevelObjective>
      </wsag:GuaranteeTerm>
      <wsag:GuaranteeTerm wsag:Name="ApplicationAvailability">
        <wsag:ServiceLevelObjective>
          <wsag:KPITarget>
            <wsag:KPIName>AppAvailable</wsag:KPIName>
            <wsag:CustomServiceLevel>{"constraint": "AvarageAppAvailability_Chat_Violation", "qos" :
"AppAvailable GE 0.998"}</wsag:CustomServiceLevel>
          </wsag:KPITarget>
        </wsag:ServiceLevelObjective>
      </wsag:GuaranteeTerm>
    </wsag:All>
  </wsag:Terms>
</wsag:Agreement>
```

**Listing 5: Generated SLA document for the WebChat application**

### 3.5 Deployment on IaaS/PaaS

Once the design phases have been completed and the DAM is generated, this DAM document will contain the necessary information for executing the deployment. Target providers are identified by the different *locations* assigned in the DAM to each module of the application. Thanks to the homogeneity provided by the *Deployer* component, a mixed or multi-cloud deployment can be achieved for the target application, assigning different locations to each of its modules. SeaClouds *Deployer* will process the DAM and execute the deployment for each of the modules.

SeaClouds *Deployer* component provides support for deploying on both IaaS and PaaS cloud providers. Capabilities for IaaS deployment were provided natively by the SeaClouds deployer engine on most providers by means of jClouds<sup>15</sup> used for Brooklyn. We have integrated in the deployer engine the capabilities for PaaS deployments as well. In this sense, currently SeaClouds integrates CloudFoundry<sup>16</sup> as the main PaaS provider, supporting several kinds of application modules and services. An initial support for OpenShift<sup>17</sup> is also provided, as a proof of concept that needs to be extended. Moreover, current version of Seaclouds Deployer is also able to deploy PHP applications, which was a feature not supported previously by Brooklyn.

Following the sample application, a possible outcome of the design and planning process could be a mixed deployment where the `Chat` module is assigned to be deployed on a PaaS provider, like CloudFoundry, whereas the `MessageDatabase` will be targeting an IaaS service, such as Amazon EC2.

Figure 7 shows a screenshot of the *Dashboard*. This window of the *Dashboard* illustrates an overall summary of the deployment process, where the user can check the status of each of the individual tasks. If the report is successful, the user will find the deployed application listed as available for being managed.

---

<sup>15</sup> <https://jclouds.apache.org>

<sup>16</sup> <https://www.cloudfoundry.org>

<sup>17</sup> <https://www.openshift.com>

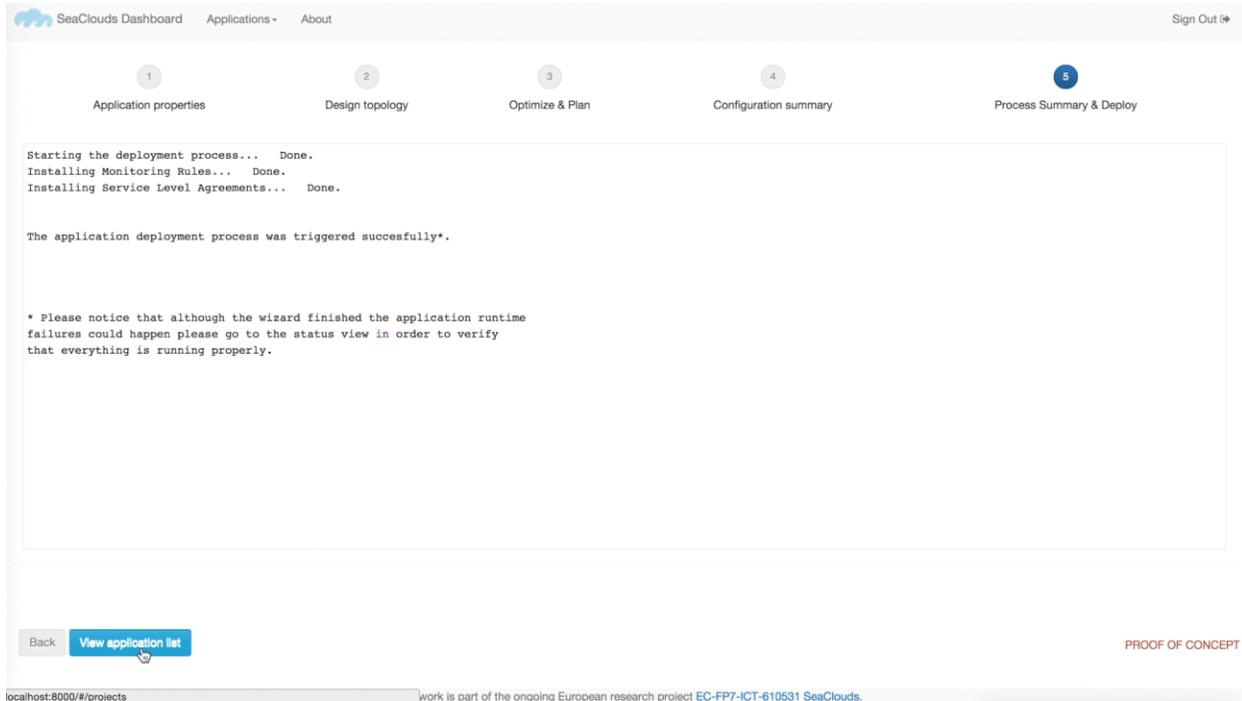


Figure 7: SeaClouds Deployer - Deployment process and summary

During the deployment process, SeaClouds will prepare the application to be monitored and to analyse the QoS and requirements are not violated. Thus, in order to achieve monitoring tasks, the *Deployer* needs to attach specific data collectors, that will be deployed along each of the application modules. These agents will perform the collection of metrics on the target cloud providers.

Brooklyn will take care of the application repairing by means of autoscaling policies based on the modules response time. Brooklyn provides by default this kind of autoscaling policy for the *DynamicControlledWebAppCluster* entity, which also provides a load balancing service. The monitoring system does not take part in the repairing process, but instead it does in the SLA enforcement and in the replanning processes. In particular, as presented in Section 3.5, a monitoring rule for each QoS requirement is generated by the *Monitoring Rules Generator*.

Concerning the replanning a first considered scenario is to replan when an application is “ON FIRE” (terminology used by Brooklyn to indicate a undesirable status of the application and lack of internal actions to repair it). In this case, the *Dashboard* is notified and the *replan()* endpoint exposed by the *Planner* is called. This endpoint receives the current DAM of the deployed application and additional diagnostic information regarding the cloud provider that is failing. This additional information will let the *Planner* avoid choosing this provider again for the next deployment. To this aim a specific data collector named `reconfiguration-data-collector` continuously asks the *Deployer* for the application status and if this is “ON FIRE” the *Dashboard* is notified and the replanning is triggered.

Next section gives more details related to the Monitor and SLA components.

### 3.6 Execution, visualization of monitoring data, autoscaling

In order to visualize the application status, the user of SeaClouds can choose among two monitoring engines:

- Brooklyn built-in monitoring platform: SeaClouds will be able to retrieve information as fallback reading directly from Brooklyn Sensors. This system works based on existing monitoring mechanisms on the hosts, like JMX<sup>18</sup>. Figure 8 presents the graphs that can be obtained through this approach. Specifically, for our example, it shows the case when latency of the application is too high and the application performance is lower than the defined limits. Then, the policy modifies the assigned resource to the application.
- Tower4Clouds + Graphite & Grafana<sup>19</sup>: This approach uses Tower4Clouds data-collectors (from the MODAClouds<sup>20</sup> project) to retrieve monitoring information based on monitoring rules. The data collectors push the information into Tower4Clouds where Graphite registers as an observer and can display data through its interface of pass them to Grafana. For instance, Figure 9 shows this second type of data presentation. We can see a visualization of the “Average RAM Usage” evolution over the last 15 minutes of the virtual machine where the `MessageDatabase` module of the example application is running.

While an application is running and it is being monitored, SeaClouds *Monitor* can detect incidents in its behaviour - for example, any SLA or monitoring rule violations - and respond to this event in a reactive way. Besides, when an unexpected drop in performance is detected in any application module, Brooklyn can react to this event and scale the clouds resources which are bound to the application according to the application requirements.

Thus, in our example, once the `Chat` module of the example application is running in a cloud provider and the data collectors send the application status to the SeaClouds *Monitor*, this calculates the performance of the application and the target cloud provider. If any issue is found SeaClouds *Deployer* will try to rescale the resources assigned to the application using the mechanisms defined by Brooklyn.

---

<sup>18</sup> <http://www.oracle.com/technetwork/articles/java/javamanagement-140525.html>

<sup>19</sup> <http://docs.grafana.org/datasources/graphite/>

<sup>20</sup> <http://www.modaclouds.eu>

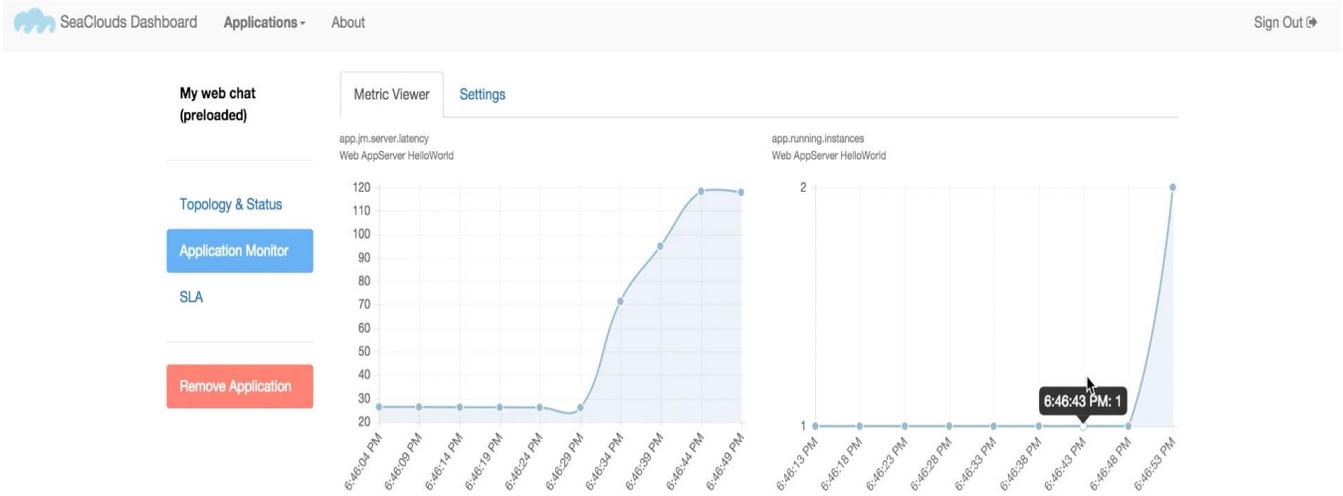


Figure 8: Legacy monitoring view with autoscaling status

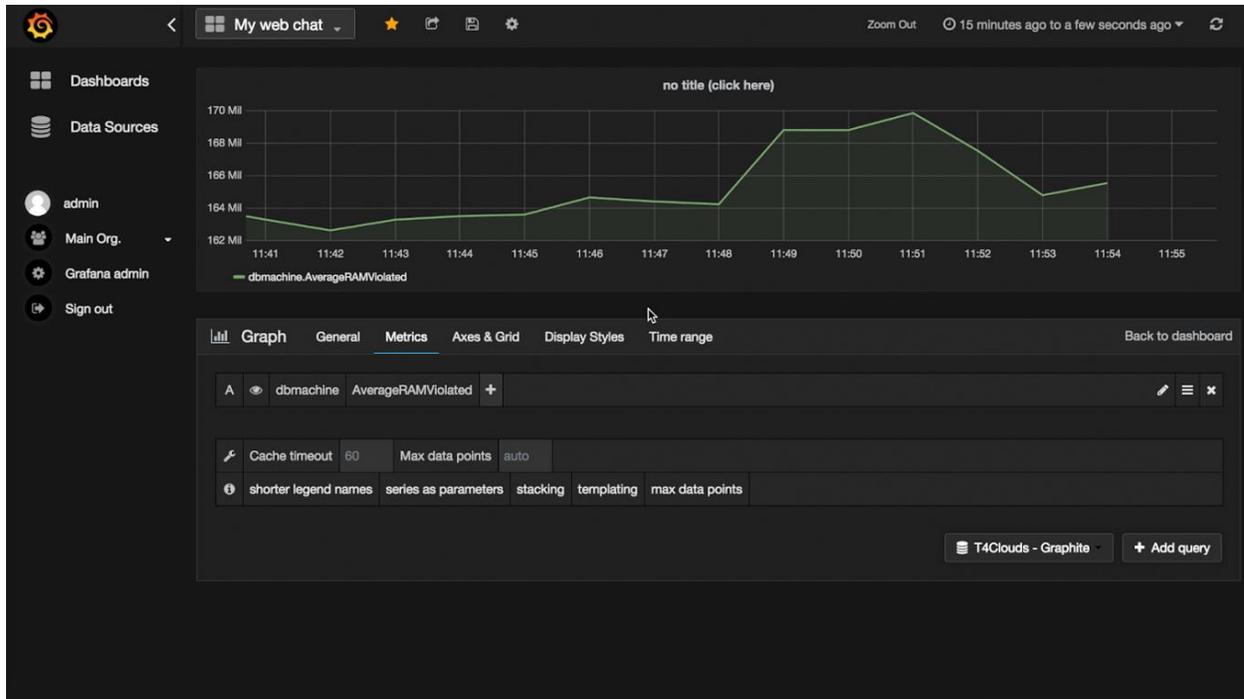


Figure 9: Tower4Clouds + Graphite + Grafana monitoring platform

Finally, Figure 10 depicts the SLA viewer where the user can keep track of the status of the defined SLA's. A complete registry of SLA violations is listed here, allowing the user to stay alert of repeated misbehaviour from the cloud service providers and, eventually, suggesting a replanning process in case of repeated violations.

SeaClouds Dashboard Applications - About Sign Out

My web chat (preloaded) SLA View Add new SLA's

Name: My web chat Service Provider: AgreementResponder Expiration Time: No expiration date

**Guaranteed Terms**

CPUUtilizationLT MemUsedLT

Status: VIOLATED Constrain: AverageRAMViolated NOT\_EXISTS QoS: MemUsed LT 200

**Violations**

Show 10 entries Search:

ID	Date	Value
6d2a2bbf-db32-4f60-9346-8b409503bf1a	2015-06-21T11:32:48CEST	230
9611f843-c50d-4667-baac-2343972acf3b	2015-06-20T23:50:10CEST	0.7

Showing 1 to 2 of 2 entries Previous 1 Next

Figure 10: SLA View showing a violation

## 4 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 D3.2. Discovery, design and orchestration functionalities (SeaClouds Consortium). [http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D3.2%20Discovery\\_design\\_and\\_orchestration\\_functionalities.pdf](http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D3.2%20Discovery_design_and_orchestration_functionalities.pdf) . 2015.
- [2]. SeaClouds Project. Deliverable D2.4 Final SeaClouds Architecture (SeaClouds Consortium), [http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D2.4-Final\\_SeaClouds\\_Architecture.pdf](http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D2.4-Final_SeaClouds_Architecture.pdf) 2015.
- [3]. SeaClouds Project. Deliverable D5.1.2. Integrated Platform (SeaClouds Consortium). <http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D5.1.2-IntegratedPlatform.pdf> , 2015.
- [4]. Apache brooklyn deployment blueprint <https://brooklyn.incubator.apache.org/v/latest/start/blueprints.html><https://brooklyn.incubator.apache.org/v/latest/start/blueprints.html>
- [5]. Seaclouds Project. Deliverable D5.4.2. Second version of s/w platform (SeaClouds Consortium), *To be published.* 2015
- [6]. Seaclouds Project. Deliverable D5.3. Implementation of the user interface (SeaClouds Consortium), *To be published.* 2015
- [7]. Web Services Agreement Specification, <http://www.ogf.org/documents/GFD.192>, Open Grid Forum, 2011.
- [8]. SeaClouds Project. Deliverable D4.4. Dynamic QoS Verification and SLA Management Approach (SeaClouds Consortium), [http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D4.4-Dynamic\\_QoS\\_verification\\_and\\_SLA\\_management\\_approach.pdf](http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D4.4-Dynamic_QoS_verification_and_SLA_management_approach.pdf) , 2015.