SeaClouds Project

D5.3 – Implementation of the User Interface

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	D5.3
Deliverable Title	Implementation of the User Interface
Nature	Prototype
Dissemination Level	Public
Due Date:	M24
Submission Date:	14th October 2015
Version:	1.0
Status	Final
Author(s):	Román Sosa González (Atos), Adrián Nieto (UMA)
Reviewer(s)	Marc Oriol (UPI), Christian Tismer (Nuro)

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	18/09/2015	ToC	Román Sosa
0.2	21/09/2015	Introduction and overview	Román Sosa, Adrián Nieto
0.3	22/09/2015	Installation	Román Sosa, Adrián Nieto
0.4	23/09/2015	Usage screens	Román Sosa, Adrián Nieto
0.5	25/09/2015	Usage section description	Román Sosa, Adrián Nieto
0.6	28/09/2015	Version to internal review	Román Sosa
1.0	14/10/2015	Final version after internal review	Román Sosa, Marc Oriol, Adrián Nieto

Table of Contents

Table of Contents 3
Executive Summary 5
1 Introduction 6
1.1 Glossary of Acronyms 6
2 Overview of the User Interface 7
2.1 Architecture 7
2.2 Technology 8
2.3 File formats 8
2.3.1 Topology 8
2.3.2 Abstract Application Model 11
3 Installation 14
3.1 Standalone installation 15
3.2 Installation with Brooklyn 15
4 Usage of the User Interface 17
4.1 Add new application 17
4.1.1 New application properties 18
4.1.2 Topology designer 19
4.1.3 Optimization result 25
4.1.4 New application summary 25
4.1.1 Application deployment process 27
4.2 Application status 27
4.2.1 Overview 27
4.2.2 SLA 27
4.2.3 Legacy monitoring 29
4.3 Grafana Monitoring View 29
4.4 Remove application 30
5 Conclusions 31
Appendix A. Topology model for Chat application 32
Appendix B. Abstract Application Model for Chat application 33
References 35

List of Figures

FIGURE 1: DASHBOARD ARCHITECTURE 7
FIGURE 2: NEW APPLICATION PROPERTIES 19
FIGURE 3: DESIGN OF THE TOPOLOGY 20
FIGURE 4: COMPONENT DETAILS 21
FIGURE 5: WEB APPLICATION TECHNOLOGICAL REQUIREMENTS 22
FIGURE 6: DATABASE TECHNOLOGICAL REQUIREMENTS 22
FIGURE 7: NON-FUNCTIONAL REQUIREMENTS 23
FIGURE 8: PROVIDER INFRASTRUCTURE 24
FIGURE 9: CONNECTTO RELATIONSHIP PROPERTIES 24
FIGURE 10: OPTIMIZATION RESULT 25
FIGURE 11: NEW APPLICATION SUMMARY 26
FIGURE 12: DEPLOYMENT PROCESS 26
FIGURE 13: OVERVIEW OF APPLICATION STATUS 28
FIGURE 14: SLA STATUS 28
FIGURE 15: LEGACY MONITORING 29
FIGURE 16: GRAFANA MONITORING VIEW 30
FIGURE 17: REMOVE APPLICATION 30

List of Tables

TABLE 1: GLOSSARY OF ACRONYMS 6

List of Listings

LISTING 1: TOPOLOGY MODEL STRUCTURE 9
LISTING 2: ABSTRACT APPLICATION MODEL STRUCTURE 12
LISTING 3: DEFAULT CONFIG CONFIG FOR DASHBOARD 14
LISTING 4: DEFINING THE NEEDED ENVIRONMENT VARS FOR THE DASHBOARD EXECUTION 15
LISTING 5: DASHBOARD BROOKLYN BLUEPRINT 16

Executive Summary

This deliverable is the final User Interface developed within the SeaClouds project. This document aims at accompanying the software by offering information about: i) the interactions between the User Interface and the rest of the components of SeaClouds; ii) the way of install and run the User Interface, and iii) the usage of the User Interface.

1 Introduction

In the context of the SeaClouds project, the SeaClouds User Interface constitutes the uppermost layer in the SeaClouds architecture. The high-level specification of this interface was defined in D2.4 [1], according to the requirements elicitation defined in D2.1 [2].

The User Interface has been implemented following the design principles outlined in D5.2.1 [4] and the design of the prototype shown in D5.2.2 [5].

This document is structured as follows:

- Section 2 of the deliverable is an overview of the SeaClouds components that constitutes the User Interface, the relation with other SeaClouds components, and the data that these components process.
- Section 3 contains the installation and running guide of the User Interface.
- Section 4 explains the usage of the User Interface through an example.
- Appendix A contains examples of the file formats described in section 2.

1.1 Glossary of Acronyms

Acronym	Definition
AAM	Abstract Application Model
API	Application Program Interface
DBMS	DataBase Management System
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
JSON	JavaScript Object Notation
MVC	Model View Controller
PaaS	Platform as a Service
QoB	Quality of Business
QoS	Quality of Service
REST	REpresentational State Transfer
SLA	Service Level Agreement
VM	Virtual Machine
YAML	YAML Ain't Markup Language

Table 1: Glossary of acronyms

2 Overview of the User Interface

SeaClouds Dashboard Component provides a rich and simple user interface among with the necessary services built over a REST API. The Dashboard[1] code is located in the SeaClouds repository.

2.1 Architecture

SeaClouds Dashboard is built on top of DropWizard[2] Java library. It eases the packaging and the development process by the combination of some technologies, like Jetty (as the Web Server), Jersey (to build the REST API) or Jackson (a JSON to Java Object parser).

The user interface is provided by DropWizard to the user as pure HTML+JS code. This code will interact with SeaClouds Platform by calling the REST API endpoints. They also allow developers to extend SeaClouds functionality or accessing to each Module with independence of the other ones.

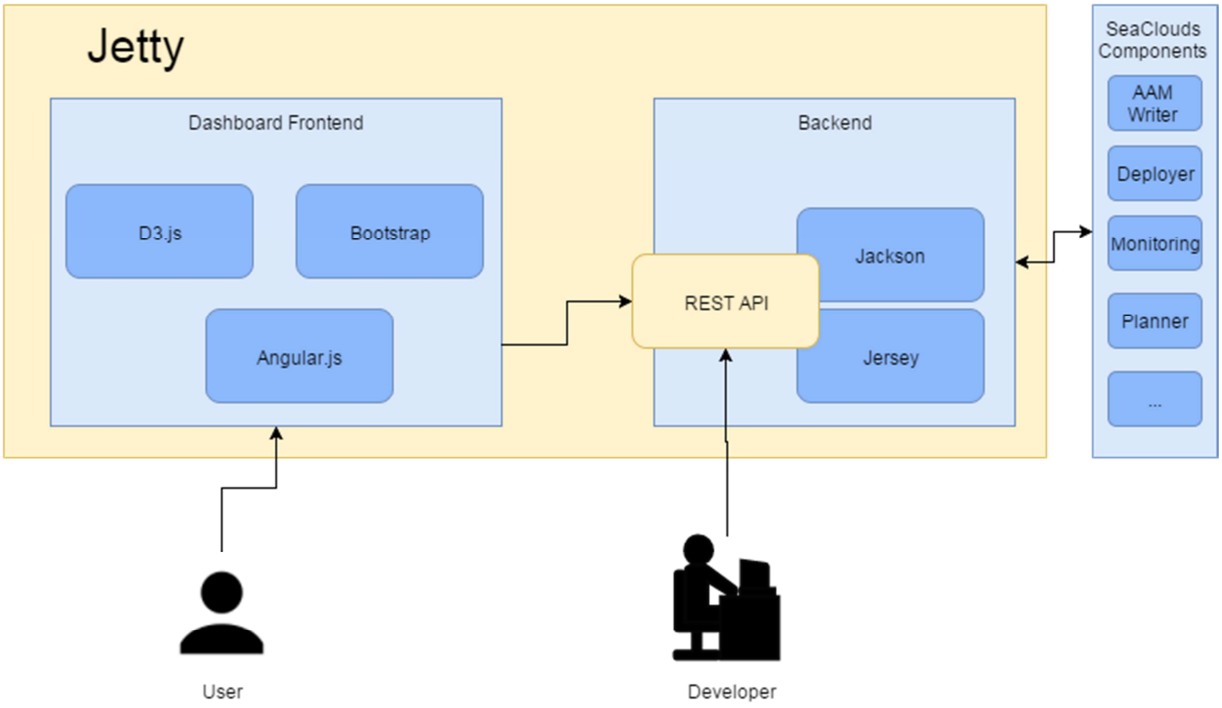

Figure 1: Dashboard architecture

[1] https://github.com/SeaCloudsEU/SeaCloudsPlatform/tree/master/dashboard
[2] http://www.dropwizard.io/

2.2 Technology

The Dashboard is a pure HTML5 + JavaScript application. It uses REST calls to interact with the SeaClouds Platform. The SeaClouds Dashboard is based on the Bootstrap library, which allows adapting the website to the size of the screen, providing a nice user experience with independence of the device (mobile, tablet or traditional desktop). It also uses technologies like AngularJS as a client side MVC to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) and model–view–viewmodel (MVVM) architectures, along with components commonly used in rich Internet applications.

The AngularJS library works by first reading the HTML page, which has embedded into it additional custom tag attributes. Angular interprets those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables. The values of those JavaScript variables can be manually set within the code, or retrieved from static or dynamic JSON resources

Among with these core libraries the topology editor/viewer uses D3.js, a JavaScript library for manipulating documents based on data by providing a framework to work with HTML, SVG, and CSS to display the information in a user friendly format.

2.3 File formats

The main goal of the User Interface is to support users in defining the Topology Model of an application, that is, a description of its components and the way they are connected, and in generating the Abstract Application Model, that is, the TOSCA YAML representation of the Topology Model.

The following of this section provides more details on these two elements.

2.3.1 Topology

The topology model describes the application in a friendly way for the User Interface. Its structure resembles a visually represented graph. It is a JSON file, used only inside the User Interface, which describes: i) the application in terms of its modules, ii) the relationships

between the modules and iii) the user requirements. This model is created during the "New application" wizard, and is used during the status view of the application.

The topology model has the structure shown in Listing 1.

```
{
   "name": "<Application Name>",
   "application_requirements": {
       "response_time": "<Response Time in ms>",
       "availability": "<Availability between 0 and 1>",
       "cost": "<Max desired euros per month>",
       "workload": "<Estimated requests / minute>"
   },
   "nodes": [
       {
           "name": "<Node Name>",
           "type": "<Node Type>",
           "properties": {
               ...
           }
       },
       ...
   ],
   "links": [
       {
           "source": "<Source Node Name>",
           "target": "<Target Node Name>",
           "type": "<ConnectsTo|Host>"
           "properties": {
               ...
           }
       },
   ],
}
```

Listing 1: Topology model structure

A full example of a real application is found in Appendix A.

The fields of the topology model are explained in the following subsections:

Application Requirements

The application requirements are filled in the first step of the "New application" wizard, and describes general requirements related to the application itself. These requirements are used by the Optimizer, and they must be set to obtain a sensible result from the optimization process. If these requirements are left empty, all cloud providers that fulfil the technical requirements may be result of the Optimizer in any order.

The description of the application requirements are:

- *Response Time*. This is the desired maximum response time of the application, measured in milliseconds. The response time is defined as the time needed by the frontend module of the application to respond to a client request, and englobes all the calls between the different modules. The latency times between the client and the application are not included in this time.
- *Availability*. Express the desired availability of the application. The availability describes the time in a month where the service must be available, over the total possible available time, expressed as a percentage. The cloud providers usually state the availability of their services in their SLA, so providers with declared availability lower than the requested availability will not be considered.
- *Cost*. The cost, or budget, describes the maximum estimated cost, measured in euros per month, that the application is willing to spend in cloud provider resources.
- *Workload*. This is the averaged requests per minute that the application is expected to receive.

Such application requirements are used by the Optimizer, which will allocate cloud resources to the application by ensuring that, for the given workload, the application will fulfil the availability and response time requirements, and the cost of the cloud resources needed to obtain the desired QoS is lower than the budget.

The Optimizer also calculates the threshold where the system should scale out. These threshold values are specified in requests per minute: when the workload reaches a threshold, the system should add an instance (a VM in the case of IaaS infrastructures).

A detailed description of the Optimizer is found in Section 5.2.2 of the deliverable D3.2 [6].

Nodes

The nodes are the components that the application is composed of. They include the software artifacts and the services needed by the components itself: databases, message queues…

When the application being designed is sent to the planner, it responds with a set of possible offerings where to deploy each component of the application. When the user selects one of the options, the model is enriched with nodes representing the cloud providers.

There are two main properties in the node:

- *Type*. It represents the kind of the node: a web application, a relation database, a NoSQL database, a cloud provider…
- *Properties*. This field is a dictionary of values that contain the node properties. These properties are dependent on the type of the node, and will be detailed in section 4.1.2 of this document. In the case of a web application, the properties are the language used for the implementation, the application server where the web application is going to be hosted on, the required port where the application will listen… In the case of a cloud provider, the properties are the user credentials to that cloud provider.

Links

A link expresses an existent relationship between a source node and a target node.

There are two main properties in a link:

- *Type*. There are two types of links: i) ConnectTo: a dependency link, where a source node uses a target node, and ii) Host: a host link, where a source node is hosted on a target node.
- *Properties*. This field is a dictionary of values that contain the link properties. These properties are dependent on the type of the link, and will be detailed in section 4.1.2 of this document.

2.3.2 Abstract Application Model

The Abstract Application Model (AAM) describes the topology of the application and its requirements, following the TOSCA specification. A complete definition of its semantics can be found in Deliverable D3.2 [6].

One of the components of the Dashboard is the AAM Generator, which generates the AAM from the topology model. The AAM is needed in the Planner component in order to perform the matchmaking and optimizing processes.

The Abstract Application Model contains most of the information contained in the topology model, but restructured to follow the TOSCA specification. Concretely, its structure is depicted in Listing 2.

```
tosca_definitions_version: tosca_simple_yaml_1_0_0_wd03
description: <Application Name>
imports:
- tosca-normative-types:1.0.0.wd03-SNAPSHOT

topology_template:
 node_templates:
   Module:
     type: sc_req.Module1
     artifacts: {}
     - war: <artifact url>
       type: tosca.artifacts.File
     properties: {}
     requirements:
     - endpoint: <Other Module>
   ...

node_types:
 sc_req.Module:
   derived_from: seaclouds.nodes.<deployable entity>
   properties: {}
 ...

groups:
 operation:
   members:
   - Module
   policies:
   - QoSInfo: {}
   - dependencies: {}
   - QoSRequirements: {}
```

Listing 2: Abstract Application Model structure

The node_templates section contains the properties the deployment layer of the application, i.e., the software components to be deployed, and correspond with the nodes in the topology model. Each node template defines the artifact, technical requirements and dependencies with other node templates.

The node_types section defines the node types of the node templates. Each node template representing an application component has as type a node type defined in this section. These node types are marked with a special prefix (sc_req), intended to be recognized by the Matchmaker as an entity that contains the properties that should be used to filter the cloud offerings. For example, if a database is required to be Mysql 5.5 and to be hosted on a PaaS provider, the associated node type contains the properties mysql_support=true, mysql_version=5.5 and resource_type=platform.

Finally, the groups section defines the policies, which will be used for optimization at design time, and for reconfiguration at runtime. A group is defined by a set of operations, and a operation is defined by a module and a policy that applies to the module.

Each policy is composed of:

- *QoSInfo*: benchmarking information specified by the user when designing the application,
- *Dependencies*: information about the "connects to" relationships between the referred module and other modules, and the number of times that this relationship is called.
- *QoS requirements*: contains the application requirements in case of a frontend module, and specific QoS constraints and business values for a module.

A full example of a real application is found in Appendix B.

3 Installation

The usage of DropWizard Library eases the installation and usage of SeaClouds Dashboard. It simplifies the setup to the installation of Java Virtual Machine (JVM) greater than 1.7 and running the associated jar file.

In order to wire the Dashboard with the other SeaClouds Components, DropWizard uses a YAML file (shown in Listing 3), which is passed as an argument to the dashboard jar file. This YAML file specifies DropWizard specific parameters like HTTP port among with the required parameters to reach every Component endpoint.

```
server:
 applicationConnectors:
 - type: http
   port: 8000
 adminConnectors:
 - type: http
   port: 8001
planner:
 host: ${PLANNER_HOST}
 port: ${PLANNER_PORT}
deployer:
 host: ${DEPLOYER_HOST}
 port: ${DEPLOYER_PORT}
 user: ${DEPLOYER_USER}
 password: ${DEPLOYER_PASSWORD}
monitor:
 host: ${MONITOR_HOST}
 port: ${MONITOR_PORT}
sla:
 host: ${SLA_HOST}
 port: ${SLA_PORT}
```

Listing 3: Default config config for Dashboard

There are two ways to run the Dashboard. These two options require cloning the SeaClouds GitHub repository or downloading a precompiled version. Please read the SeaClouds Usage Guide[3] for further information.

3 https://github.com/SeaCloudsEU/SeaCloudsPlatform/blob/master/usage/installer/README.md

3.1 Standalone installation

The default configuration file for the dashboard delegates on some environment variables the actual location of the other SeaClouds components. As such, the easiest way to manually start the dashboard is defining these environment variables (in uppercase in Listing 4).

The Listing 4 is an example on how to define the environment vars.

```
export PLANNER_HOST=localhost
export PLANNER_PORT=9000
export DEPLOYER_HOST=localhost
export DEPLOYER_PORT=8081
export DEPLOYER_USER=admin
export DEPLOYER_PASSWORD=admin
export MONITOR_HOST=localhost
export MONITOR_PORT=9002
export SLA_HOST=localhost
export SLA_PORT=9003
```

Listing 4: Defining the needed environment vars for the Dashboard execution

Once the variables have been defined, the dashboard is executed with the following command:

java -jar target/dashboard.jar server config.yml

where config.yml is the path of the dashboard configuration file.

3.2 Installation with Brooklyn

The default way to deploy SeaClouds, as explained in deliverable D5.1.3 [7], is using Brooklyn [8] blueprints. The blueprints used to deploy SeaClouds are located in the blueprints folder[4] in the Usage subproject of the SeaClouds platform.

The Listing 5 shows the excerpt of Brooklyn blueprint that deploys the Dashboard component.

```
- serviceType: brooklyn.entity.basic.SameServerEntity
 name: SeaClouds Dashboard


 shell.env:
   SLA_HOST: $brooklyn:component("sla-core").attributeWhenReady("host.address")
   SLA_PORT: $brooklyn:component("sla-core").attributeWhenReady("http.port")

 brooklyn.children:
 - serviceType: eu.seaclouds.dashboard.SeacloudsDashboard
   name: SeaClouds Dashboard
   id: dashboard
   brooklyn.config:
```

4 https://github.com/SeaCloudsEU/SeaCloudsPlatform/blob/master/usage/installer/src/main/assembly/files/blueprints/

```
     port: 8000
     adminPort: 8001
     deployerHost: $brooklyn:component("deployer").attributeWhenReady("host.address")
     deployerPort:
$brooklyn:component("deployer").attributeWhenReady("brooklynnode.webconsole.httpPort")
     deployerUser: $brooklyn:component("deployer").config("brooklynnode.managementUser")
     deployerPassword:
$brooklyn:component("deployer").config("brooklynnode.managementPassword")
     slaHost: $SLA_HOST
     monitorHost: $brooklyn:component("monitoring-manager").attributeWhenReady("host.address")
     monitorPort: $brooklyn:component("monitoring-
manager").attributeWhenReady("modaclouds.mm.port")
   install.latch: $brooklyn:component("sla-core").attributeWhenReady("service.isUp")
```

Listing 5: Dashboard Brooklyn Blueprint

4 Usage of the User Interface

The usage of the User Interface will be illustrated by the utilization of an application example. This application example is described in D5.1.3 [7], and its requirements are included here for convenience.

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 behaviour 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`.

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` must be deployed in a PaaS provider”.

4.1 Add new application

SeaClouds Dashboard provides a user friendly wizard to define and configure all the required parameters to deploy a new application.

4.1.1 New application properties

In the first step of the wizard (see Figure 2) the user set the application name, some properties needed for the optimization properties and the business rules that apply when the QoS constraints are not fulfilled.

The optimization properties are not mandatory, but they should be filled if the user wants SeaClouds to optimize the distribution of the different components of the application into the suitable cloud offerings. These properties are:

- *Response Time:* desired maximum response time of the application in milliseconds.
- *Availability*: desired availability over a month of the application.
- *Cost:* maximum estimated budget, in euros per month.
- *Workload*: average requests per second.

The Section 2.3.1 contains more details about these properties.

If the desired response time and availability are filled, they will be used to generate an SLA agreement that the developer, as an Application Provider, is willing to offer. The business rules, when filled, are used in the business values of the agreement, usually penalties that the Application Provider will incur if the QoS is not fulfilled. Refer to deliverable D4.4 [7] about the approach SeaClouds projects takes about the management of SLAs.

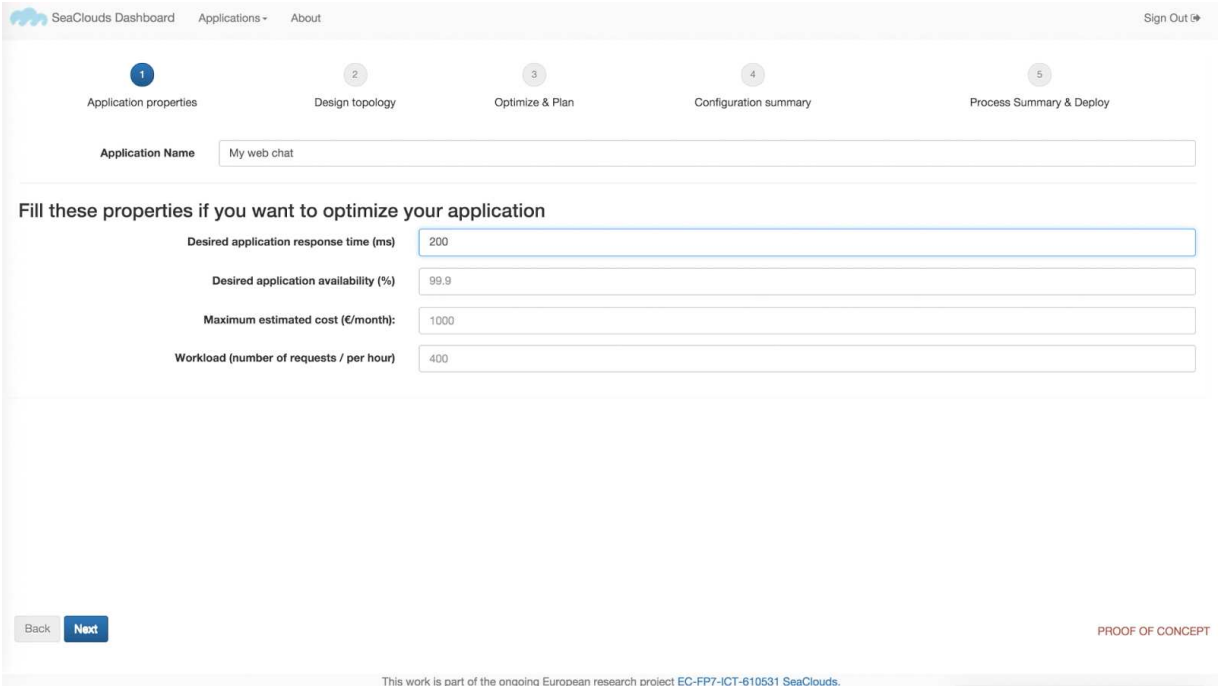

Figure 2: New application properties

4.1.2 Topology designer

In this step, the user describes the components of the application, the use dependencies of the components and the requirements of each component. The user must depict a graph of the application (see Figure 3), where:

- Each component is represented by a graph node. To add a node, the user must select the appropriate kind of component in the palette.
- A relationship between two modules A and B is represented by an arc from A to B, meaning “A depends on B”. To create a relationship, the user must select the “Link” tool or press the Shift key and drag from node A to node B.

The modules are configured through an individual form, like shown in Figure 4. This configuration includes technological requirements (programming language, version…) and non-functional requirements, including information about the cost, location and QoS constraints.

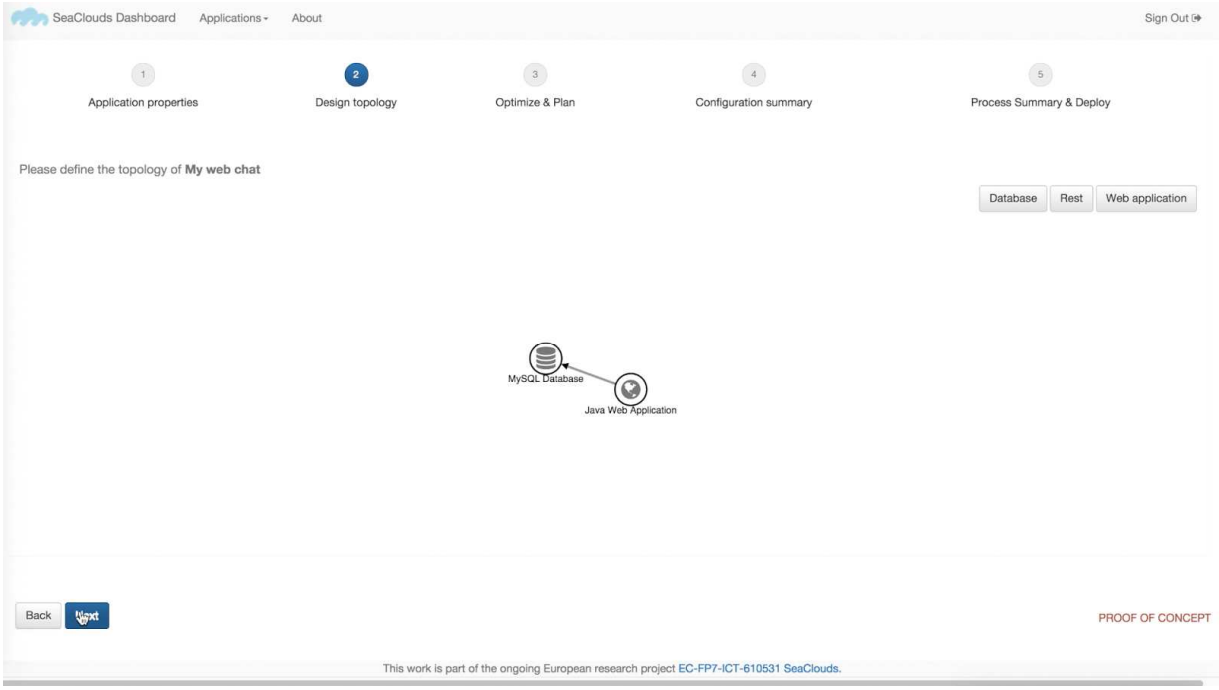

Figure 3: Design of the topology

Once all this information is filled in, the user can keep adding more components in the same way.

In Figure 3, there is designed a simple application with two components, a web application and a database, with a relationship where the web application uses (connects to) the database.

In the following subsections, the forms that handle the currently supported types are described:

Each form type is composed of several field sets. For example, there is a Common fieldset, which collects the name of the component.

Usually, each form type needs to collects information about technical requirements, non-functional requirements and information about the desired cloud provider.

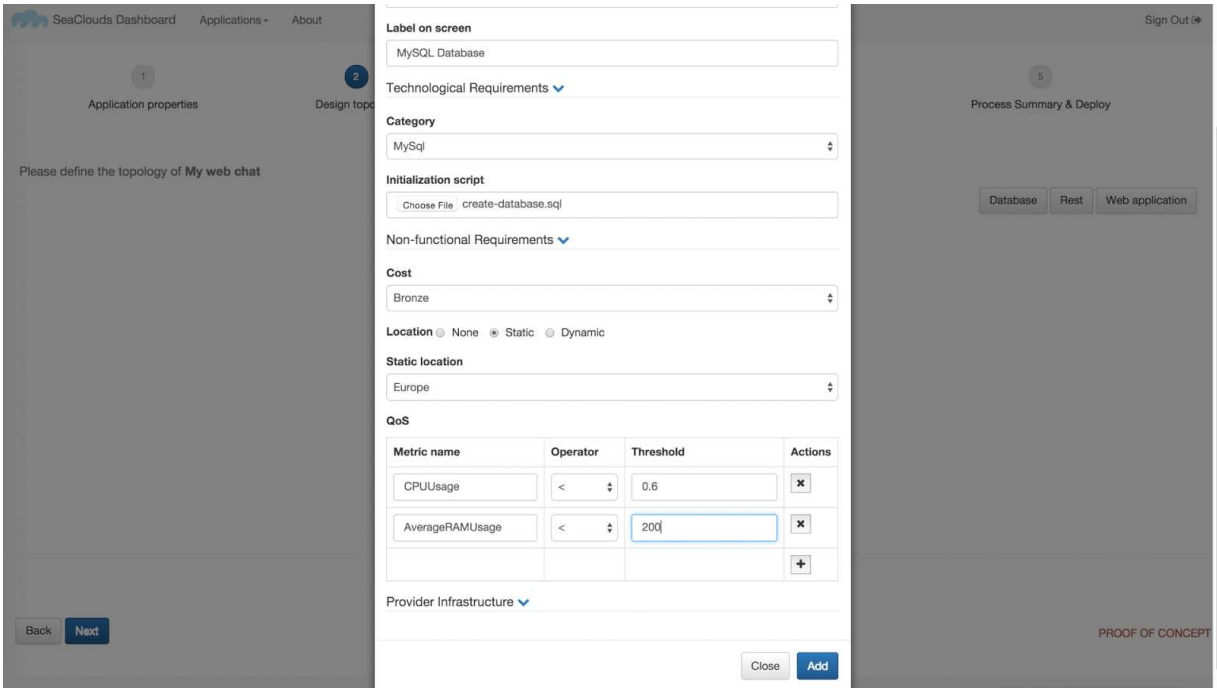

Figure 4: Component details

Technological requirements

This fieldset collects about the used technology for the component. For example, a database would need to collect the DBMS and the initialization script; a web application would need to collect the programming language. The fields used to collect the technological requirements for web application and databases are shown in figures Figure 5 and Figure 6 respectively.

Non-functional requirements

This fieldset (see Figure 7) collects the information about the QoS of the component. The benchmark fields are used for the optimization process:

- Benchmarked response time: measured response time, in average, of the component.
- Benchmark platform: the platform where the response time was measured. This field contain cloud provider platform. The user must select the one that more approximates the actual benchmark platform.

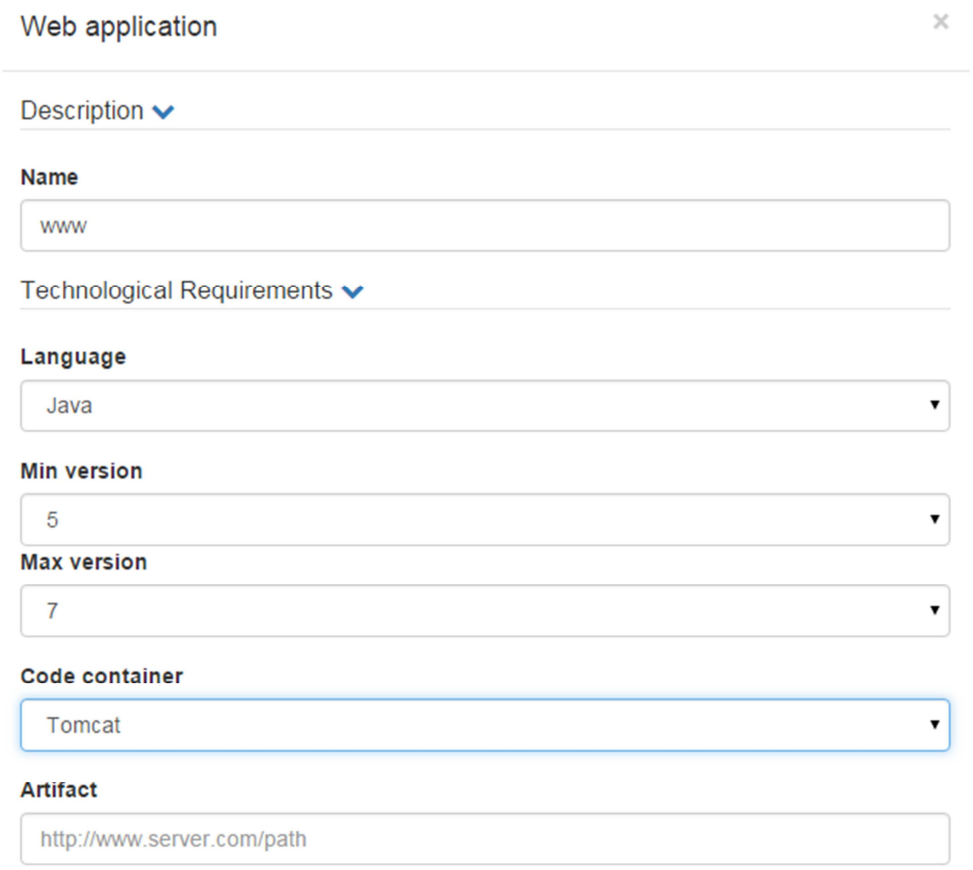

Figure 5: Web application technological requirements

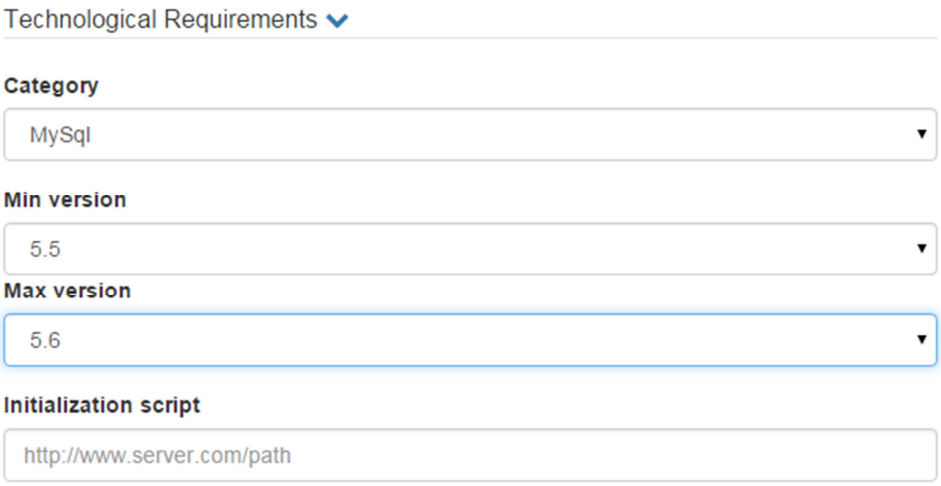

Figure 6: Database technological requirements

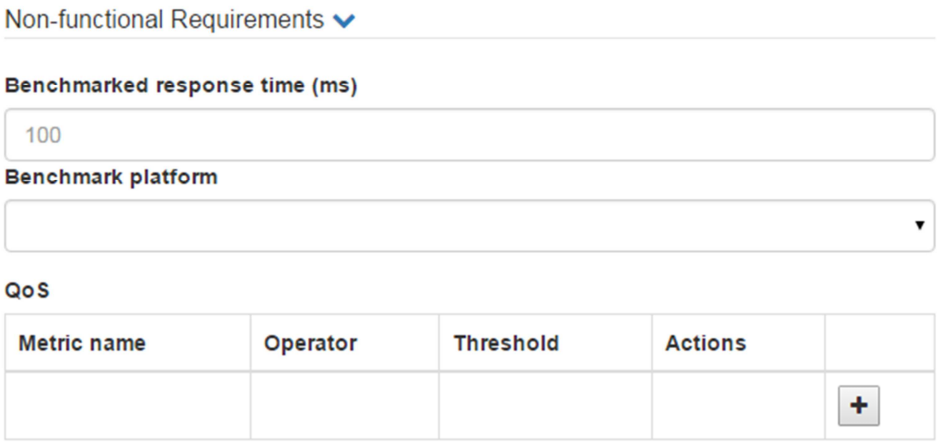

Figure 7: Non-functional requirements

Provider

This fieldset (see Figure 8) collects the information about the desired provider where to deploy the component. The user can select between deploying in PaaS or in IaaS. This information will be used in the matchmaking process. The IaaS selection can be narrowed specifying the number of CPUs and the needed disk size.

The other collected field is the location, which, if selected, may be:

- *Static*. This means that the component will be deployed in one zone of the world: Europe, America or Asia.
- *Dynamic*. This means that the location may change according to a desired policy: follow the sun, follow the moon...

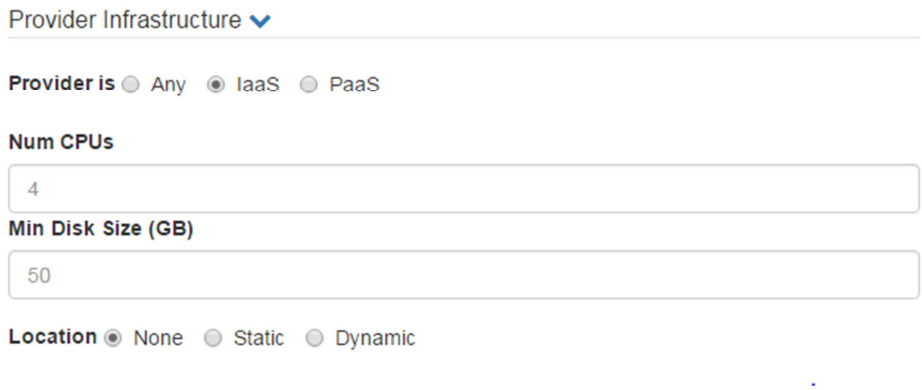

Figure 8: Provider infrastructure

Relationship

The relationship form is not related to nodes, but links between components. The current form, shown in Figure 9, only asks for the average number of calls from the source component to the target component; this is a value needed for the optimization process.

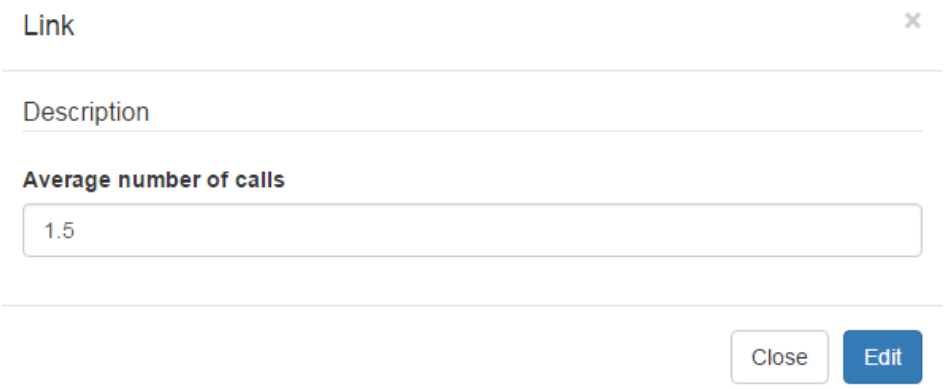

Figure 9: ConnectTo relationship properties

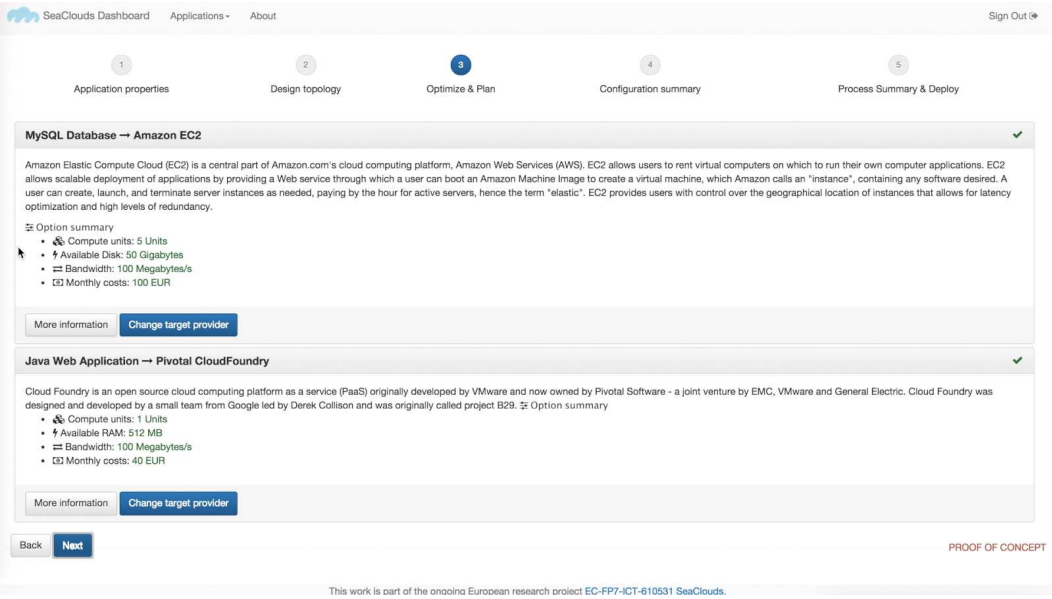

Figure 10: Optimization result

4.1.3 Optimization result

Once the user has entered the topology and requirements of the application, the next step proceeds to calculate a distribution of modules into cloud providers. This distribution is the outcome of the planner and optimizer (see Figure 10). A default distribution, considered optimal, is shown to the user. The user may select this or show up to four more suitable distributions, selecting one of all these.

4.1.4 New application summary

The step 4 (see Figure 11) is the last step of configuration, where the user can review the deployment just before the process starts: the selected providers, cloud resources, estimated costs and other key properties of the application. In this step, the user must enter the cloud provider credentials and any other information needed to generate the deployment descriptor.

The “Deploy” button starts the deployment of the application, and takes us to the “Process” view.

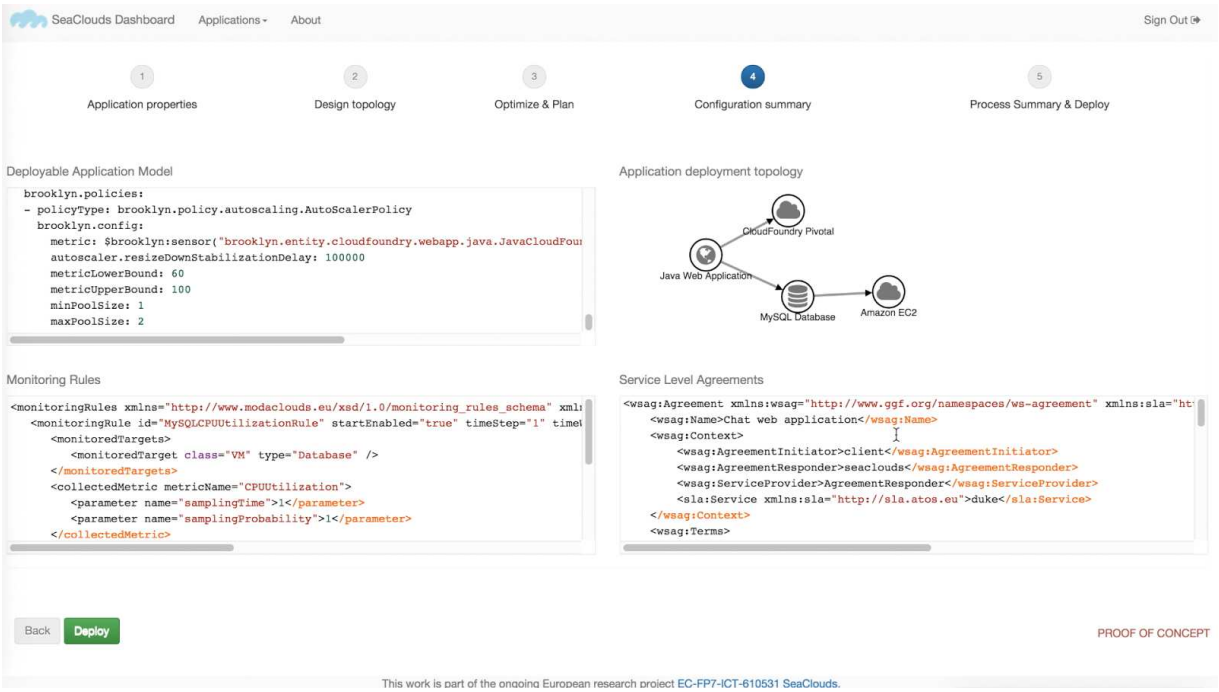

Figure 11: New application summary

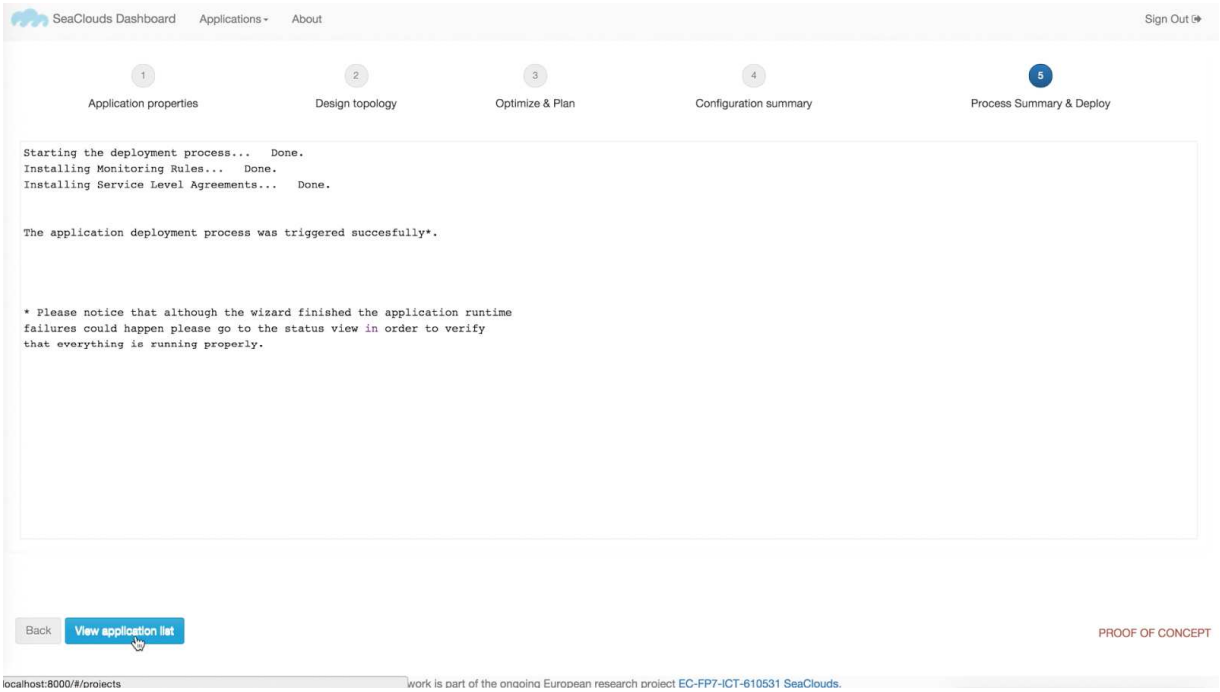

Figure 12: Deployment process

4.1.1 Application deployment process

The Step 5 (see Figure 12) only shows the deployment process with a button to go directly to "Status" view.

4.2 Application status

This view allows the user to check all the relevant information related with each application.

4.2.1 Overview

The Application Overview is designed to show the user the overall status of the application at a glance. It contains a graph with the status of each Application Module.

It uses a modified version of the Topology Designer to display coloured circles around each Application Module according to its status. An example of the Application Overview is shown in Figure 13.

4.2.2 SLA

The SLA view (see Figure 14) details the current SLA accomplishment, allowing the administrator to check if the application is fulfilling the QoS and QoB requirements specified at design time. It maintains a list of the occurred QoS violations and the list of penalties as consequence of the aforementioned violations.

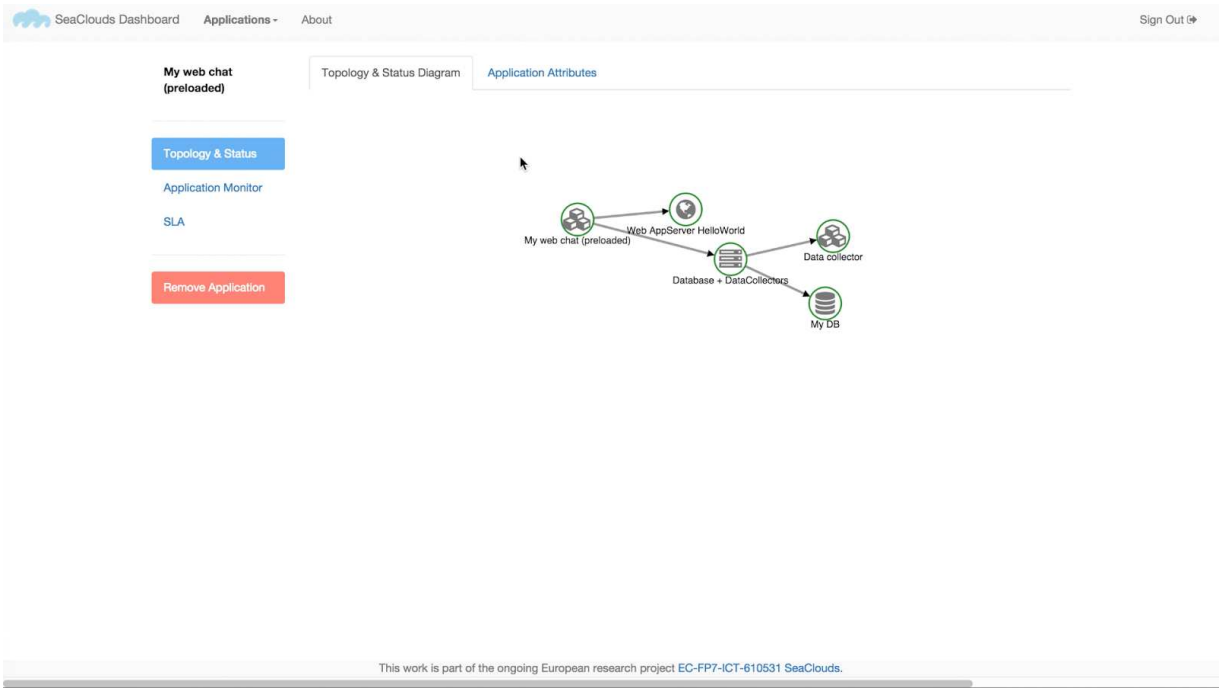

Figure 13: Overview of application status

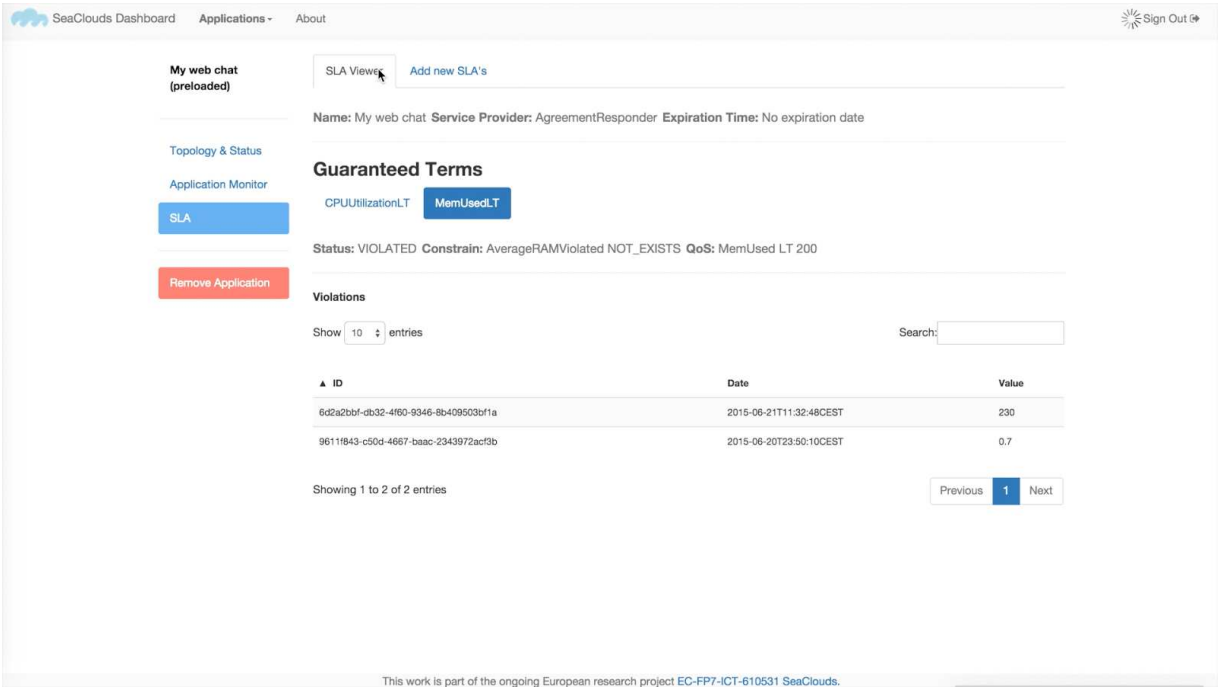

Figure 14: SLA status

4.2.3 Legacy monitoring

Brooklyn provides a basic framework to monitor application metrics. SeaClouds uses this information to provide an alternative to Tower4Clouds as legacy mode. This view, shown in Figure 15, is configurable by the user to define what metrics are more interesting to him.

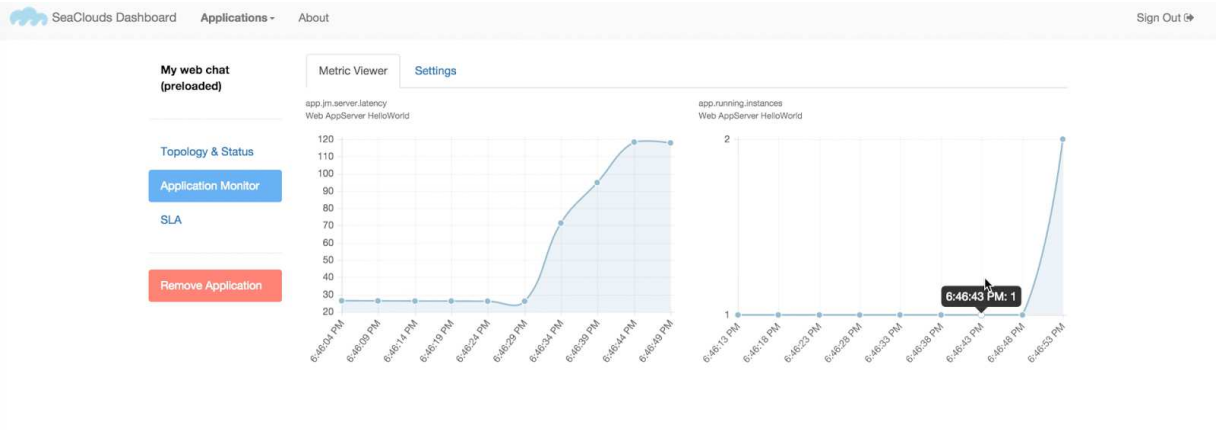

Figure 15: Legacy monitoring

4.3 Grafana Monitoring View

SeaClouds provides an additional monitoring view to the legacy view. It uses Tower4Clouds[5] to define monitoring rules among with data-collectors which retrieves the information directly from the hosts. The data-collectors push the information into Tower4Clouds where Graphite & Grafana register is registered as an observer, in order to let the user to monitor this information from an external application. This view is shown in Figure 16.

[5] http://deib-polimi.github.io/tower4clouds/

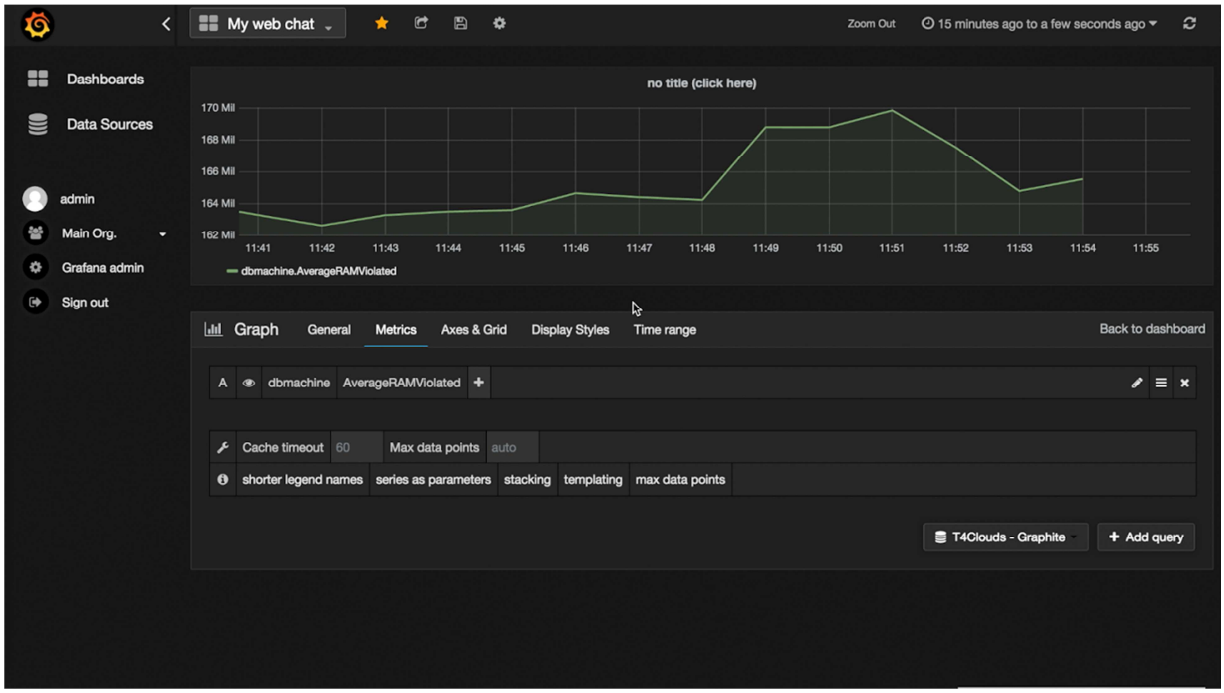

Figure 16: Grafana monitoring view

4.4 Remove application

This section allows a user to remove the selected application. It consists in a simple dialog (see Figure 17) to confirm the operation.

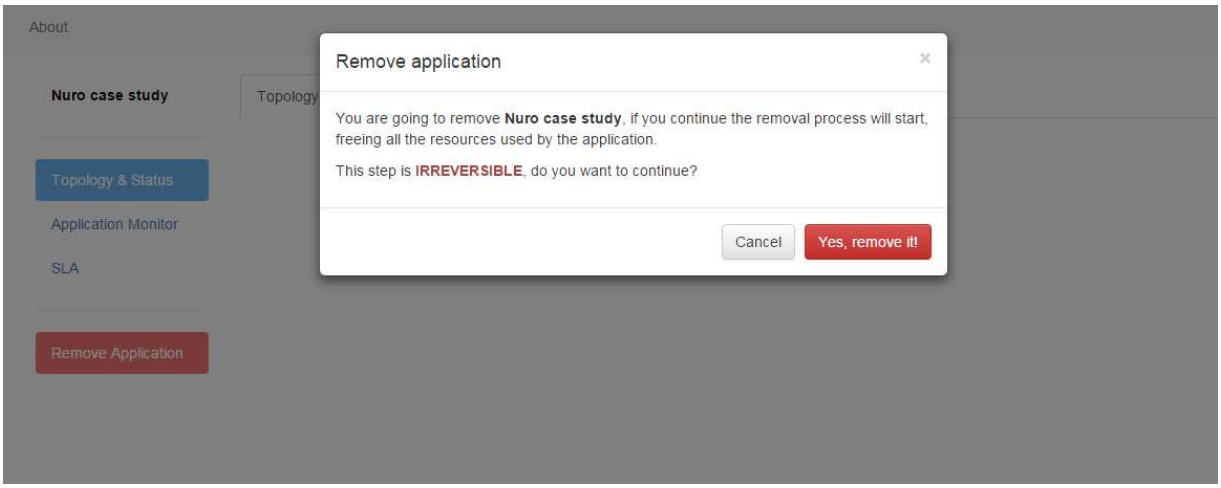

Figure 17: Remove application

5 Conclusions

This document describes the work done in the context of the Task 5.3 “Implementation of the User Interface”, ranging from M14 to M22. It is part of the series of documents to be delivered in M24 in the context of WP5, along with D5.1.3 “Final integrated platform” [7].

The deliverable describes the User Interface in terms of architecture, technology, and format or interchanged data with other components. It also contains a usage guide of the current implementation.

The SeaClouds User Interface 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. The code of the Dashboard[6] component will be always available at the GitHub repository of the SeaClouds platform[7].

[6] https://github.com/SeaCloudsEU/SeaCloudsPlatform/tree/master/dashboard
[7] https://github.com/SeaCloudsEU/SeaCloudsPlatform

Appendix A. Topology model for Chat application

```
{
   "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.2xl"
           }
       },
       {
           "name": "MessageDatabase",
           "type": "Database",
           "properties": {
               "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.2xl"
           }
       }
   ],
   "links": [
       {
           "source": "Chat",
           "target": "MessageDatabase",
           "properties": {
               "calls": "2"
           }
       },
   ],
   "application_requirements" {
       "response_time": "2000",
       "availability": "0.998",
       "cost": "200",
       "workload": "50"
   }
}
```

Appendix B. Abstract Application Model for Chat application

```
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.2xl
   - 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.2xl
  - dependencies: {}
```

References

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

[2]. SeaClouds Project. Deliverable D2.1 Requirements for the SeaClouds Platform (SeaClouds Consortium), February 2015.

[3]. SeaClouds Project. Deliverable D4.5 Unified dashboard and revision of cloud API (SeaClouds Consortium), February 2015.

[4]. SeaClouds Project. Deliverable D5.2.1 Design of the User Interface (SeaClouds Consortium), October 2014.

[5]. SeaClouds Project. Deliverable D5.2.2 Final design of the User Interface (SeaClouds Consortium), March 2015.

[6]. SeaClouds Project. Deliverable D3.2 Discovery, design and orchestration functionalities (SeaClouds Consortium), April 2015.

[7]. SeaClouds Project. Deliverable D5.1.3 Final integrated platform (SeaClouds Consortium), September 2015.

[8]. Apache Brooklyn, https://brooklyn.incubator.apache.org 2015

[9]. SeaClouds Project. Deliverable D4.4 Dynamic QoS Verification and SLA Management Approach (SeaClouds Consortium), March 2015.