SeaClouds Project

D4.6 Prototype and detailed documentation of the SeaClouds run-time evironment components

Project Acronym	SeaClouds
Project Title	Seamless adaptive multi-cloud management of service-based applications
Call identifier	FP7-ICT-2012-10
Grant agreement no.	Collaborative Project
Start Date	1st October 2013
Ending Date	31st March 2016
Work Package	WP4, WP SeaClouds run-time environment
Deliverable Code	D4.6
Deliverable title	Prototype and detailed documentation of the SeaClouds run-time evironment components
Nature	Report
Dissemination Level	Public
Due Date:	M22
Submission Date:	July 31st, 2015 (planned) - Sept 10th, 2015 (submitted)
Version:	1.0
Status	Final
Author(s):	Javier Cubo (UMA), Miguel Barrientos (UMA), Marc Oriol (UPI), Michele Guerriero (POLIMI)
Reviewer(s)	Elisabetta Di Nitto (POLIMI)

Dissemination Level

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

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	08/07/15	First ToC	Javier Cubo (UMA)
0.2	13/07/15	First contributions, including the Design of Deployer, Monitor, SLA service, and Dashboard	Javier Cubo (UMA)
0.3	20/07/15	Second contributions, including Architecture & Design of Deployer, Monitor, SLA, synchronization with D3.3	Javier Cubo (UMA), Marc Oriol (UPI)
0.4	27/07/15	Modifications as regards the API sections	Javier Cubo (UMA), Miguel Barrientos (UMA)
0.5	15/08/15	Final contributions for internal review	Javier Cubo (UMA)
0.6	20/08/15	Revision	Elisabetta Di Nitto (POLIMI)
0.7	24/08/15	Modifications	Michele Guerriero (POLIMI)
0.8	03/08/15	Modifications	Michele Guerriero (POLIMI), Miguel Barrientos (UMA), Javier Cubo (UMA)
1.0	08/09/15	Final version	Javier Cubo (UMA)

Table of Contents

List of Figures

List of Tables

Executive Summary

This document presents a prototype and the documentation for the different components of SeaClouds related to the run-time part. In detail, it presents: (i) a description of the SeaClouds unified management dashboard and a subset of the API related to the run-time environment components (in Deliverable D4.5 [1] was introduced the SeaClouds API), (ii) an architecture & design description of the Deployer, and prototype multi-deployment techniques; (iii) an architecture & design description of the component to trace cloud applications, extracting monitoring data, the Monitor, and the corresponding prototype; (iv) a description of the architecture and design of the SLA, and the SLA prototype; and finally (v) an explanation of how to get and install the SeaClouds components and the SeaClouds Integrated Platform.

The code of such implementation is released under Apache 2.0 license and can be downloaded from the SeaClouds Platform github repository https://github.com/SeaCloudsEU.

1. Introduction

This document describes the prototype of the SeaClouds components used at run-time. It includes the architectural definition, the relationship between components, and their technical details. Deliverable D3.3 [2] will present the corresponding implementation for the design-time components.

The SeaClouds project is an open source software released under Apache 2.0 license and the released prototype can be downloaded from the SeaClouds Platform github repository https://github.com/SeaCloudsEU.

1.1. Structure of this document

The structure of this document is as follows. Section 2 presents an architectural overview of the SeaClouds platform checking the different components and functionalities. Section 3 presents how the SeaClouds platform is organized, with the connections of the Dashboard with the rest of components, as well as the structure of the API (everything following the ideas already exposed in D4.5, but with some modifications). Sections 4, 5 and 6, respectively presents the individual architecture and design as well as the implementation (service layer API). Finally, we close the document with the final conclusions.

1.2. Glossary of Acronyms

Here we list the different acronyms which 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
APP	Application
AAM	Abstract Application Model
DAM	Deployable Application Model
ADP	Abstract Deployment Plan
YAML	YAML Ain't Markup Language
REST	Representation state transfer

Table 1. Acronyms.

2. Architectural overview

This section describes the architectural overview of the SeaClouds platform. In Figure 1 we illustrate the current version of the architecture which was refined from

Deliverable [3] considering the implementation and design decisions taken during the development of the platform.

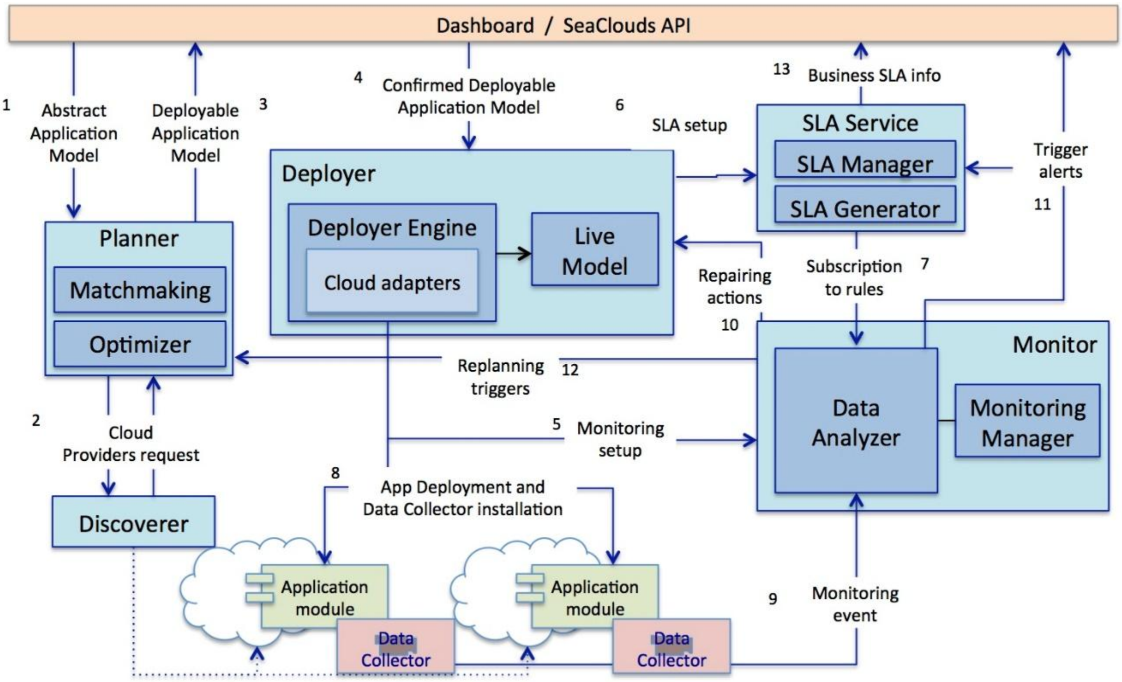


Figure 1. SeaClouds Architecture

The SeaClouds Architecture is well-explained in Deliverable D2.4 [4]. In Deliverable D3.3 [2] the implementation of the design-time part (Discoverer and Planner) will be more detailed. As regards the runtime part, in next sections of this document, we describe the architecture focusing on the Deployer, Monitor and SLA components for runtime execution. Nevertheless, in order to understand how all the components communicate among them, firstly, we present the organization of the components having as reference to the Dashboard and API, in the next section.

3. Dashboard and SeaClouds API

The main goal of the SeaClouds Dashboard is to provide a simple Graphical User Interface (GUI) to the application administrator, unifying the usage of every functionality provided by each of the SeaClouds components. SeaClouds Dashboard gives access to the SeaClouds components by composing the different services offered into a single SeaClouds API.

The code of the Deployer is available at:
https://github.com/SeaCloudsEU/SeaCloudsPlatform/tree/master/dashboard

From a technical point of view, SeaClouds Dashboard is a web application composed of two tiers: frontend and backend (Figure 2). Frontend layer is a pure HTML5 + JavaScript application. It uses REST calls to interact with the backend layer. The

SeaClouds Dashboard is based on Bootstrap [5] to provide a responsive and adaptive design, AngularJS [6] as a client side MVC to provide all the functionality. This enables the adaptation of 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

The backend layer of the dashboard is a Java application that composes a service layer, offering and forwarding REST services from the different SeaClouds components to the Dashboard frontend application. Dashboard Backend also performs authentication tasks and maintenance of user properties, like information about the deployed applications and credentials to access the different providers.

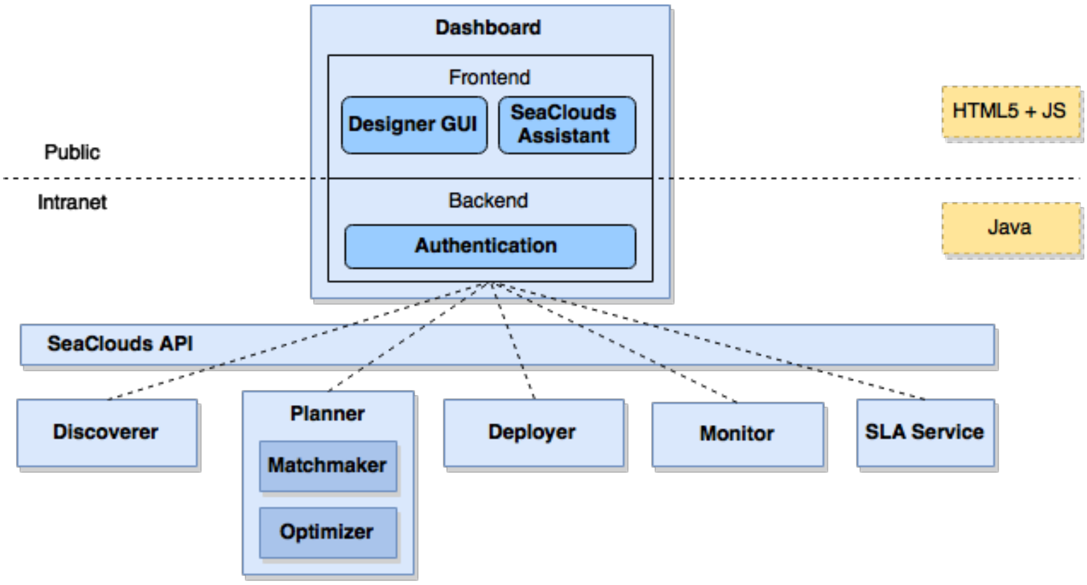


Figure 2. Dashboard interaction diagram

In Deliverable D4.5 [1], we already presented the Dashboard user interface which will be employed to use the services that are provided by SeaClouds. We described the available functionalities, like create and configure an application and its post-management, by specifying the interaction among the SeaClouds components for carrying out the different supported operations.

The SeaClouds functionalities is currently being exploited as a whole, following a wizard style, guiding the user through all the components. Also, SeaClouds is exposing the components with the possibility to exploit them individually (considering also the exploitation of design-time and runtime toolkits, or even intermediate brokers - several components used for several platforms at the same time).

We are currently working on analyzing the dependencies among components, so that single functionalities can be offered separately to the user (Discover component, Deployer component, etc). In principle, we are mainly focused on the design and user experience on the option "SeaClouds as a whole", in order to avoid incoherences between the wizards and the usage of individual components. Therefore, we have implemented an advanced mode which allows the user to interact with the modules independently (for example, deploying an existing deployment plan specified in TOSCA YAML).

4. Deployer

The main goal of the Deployer component is to deploy the application in a multi-cloud environment. To do this, it reads a Deployable Application Model (DAM) which 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, relationships, dependencies, features, constraints, target providers, etc. Therefore, the DAM includes an embedded deployment plan for carrying out the deployment based on the modules dependencies and relations.

4.1. Architecture & Design

Figure 3 shows the architecture of the Deployer component. The deployer is composed of several elements (more details in Deliverable D4.1 [7]). The main element is the Deployer Engine, which receives a Deployable Application Model (DAM) through its Deployer API and executes the DAM. 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).

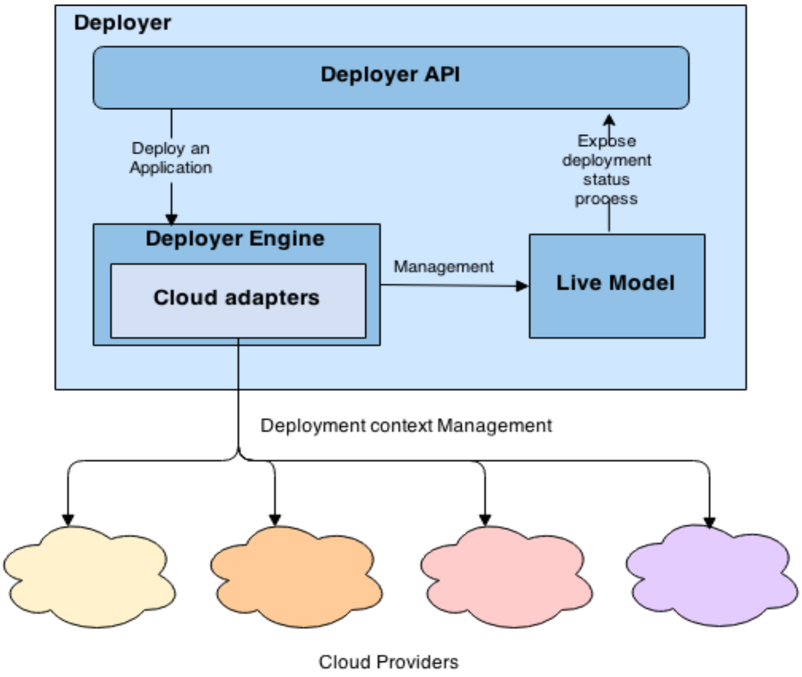


Figure 3. Architecture of the Deployer component

Once the application has been deployed, the Deployer Engine maintains the Live Model, which contains (i) the data structure (components and relationship between these) in order to maintain the topology of the application, with a profile about the topology of the real application distribution over the target providers (using the cloud adaptors), (ii) the relationships and dependencies, (iii) the used services, and the deployment context (IP of the used machines, used O.S., real listener port, metadata of the applications components, etc).

Currently, SeaClouds is using Brooklyn [8] as Deployer Engine to accomplish the multi-cloud deployment of the application components and the Live Model generation and management.

The application components could be deployed over different cloud providers simultaneously, using jClouds [9] as Cloud Adapter at IaaS level, and currently the SeaClouds consortium is developing a set of PaaS connectors to deploy against PaaS providers, including mainly Cloud Foundry, OpenShift and Heroku (trying to generate a unified layer at the PaaS level, similar to jClouds at the IaaS level). Thus, we define the DAM based on the YAML Blueprint specification of Brooklyn [10].

The live model is used by the Monitor and Dashboard, to maintain the status of the application according to the constraints and the features which have been described by the user. In addition, the live model is checked by the Dashboard in order to maintain the graphical representation of the application distribution.

Therefore, the Deployer provides a way to manage the application related resources, like the Live Application Model which contains the services used by an application and their configuration, the location for each of the application modules, and the relationships among modules.

The live model consists of many of the same pieces as in the deployable model, with some important additions, such as sensors values, policies, additional entities (if creation of some entities results in children), or effectors (operations) available.

All aspects of the live model are exposed through a REST API where entities can be navigated, and all current information about children, sensor values, and policies can be accessed. By assigning UUID’s as part of the deployable model, these ID’s can be tracked in the live model and live information about the components requested to be deployed can be accessed as needed, by the planner or by an operator.

Once the application have been deployed, its management is accomplished by the Deployer and the Monitor is checking the status of possible violations, connected to the SLA Service (see Deliverable 4.4 [11] where the Dynamic QoS verification and SLA management approach is detailed).

Whether a violation occur, then a reconfiguration process is required. This process is detailed in Deliverable 4.3 [12].

Figure 4 shows the sequence diagram for the deployment process, while Figures 5 and 6 show the reconfiguration process after a violation occurs.

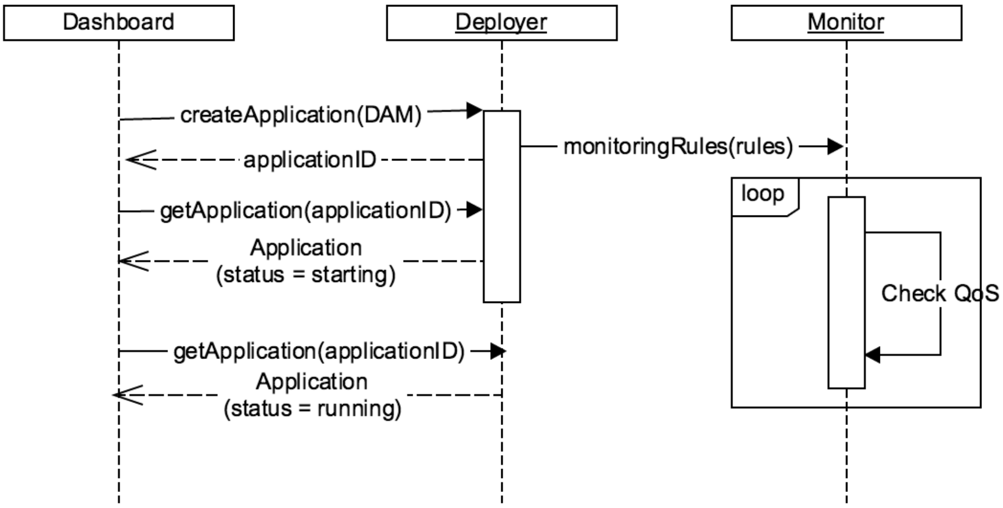


Figure 4. Deployment process

As seen in Figure 4, deployment process starts with the input of the DAM (*createApplication*) to the Deployer. Once the deployer has the deployment information, it forwards the monitoring rules to the Monitor.

This will setup the connections to the Data Collectors needed by the Monitor so that it can start checking QoS properties. While the application is being deployed, any request of information about the application (*getApplication*) will return a "STARTING" status, indicating that it hasn't been deployed completely. Once the deployment has finished, the application status will change to "RUNNING".

The repairing process (Figure 5) requires an existing deployed application already running in SeaClouds. During the deployment phase, monitoring rules have been installed and configured on the Monitor.

This will allow the Deployer to be aware of QoS violations that can trigger one of the repairing processed supported, mainly based on policies. Dashboard can retrieve the status of the application to keep track of the repairing process.

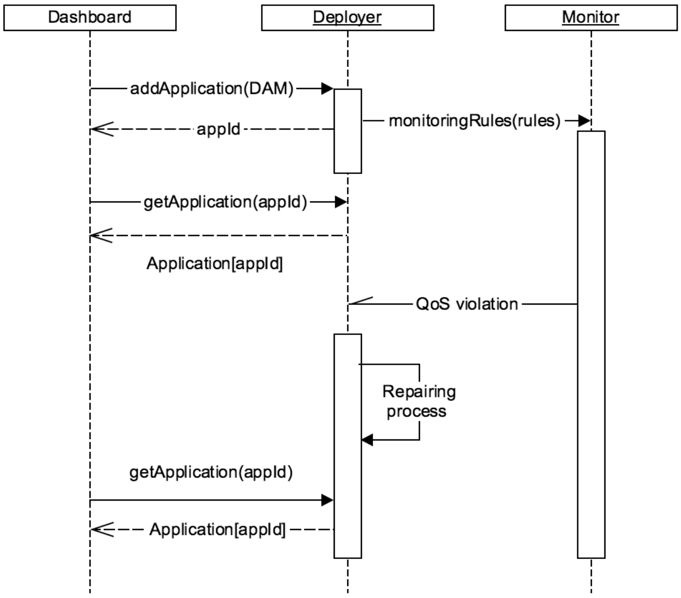


Figure 5. Sequence diagram for Repairing process

Replanning process (Figure Z) follows a more complex interaction between components. Once repairing actions are unable to fix existing QoS violations an alert is triggered from the Deployer and it will show the need of performing a replanning. Once a replanning scenario has been recognised, a confirmation request will be send to the user through the Dashboard, triggering a replanning request on the Planner (*replan*). The new actions generated by the Planner will be forwarded to the Deployer to perform the needed modifications.

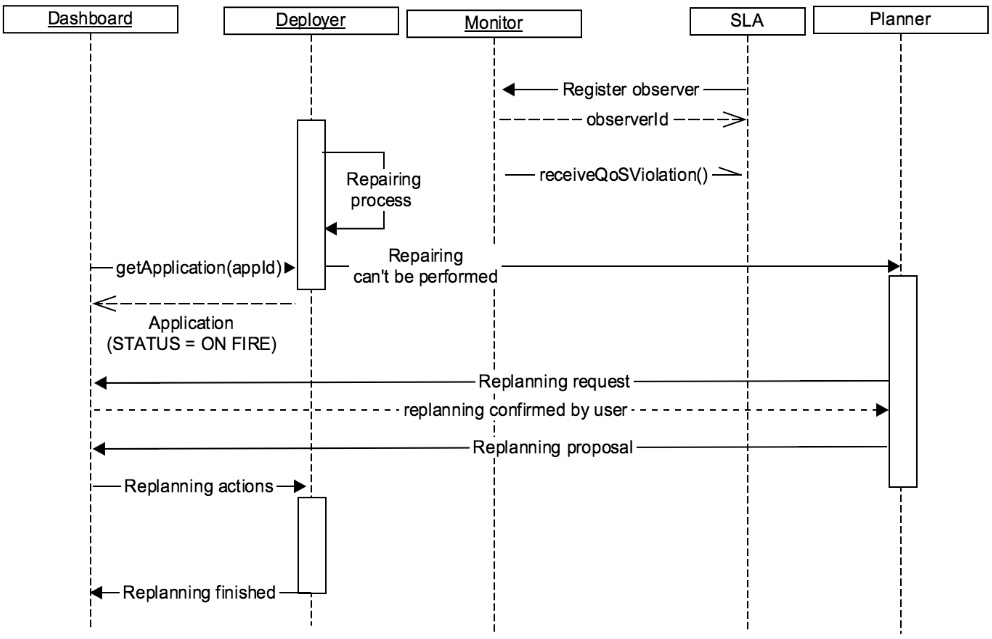


Figure 6. Sequence diagram for replanning process

4.2. Implementation & Deployer API

The Deployer API provides the resources to deploy and manage application modules. It contains the methods for managing the application deployments. The code of the Deployer is available at:
https://github.com/SeaCloudsEU/SeaCloudsPlatform/tree/master/deployer

4.2.1. Terminology

- **Application**: represents an application which was deployed by the Deployer.
  - **ID**: unique identifier
  - **Name**: given name for the application
  - **Status**: current lifecycle status of the application (*"Starting", "Running", "On fire", "Stopped",* etc.).
  - **ConfigParameters**: list of configured application parameters
  - **Policies**: list of attached policies
  - **Modules**: modules that compose the application
  - **Effectors**: operations that can be invoked on application
- **Module**: represents each of the modules that compose an application.
  - **ID**: unique identifier
  - **Name**: given name for the module
  - **Status**: current lifecycle status of the entity (*"Starting", "Running", "On fire", "Stopped",* etc.).
  - **ConfigParameters**: list of parameters configured on the module. This includes environment variables, endpoint description (domain, ports, etc),
  - **Policies**: list of attached policies
  - **Effectors**: operations that can be invoked on module
- **Effector**: represents the possible actions that can be performed on each application module.
  - **Action**: the action that will be performed
  - **Description**: a description of the action.
- **Location**: represents the cloud provider where the managed application modules will be deployed.
  - **Provider**.
  - **Region**.

4.2.2. Interface

ID	getApplication
Description	Returns the details of an existing application (modules, status, location, etc).
Parameters	● **(String) applicationId**: ID of the application.

Response	● **(Application) application:** found application

ID	getModule
Description	Returns the details of an existing application module (status, location, policies, configuration, etc).
Parameters	● **(String) applicationId:** ID of the application. ● **(String) moduleId:** ID of the module.
Response	● **(Module) module:** application module details.

ID	getApplications
Description	Returns the list of deployed applications.
Parameters	
Response	● **(Application[]) applications:** list of available deployed applications

ID	createApplication
Description	Creates and deploys a new application, given a Deployable Application Model.
Parameters	● **(String) deployableApplicationModel:** application description
Response	● **(String) applicationId:** ID of the created application

ID	deleteApplication
Description	Removes a running application, releasing all cloud resources associated to it.
Parameters	● **(String) applicationId**: ID of the application to be removed.
Response	

ID	getEffectors
Description	Returns the list of available effectors for an Application Module.

Parameters	- **(String) applicationId**: ID of the application. - **(String) moduleId**: ID of the module where to retrieve the effectors.
Response	- **(Effector[]) effectors**: List of effectors available on the module.

ID	callEffector
Description	Triggers an effector action associated to a module.
Parameters	- **(String) applicationId**: ID of the application. - **(String) moduleId**: ID of the module that contains the target effector. - **(String) effector**: ID / action to be triggered - **(String[]) effectorParameterList**: additional parameters required by the effector.
Response	

ID	getLocations
Description	Retrieves the available locations in the deployer.
Parameters	
Response	- **(Location[]) locations**: list of currently supported locations.

ID	getAvailablePolicies
Description	Retrieves the available policies that can be attached to certain kind of module.
Parameters	- **(String) moduleType:** module type
Response	- **(Location[]) locations**: list of currently supported locations.

ID	addSlaAgreements
Description	Setup SLA agreements for a given application.

Parameters	• **(String) applicationId:** ID of the application • **(SlaAgreements) agreements:** agreements to be setup on the SLA service.
Response	

ID	addMonitoringRules
Description	Setup MODAClouds monitoring rules for a given application.
Parameters	• **(String) applicationId:** ID of the application • **(MonitoringRules) rules:** rules to be setup on the Monitor component.
Response	

5. Monitor

The Monitor component is in charge of monitoring that the QoS properties of the application modules and the whole application are not violated by the clouds in which they were deployed. For this component SeaClouds adopts Tower 4Clouds, a multi-cloud monitoring system developed within the MODACLouds european project. For the SeaClouds purposes, thank to Tower 4Clouds flexible and extendible architecture, specific pluggable components are implemented. Along this path the SeaClouds Monitor can be considered the conjunction of Tower 4Clouds plus a collections of self contained additional components provided by SeaClouds. It can be accessed using the Monitor API. Tower 4Clouds works by means of monitoring rules to be executed for controlling the corresponding cloud application. These monitoring rules include the resources to be monitored, the metrics to be collected, the formulas to be verified and the actions to be executed when the formulas become true. Such actions can include performing REST calls to other components, e.g., the Deployer (Live Model) or the Planner, enabling/disabling monitoring rules and generating new metrics as output. Such metrics could be used then to trigger other monitoring rules or to trigger reconfiguration actions.

5.1. Architecture & Design

Figure 4 shows the architecture of the Tower 4Clouds. The system is based on the idea of using Data Collectors to acquire monitoring data from various sources. In the figure we have as an example five different types of Data Collectors in charge of interacting with different components, either at the infrastructural or at the application level.

These Data Collectors can be installed anywhere it is suitable for the system under analysis. New Java Data Collectors can be implemented and plugged in starting from the data-collector-library that is available here [13], which implements the necessary protocol between a Data Collector and Tower 4Clouds. Non Java Data Collectors should be able to use the following MODAClouds platform interfaces:

- DDA (Deterministic Data Analyzer) interface as monitoring data must be sent to the DDA accessing the exposed interface.
- Monitoring Manager to retrieve the configuration and notify the monitored resources and the provided metrics per resource.

The Monitoring Manager is the main interface to access all Tower 4Clouds exposed services, like for example to install new monitoring rules, which can be also accessed using the provided Java client available here [14], which is on turn used by the data-collector-library.

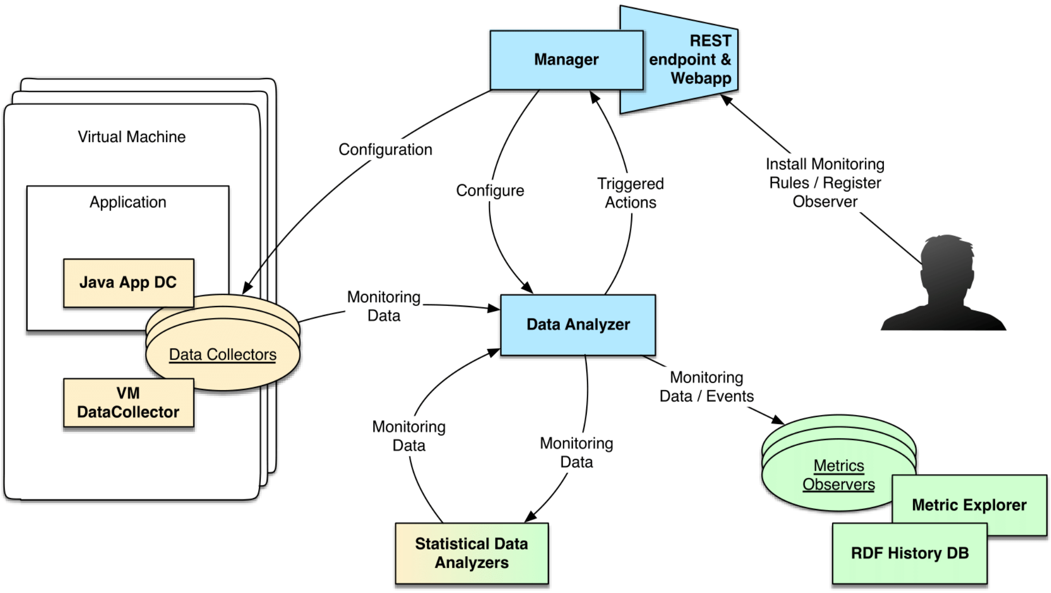
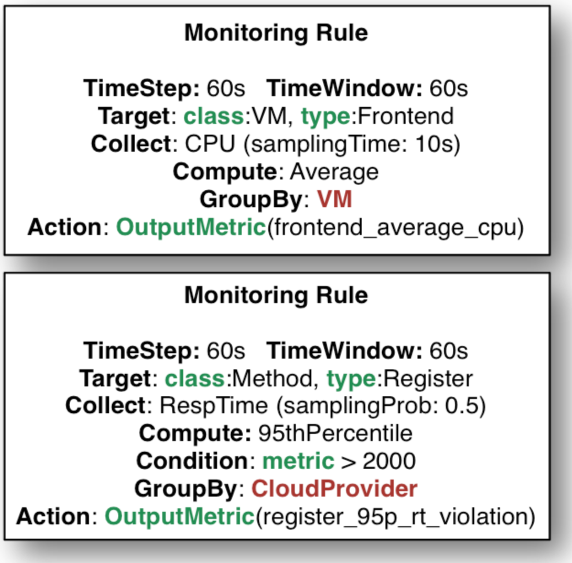


Figure 7. Architecture of the Monitor component

Each Data Collector first notifies the Monitoring Manager about the resources it is in charge to monitor and the provided metrics per resource. Then it retrieves its configuration, which depends on the installed rules looking for a metric the Data Collector is able to provide. Tower 4Clouds merges the information coming from all the installed Data Collectors and builds an unique monitoring model containing all the available resources in a hierarchical structure. Thanks to this, it can aggregate and filter data at various levels of abstraction. During their execution, Data Collectors send periodically data to the Deterministic Data Analyzer that filters them based on the definition of the Monitoring Rules it has installed. Monitoring rules predicate on Resources that are defined in the monitoring model. For instance, in Figure 8, you can see the pseudocode of two rules. The execution of the first rule will result in the following behavior. All data collectors in charge of monitoring CPU metric on all VMs of type Frontend will send a monitoring datum every 10 seconds. The Data Analyzer will compute every 60 seconds the Average of the last 60 seconds of data. It will also partition data per VM and output the results as frontend_average_cpu metric.

The second rule is acquiring the datum RespTime (response time) measured at the level of each call of the Register method. This datum is acquired by the data collector with probability 0.5. Every 60 seconds the data analyzer computes the 95th percentile of this response time considering the data acquired in the last 60 seconds. The result of this calculation is grouped by cloud provider. If such a metric is greater than 2000, the register_95p_rt_violation metric is produced. This metric could be observed, for instance, by the Deployer that could take some recovery action (e.g., scaling up).



```
Monitoring Rule

TimeStep: 60s   TimeWindow: 60s
Target: class:VM, type:Frontend
Collect: CPU (samplingTime: 10s)
Compute: Average
GroupBy: VM
Action: OutputMetric(frontend_average_cpu)
```

```
Monitoring Rule

TimeStep: 60s   TimeWindow: 60s
Target: class:Method, type:Register
Collect: RespTime (samplingProb: 0.5)
Compute: 95thPercentile
Condition: metric > 2000
GroupBy: CloudProvider
Action: OutputMetric(register_95p_rt_violation)
```

Figure 8. Monitoring rules pseudocode

```
<monitoringRule timeWindow="60" timeStep="60" id="cpuRule">
       <monitoredTargets>
              <monitoredTarget class="VM" type="Frontend" />
       </monitoredTargets>
       <collectedMetric metricName="CpuUtilization">
              <parameter name="samplingTime">10</parameter>
       </collectedMetric>
       <metricAggregation aggregateFunction="Average"
              groupingClass="VM" />
       <actions>
              <action name="OutputMetric">
                     <parameter name="name">frontend_average_cpu</parameter>
              </action>
       </actions>
</monitoringRule>
```

Figure 9. XML code of a simple monitoring rule.

Figure 9 shows the full code of the first monitoring rule in Figure 4. MODAClouds deliverable D5.2.2 [15] provides more details on this language.

A component willing to observe an OutputMetric generated by the DDA is called Metric Observer. It should basically be a REST endpoint able to receive monitoring data serialized as RDF JSON or as other available output formats. The Monitoring Manager exposes a specific service to attach an Observer to an available OutputMetric, specifying the endpoint to which it has to send metric values and the preferred format. As we said every component which needs to interact with Tower 4Clouds, or with the SeaClouds Monitor component, should access the Monitoring Manager exposed REST services.

Figure 10 shows the sequence diagram of the monitoring process and interactions.

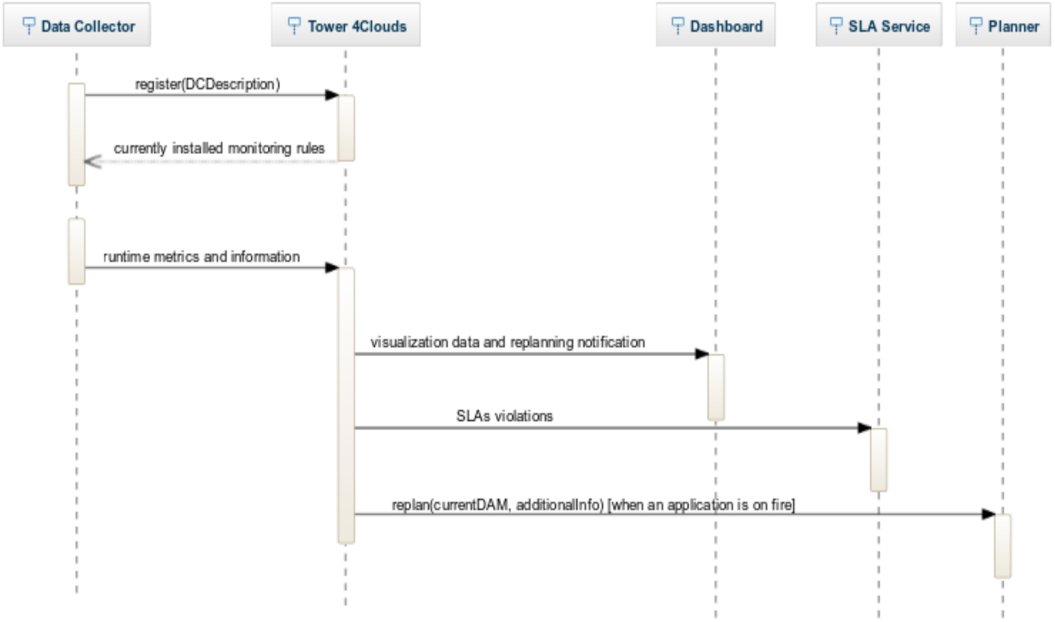

Data Collector
Tower 4Clouds
Dashboard
SLA Service
Planner
register(DCDescription)
currently installed monitoring rules
runtime metrics and information
visualization data and replanning notification
SLAs violations
replan(currentDAM, additionalInfo) [when an application is on fire]

Figure 10. Sequence diagram of the monitoring process and interactions.

5.2. Implementation & Monitor API

As we said the SeaClouds Monitor is realized by Tower 4Clouds plus some additional components, mainly new Data Collectors, developed for SeaClouds specific purposes. So far a new Data Collector ha been developed for the NURO case study providing NURO application specific metrics. New Data Collector are probably to be developed within the development of SeaClouds reconfiguration and replanning features.
Regarding the SeaClouds Monitor APIs, which have to be used by every the other SeaClouds component in order to interact with the SeaClouds Monitor as descrived in

the previous section, they are the same provided by the Monitoring Manager to access the main services of Tower 4Clouds.

The latter, as we described in the previous section, uses three core components, the Monitoring Manager, the Data Analyzer, and one or more Data Collectors, as specified below:

- The Monitoring Manager is the coordinator of MODAClouds platform.
- A Data Collector is responsible for collecting monitoring data from cloud resources and applications and to associate semantic information to the data.
- The Data Analyzer processes monitoring data coming from data collectors and tries to detect on-the-fly patterns that emerge directly from the data, without the need of major transformations of the data itself.

The code for the Monitor can be found at:
https://github.com/SeaCloudsEU/SeaCloudsPlatform/tree/master/monitor

In the following section, following the same schema adopted for the other SeaClouds components, a description of the available monitoring services is provided.

5.2.1. Monitoring Manager APIs

ID	GET /monitoring-rules
Description	Returns the list of installed monitoring rules.
Parameters	● None
Response	● Status: 200 OK Body: An XML object with a list of monitoring rules

ID	GET /monitoring-rules/{ruleId}
Description	Updates monitoring rules properties.
Parameters	● enabled ○ type: boolean ○ optional: no ○ description: enable/disable a rule
Response	● Status: 204 No Content

ID	POST /monitoring-rule

Description	Install monitoring rules.
Parameters	● An XML object with monitoring rules conforming to the Monitoring Rules Schema.
Response	● Status: 204 No Content

ID	DELETE monitoring-rule/{ruleId}
Description	Deletes a monitoring rule.
Parameters	● None
Response	● Status: 204 No Content

ID	GET /metrics
Description	Returns the list of Observable Metrics.
Parameters	● None
Response	● Status: 200 OK Body: A json array with the observable metrics.

ID	GET /metrics/{metricId}/observers
Description	Returns the list of observers attached to the metric.
Parameters	● None
Response	● Status: 200 OK Body: A json array with information about attached observers.

ID	POST /metrics/{metricId}/observers
Description	Attach an observer to the metric.
Parameters	● A JSON object containing the following fields: ● format ○ type: String ○ optional: yes ○ default: RDF/JSON

	○ description: specifies the serialization format ● protocol ○ type: String ○ optional: yes ○ default: HTTP ○ description: specifies the transfer protocol ● callbackUrl ○ type: String ○ optional: no, if protocol is HTTP ○ description: the full endpoint url of the observer ● observerHost ○ type: String ○ optional: no, if protocol is either TCP or UDP ○ description: the IP address of the observer ● observerPort ○ type: int ○ optional: no, if protocol is either TCP or UDP ○ description: the port to which the observer is listening to
Response	● Status: 201 Created Body: a json object containing the information about the observer just registered together with its server assigned id.

ID	DELETE /metrics/{metricsId}/observers/{observerId}
Description	Detach the observer from the metric.
Parameters	● None
Response	● Status: 204 No Content

6. SLA service

The SLA service component is in charge of mapping the low level information gathered from the Monitor into business level information about the fulfilment of the SLA defined for a SeaClouds application, using the SLA Service API. At runtime, the component is in charge of supervising that all the agreements are respected. Because QoS is already assessed by the Monitor Component, the SLA service is more focused on the enforcement of business oriented policies (QoB: Quality of Business), which represent a constraint on a metric that impact on the business of the application, and the business actions to apply in case of violation. It relies on the Monitor Component to fulfill this task. As a result of the enforcement process, the component stores the produced QoS and QoB violations and penalties, maintains the fulfillment state of each agreement and notifies the results to other components.

6.1. Architecture & Design

Figure 11 shows the architecture of the SLA service component.

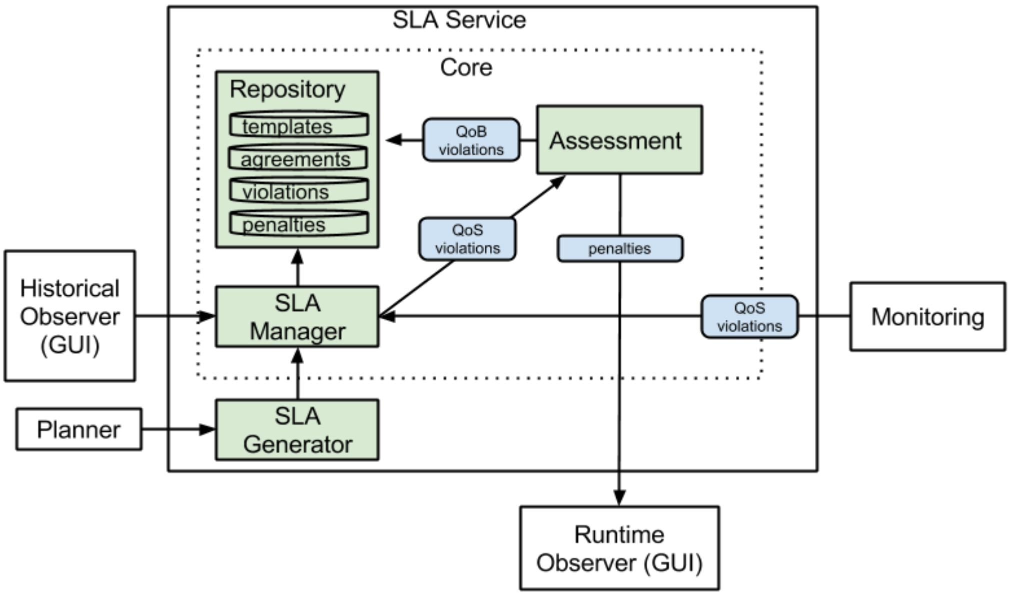


Figure 11. Architecture of the SLA Service

Figure 11 gives a detailed overview of the software components of the SLA Service and how they are related to other SeaClouds services.

The SLA Service enables the Service Level Agreements (SLA) management of business-oriented policies. The main responsibilities of the SLA service are: generating and storing WS-Agreement templates and agreements, and assessing that all the agreements (SLA guarantees) are respected by evaluating the business rules.

The SLA Service is an implementation of the WS-Agreement specification [16], which defines schemas for SLA Templates and SLA Agreements. A summary of the format of agreements and templates can be located at [17].

According to WS-Agreement:

- A template is a document used by the service provider to advertise the types of offers it is willing to accept.
- An agreement defines a dynamically-established and dynamically-managed relationship between a provider and a customer, where the object of this relationship is the delivery of a service by the provider to the customer.

A template or agreement contains functional and nonfunctional terms that describes the service being delivered. In SeaClouds, we are mostly interested in nonfunctional terms (Guarantee Terms), where a Service Level Objective (SLO) is defined as a constraint on a metric, and a list of business values describing the result of not fulfilling an objective.

The templates may be used as a base to create the actual agreements. Also, an agreement may contain additional terms not found in a template. For example, in SeaClouds, the agreements will contain Quality of Business (QoB) policies specified by the application designer, but not specified in a cloud provider template.

The purpose of QoB constraints is to perform a long-term analysis of the service, while the QoS constraints evaluated by the Monitor Component have a closer look at the performance of application.

The application designer can enrich the agreements based on the cloud provider templates with other QoB terms. Due to the fact that the cloud provider enforces its own SLA, and therefore, SeaClouds can not impose any penalty to the cloud provider, the actions that make sense to be specified here are unilateral actions. The most obvious action of this type is a migration of the modules in the affected cloud provider to another cloud provider. In SeaClouds, this is achieved with a replanning trigger generated by the SLA Service.

6.2. Implementation & SLA Service API

The SLA Service API provides the methods to manage templates and agreements of the two SLA levels identified in SeaClouds. The code of the SLA service is found at: https://github.com/SeaCloudsEU/SeaCloudsPlatform/tree/master/sla

6.2.1. Terminology

- **Agreement**: document that describes the delivered service, the involved parties, and the non-functional properties that the service must fulfill.
- **AgreementId**: Unique identifier of the agreement.
- **Template**: document that describes a provider offer. Actual agreements may be based on templates.
- **TemplateId**: Unique identifier of the template.
- **Guarantee term**: Term that express service guarantees in an agreement, define how guarantees are assessed and which compensation methods apply in case of meeting or violating the service guarantees.
- **ServiceId**: Identifier of a service being delivered. In the case of Customer - Application Provider level, it corresponds to the application id; otherwise, it corresponds to an identifier of the actual service offered by the cloud provider.

- **Resource**: Identifier of the entity using a service. Used in the Application Provider - Cloud Provider level, corresponding to the moduleId(s) being hosted.
- **Enforcement**: Process that evaluate the guarantee terms are being fulfilled.
- **QoB (Quality of Business)**: express a constraint over business-related metrics and the penalties and recovery actions that are applied in case this constraint is violated.

6.2.2. Interface

ID	getAgreement
Description	Retrieves the agreement identified by its id
Parameters	● **(AgreementId) Id**, WS-Agreement Id of agreement
Response	● **(Agreement)** WS-Agreement representation of agreement

ID	getAgreements
Description	Retrieves all the agreements that match the filter
Parameters	● **(ProviderId) provider**, Id of a Provider ● **(ServiceId) service**, Id of a Service ● **(ResourceId)** resourceId, Id of a Resource ● **(ConsumerId) consumer**, Id of a Consumer ● **(TemplateId) templateId**, WS-Agreement template id of agreements based on this template.
Response	● **(Agreement[])** WS-Agreement representation of agreements matching the filter

ID	createAgreement
Description	Creates an agreement for a given application.
Parameters	● **(AAM) aam**, Abstract Application Model of the application ● **(DAM) dam**, Deployable Application Model of the application
Response	● **(Agreement[])** WS-Agreement (including generated AgreementId) representation of the created agreements: - 1 agreement in Customer - Application Provider level - n agreements in Application Provider - Cloud Provider level, one per each cloud service used in the plan.

ID	updateAgreement
Description	Updates an existing agreement.
Parameters	● **(Agreement) description**, WS-Agreement representation of the agreement. It may be internally modified, so the parameter should not be taken as the agreement finally stored.
Response	● **(Agreement)** WS-Agreement representation of the agreement

ID	terminateAgreement
Description	Changes an agreement state to "Terminated" and stops any enforcement.
Parameters	● **(AgreementId) Id**, WS-Agreement Id of agreement
Response	

ID	getTemplate
Description	Retrieves the template identified by its id
Parameters	● **(TemplateId) Id**, WS-Agreement Id of template
Response	● **(Template)** WS-Agreement representation of template

ID	getTemplates
Description	Retrieves all the templates that match the filter
Parameters	● **(ProviderId) provider**, Id of a Provider ● **(ServiceId) service**, Id of a Service
	● **(Template[])** WS-Agreement representation of templates matching the filter

ID	createTemplate
Description	Creates a template
Parameters	● **(Template) description**, WS-Agreement representation of

	the template. It may be internally modified, so the parameter should not be taken as the template finally stored.
Response	● **(Template)** WS-Agreement representation of template (including generated TemplateId)

ID	getAgreementStatus
Description	Retrieves the status (violated, not violated) of service level objectives and the overall agreement
Parameters	● **(AgreementId) Id**, WS-Agreement Id of agreement
Response	● **(AgreementStatus)** status of agreements and its respective guarantee terms.

ID	startEnforcement
Description	Starts the enforcement of an agreement
Parameters	● **(AgreementId) Id**, WS-Agreement Id of agreement
Response	● **(Boolean)** status of enforcement

ID	stopEnforcement
Description	Stop the enforcement of an agreement
Parameters	● **(AgreementId) Id**, WS-Agreement Id of agreement
Response	● **(Boolean)** status of enforcement

ID	createProvider
Description	Creates a provider.
Parameters	● **(Provider) description**, name and description of provider to create
Response	● **(Provider)** SLA Service representation of provider (including generated ProviderId)

ID	getQoSViolations

Description	Get a list of QoS violations that match the filter
Parameters	● **(AgreementId) agreementId** ● **(ProviderId) provider**, Id of a Provider ● **(ServiceId) service**, Id of a Service ● **(ResourceId)** resourceId, Id of a Resource ● **(xs:datetime[]) dateInterval**, if provided, violation must be in the interval.
Response	● **(QoSViolation[])** Violations matching the filter

ID	getQoBViolations
Description	Get a list of QoB violations that match the filter
Parameters	● **(AgreementId) agreementId** ● **(ProviderId) provider**, Id of a Provider ● **(ServiceId) service**, Id of a Service ● **(ResourceId)** resourceId, Id of a Resource ● **(xs:datetime[]) dateInterval**, if provided, violation must be in the interval.
Response	● **(QoBViolation[])** Violations matching the filter

ID	receiveQoSViolation
Description	Notifies the SLA Service a QoS violation.
Parameters	● **(QoSViolation) violation**, violation sent by the Monitor
Response	

ID	receiveHealingNotification
Description	Notifies the SLA Service that the Policy Action was already done.
Parameters	● **(AgreementId) agreementId** ● **(PolicyId) policyId** ● **(QoSViolationId) violationId**
Response	

7. How to get and install the SeaClouds Integrated Platform

SeaClouds project has a Continuous Integration chain in place. 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[1].

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

7.1 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 Maven(3.5.5+)[2], Vagrant[3] and Apache Brooklyn[4] installed, then:

```
mvn clean install -DskipTests
cd usage/installer/target/seaclouds-installer-dist/seaclouds-installer
pushd .
cd byon
vagrant up
popd
nohup ./start.sh &
tail -f nohup.out
```

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

This spins up a virtual environment, made up of 2 VMs, which are accessible at `192.168.100.10` and `192.168.100.11`. Also, it starts up your instance of Apache Brooklyn on your workstation, accessible at:
http://localhost:8081.
Please double-check in nohup.out the correct url.

Finally, copy and paste “seaclouds-on-byon” blueprint[5] to deploy the SeaClouds platform on the 2 VMs created by Vagrant previously.

[1] http://jclouds.org
[2] https://maven.apache.org/
[3] https://www.vagrantup.com/
[4] https://brooklyn.incubator.apache.org/

7.2 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 on the "*seaclouds.yaml*" blueprint[6], it'd be possible to deploy SeaClouds against any IaaS provider supported by Apache Jclouds.

Remember to specify the target location. 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.

8. Conclusions

This document has been structured in two main topics, the first one introduced the SeaClouds Architecture together with the Dashboard and API overview. Later, for every runtime component, a description of the individual architecture as well as the implementation API has been presented. Such implementation will be continuously updated until the end of the project, following the continuous integration approach we have adopted.

[5] https://github.com/SeaCloudsEU/SeaCloudsPlatform/blob/master/usage/installer/src/main/assembly/files/blueprints/seaclouds-on-byon.yaml

[6] https://github.com/SeaCloudsEU/SeaCloudsPlatform/blob/master/usage/installer/src/main/assembly/files/blueprints/seaclouds.yaml

References

1. SeaClouds Project. Deliverable D4.5. Unified dashboard and revision of Cloud API (SeaClouds Consortium), March 2015
http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D4.5-Unified_dashboard_and_revision_of_Cloud_API.pdf

2. SeaClouds Project. Deliverable D3.3. SeaClouds discovery and adaptation components prototype (SeaClouds Consortium), July 2015 (to be published)

3. SeaClouds Project. Deliverable D5.1.2. Integrated Platform (SeaClouds Consortium), May 2015
http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D5.1.2-IntegratedPlatform.pdf

4. SeaClouds Project. Deliverable D2.4 Final SeaClouds Architecture (SeaClouds Consortium), February 2015 (to be published).

5. Bootstrap: A framework for developing responsive, mobile first projects on the web, http://getbootstrap.com/ 2015

6. Angular JS: HTML enhanced for web apps, https://angularjs.org/ 2015

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

8. Apache Brooklyn. https://brooklyn.incubator.apache.org/, 2014.

9. Apache jClouds. The Java Multi-Cloud Toolkit, https://jclouds.apache.org/, 2014.

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

11. SeaClouds Project. Deliverable D4.4. Dynamic QoS Verification and SLA Management Approach (SeaClouds Consortium), April 2015 http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D4.4-Dynamic_QoS_verification_and_SLA_management_approach.pdf

12. SeaClouds Project. Deliverable D4.3 Design of the run-time reconfiguration process (SeaClouds Consortium), February 2015 http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D4.3-Design_of_the_run-time_reconfiguration_process.pdf

13. MODAClouds Project github - Data Collectors

https://github.com/deib-polimi/tower4clouds/tree/master/data-collector-library

14. MODAClouds Project github - Tower 4Clouds
https://github.com/deib-polimi/tower4clouds/tree/master/manager/manager-api

15. MODACLouds Project. Deliverable 5.2.2. MODACloudML QoS abstractions and prediction models specification – Final version
http://www.modaclouds.eu/wp-content/uploads/2012/09/MODAClouds_D5.2.2_-MODACloudMLQoSAbstractionsAndPredictionModelsSpecificationFinalVersion.pdf

16. Web Services Agreement Specification (WS-Agreement)
http://www.ogf.org/documents/GFD.192, Open Grid Forum, 2011

17. Guide to WS-Agreement Language, Open Grid Forum https://packcs-e0.scai.fraunhofer.de/wsag4j/wsag/wsag-language.html, 2014.