SeaClouds Project

D4.2- Cloud Application Programming 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	**WP4 SeaClouds run-time environment**
Deliverable code	D4.2
Deliverable Title	Cloud Application Programming Interface
Nature	Prototype
Dissemination Level	Public
Due Date:	M12
Submission Date:	10th October 2014
Version:	1.0
Status	Final
Author(s):	Miguel Barrientos (UMA), Jose Carrasco (UMA), Javier Cubo (UMA), Adrián Nieto (UMA), Román Sosa (ATOS), Andrea Turli (CloudSoft), PengWei Wang (UPI)
Reviewer(s)	Diego Pérez (POLIMI), Christian Tismer (NURO)

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)	

Table of Contents

List of Figures

List of Tables

Terminology clarification

The following vocabulary will be used along the document.

Component: Each of the main parts that compose SeaClouds Platform (Planner, Deployer, SLA, Discoverer, etc).

Dashboard: SeaClouds user interface.

Application Module: Entities which conforms the deployable elements (eg. Web Server, Database…).

Policy: Action/s that should be taken after a condition is triggered (eg. response time > 300, QoS or QoB rules…).

Executive Summary

This deliverable presents the first specification of the SeaClouds Application Programming Interface (API). It describes the common language that every SeaClouds component will use to interact with between them. To illustrate the functionality that is covered by the interface, we provide along this deliverable some motivating examples as possible use cases of it. Also, a very early proposal of the dashboard is presented. It is worth mentioning that in the DoW we initially considered to include an early prototype of the SeaClouds run-time environment in this document. Although some descriptions are given along the document, and it can be considered as a prototype deliverable, but we have also considered distributing some design information about the early prototype in Deliverables D3.1 [1] and D4.1 [2], and more implementations details in Deliverable D5.4.1 [3], leaving other aspects representing the details APIs (with the methods necessary), and the dashboard implemented and its functionalities, in this document.

1. Introduction

In this document, we present the initial design of the API for SeaClouds. Thus, in order to facilitate reuse and modularity, all the operations performed by the SeaClouds components will be provided as a unified API, and a dashboard will be provided as well.

Although the heterogeneous commercial cloud offerings differ in aspects as programming languages, application frameworks, etc., there are inherent similarities in the way they manage the life-cycle of the applications that are targeted for, and deployed upon them. However, cloud users currently need to interact with the providers following each of their specifications of each provider. This situation can be overcome by producing a producing a generic application and platform management API that is language, framework, and platform neutral.

Then, here we propose the first specification for the SeaClouds Unified management API for Clouds for the use cases around deploying, stopping, starting, and updating applications, this specification increases consumers' ability to port their applications between cloud offerings. Moreover, we also provide an initial description of the SeaClouds Dashboard in charge of administrating services distributed between clouds.

Section 2 presents the overview of the SeaClouds API. In Section 3, we present a set of use cases to understand more easily the methods we need to define in the APIs for every component of the SeaClouds platform. Section 4 presents the first specification of the SeaClouds API for each component. Section 5 describes a very early version of the proposal of the Unified Dashboard. Section 6 concludes the document.

1.1 Glossary of Acronyms

Acronym	Definition
QoS	Quality of Service
QoB	Quality of Business
SLA	Service Level Agreement
GUI	Graphical User Interface
API	Application Programming Interface
DAM	Deployable Application Model
YAML	YAML Ain't Another Markup Language

Table 1: Acronyms

2. Overview of the SeaClouds API

Within SeaClouds Platform all inter-client communication are performed through the unified API. This abstraction warrants the independence between component's implementation (Discoverer, Planner, Deployer, Monitor, SLA). The implementation of every component of the SeaClouds platform must provide a connector with the SeaClouds API, which translates from each proprietary component implementation to a homogeneous model.

The API definition focuses on defining the required tools to provide all the necessary functionality while it avoids as many low-level details as possible. The API keeps the structure of the previously defined architecture in Deliverable D2.2 [4], as is shown in Figure 1. Therefore Discoverer, Planner, Deployer, Monitor and SLA Service APIs encapsulate all the interaction within the SeaClouds Platform.

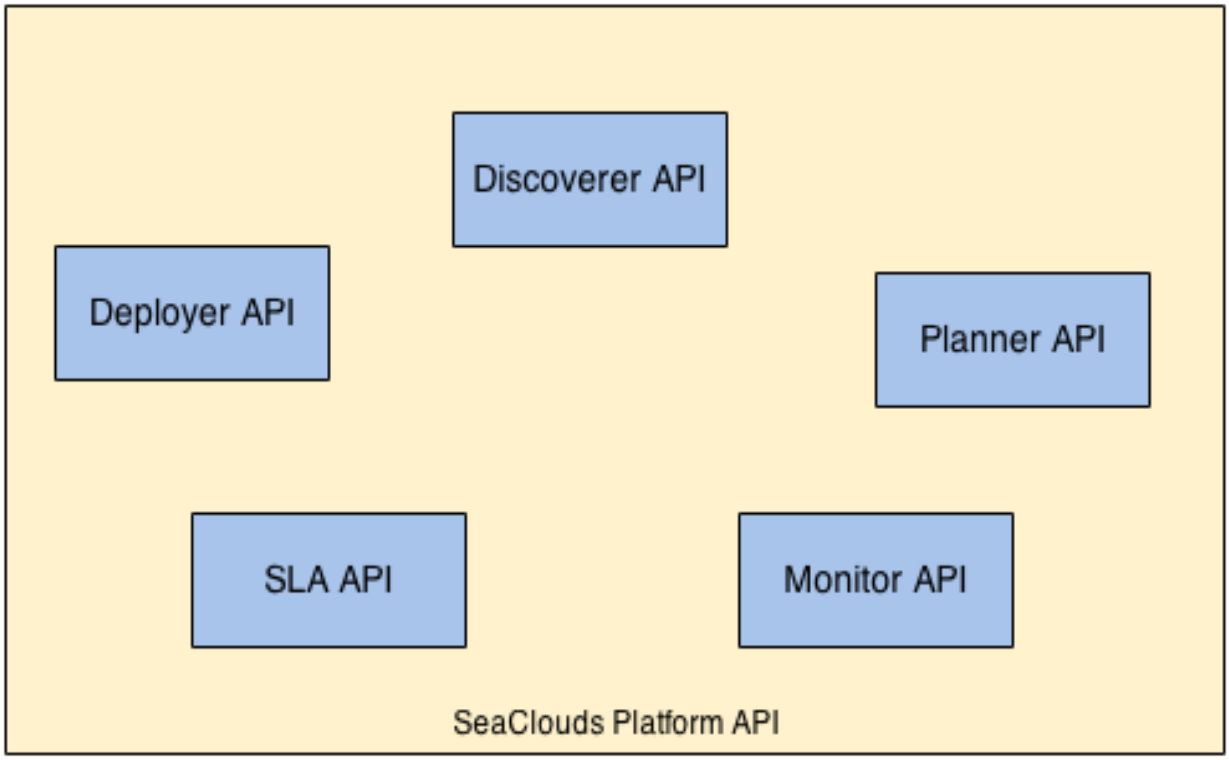

Figure 1. SeaClouds API Structure

3. Motivating examples

The API specification is designed to satisfy the most common use cases defined in the previous deliverable D2.1 about the requirements of SeaClouds [5]. In the following cases, we try to avoid the end-user interaction as we want to reflect the interaction between components inside SeaClouds.

3.1. Deploying a new application

Deploying a new application requires to define a Module Profile. The process involves Planner API, Discoverer API, Deployer API, Monitor API and SLA API:

A. The Planner takes the Module Profile as an input (**POST ModuleProfile**).
B. The Planner asks the Discoverer Component the best existing match (based on the Application Module constraints and the provider features) for this Module Profile setup (**GET Service**).
C. The Planner uses the Matchmaker information to select the suitable matches and the most appropriate among them in order to generate the DAM (Deployable Application Model).
D. The DAM is sent to the Deployer Component (**POST CreateAndStartApplication**).
E. Metrics are wired to the deployed application modules by the Monitor Component from Monitoring Agents (**POST MonitoringAgents**).
F. QoS Policies are configured and attached (if needed) on the Monitor Component (**POST AttachPolicy**).
G. QoB rules are added to the SLA Service (**POST Agreement**). SLA Service might need to get information about existing Metrics and Policies from Monitoring.

After this process finishes successfully, the application is deployed.

3.2. Stopping running application

To stop a running application the user must provide the Application ID. This process involves Deployer API, Monitor API and SLA API:

A. SLA rules are removed from the SLA Service (**DELETE Agreement**).
B. QoS policies are removed from the Monitor (**DELETE Policy**).
C. MonitoringAgents are removed from the Monitor (**DELETE MonitoringAgent**).
D. The Deployer stops the application and free the resources (**DELETE Application**).

After this process finishes successfully, the application is stopped and removed from SeaClouds Platform.

3.3. Adding a policy to a running application

To modify the policies associated to a running application the user must provide Application ID, the Application Module ID and the Policy, which is composed by the Metric ID(s) and the Policy configuration parameters (e.g. “{metric} < 100”). This process involves Monitor API and SLA API:

A. The Monitor Component checks if the Application ID and the Module ID exist and are already registered and if the metric associated with the policy is available for either Application ID or Module ID (**GET AvailableMetrics**).
B. The SLA Service calls the Monitor component to register the policy using the Monitor API (**POST AttachPolicy**).

After this process finishes successfully, the SLA Service will register the policy and it will be notified about those policy violations.

3.4. Executing a healing policy after SLA violations

When a policy is registered by the SLA Service, it begins to be watched by the Monitor. If a policy violation is triggered, the associated action must be performed. This process involves Monitor API, SLA API and Deployer API.

A. The Monitor detects that a QoS Policy has been violated (**GET QoSViolations**).
B. The Deployer does the previously defined action (triggers an effector by using **POST CallEffector**).
C. The Monitor notifies the SLA Service that healing action was done (**POST HealingPolicyDone**).

After this process finishes successfully, the application should satisfy the constraints (QoS, QoB…) previously defined.

3.5. Migrating a running application

When a provider cannot match the Application Module requirement, if the user already defined a migration Policy (which could not require user interaction), the Module could be moved to a new Provider to satisfy the Policy requirements. This process involves Monitor API, Deployer API, Planner API and Discovery API.

A. The Monitor checks the status of the application data sensors and realizes about the violation of the condition inside the policy (transparent to the other modules).
B. The Monitor checks the QoS and QoB violations using the SLA API (**GET QoSViolations, GET QoBViolations).**
C. The Dashboard uses the MonitorAPI to check pending task according to the Policies violations (**GET ApplicationStatus**).
D. If the user confirms, the Planner executes a re-planning process that will propose an application deployment where the problematic module/s have been migrated.
    a. The re-planning process is notified to the user by the SeaClouds Dashboard (by a change in the application status).
    b. The Planner uses the Discoverer API to discover the existing cloud providers; it filters the ones that do not match the application modules necessities and selects a suitable option for each module among these possible candidates. The set of selected options are sent to the Dashboard (**GET Solution**, **GET SimilarSolution**).
    c. If the proposal is accepted by the user, a DAM containing this proposal is generated. This DAM contains the differences from the output of precedent planning process (i.e., the differences between the previous plan and the current one). In this way, the Deployer receives more clear information regarding parts of the DAM that have not been modified and therefore the modules that do not need to be stopped and deployed again in the same location (**POST CreateAndStartApplication**).
    d. The Planner uses the Deployer API checks the dependences among the application modules which have to be migrated and others (**GET Dependencies**).
E. The Deployer re-deploys the application, performing the necessary actions to migrate the modules in the DAM that are marked as different from the previous DAM.

a. The Planner notifies the Application depending entities with the Deployer API (**POST CallEffector)**.
b. The Planner stops the necessary application modules to migrate the target application module using the Deployer AP I (**POST CallEffector)**.
c. The target application module is stopped by the Planner (**POST CallEffector**).
d. The target application module is deployed in the selected cloud provider by the Planner using the Deployer API (**POST CallEffector**).
e. The migrated application module is initialized using the Deployer API (**POST CallEffector**).
f. The relations and dependencies among the application modules are solved by the Planner using the Deployer API.

After this process finishes successfully, the application is migrated successfully.

4. SeaClouds API Specification

In this section, the APIs for every component of the SeaClouds platform are defined. Note that this is a first specification of the API, and this could evolve in next analysis and implementations (which will be reflected in next deliverable D4.5).

4.1. Discoverer API

Ideally, the Discoverer API allows an administrator to retrieve information about the cloud profile of each cloud provider. Thus, the Discoverer Component is able to retrieve automatically the information related to the services featured by each provider. This information is stored in a repository, whose content is kept updated by the Discoverer itself. In this scenario, the Discoverer API offers a mean to retrieve, manage and manipulate the information stored in the cloud profile repository.

4.1.1. Discoverer API Terminology

- **Repository**: a data structure used to store the services offered by cloud providers.
- **ServiceType**: reusable entity that defines the type of a general cloud service. It includes the properties, the requirements, the capabilities and the policies of such a module.
- **ServiceTemplate**: actual instantiation of a node type that specifies the occurrence of a Node Type as a module of a service.
- **AbstactService**: a general service needed by a module to run.
- **ConcreteService**: a specialization of an AbstractService.

4.1.2. Discoverer API Specification

Method	Params	Output	Description
POST ServiceTemplate	cloudId, ServiceTemplate	ServiceTemplateId	Adds a new ServiceTemplate to the repository
DELETE ServiceTemplate	ServiceTemplateId	boolean	Removes a ServiceTemplate from the repository
GET Service	ServiceType	boolean	Given a Service Type checks if there is any serviceTemplate of that type in the repository
GET Service	ServiceType	List of abstract ServiceTemplates	Retrieves for a certain ServiceType all the correspondent ServiceTemplates. The specified ServiceType can be either abstract or concrete

Table 2: Discoverer API specification

4.2. Planner API

The Planner API receives the information about the Application Modules, how they interact, their requirements, constraint and features.

4.2.1. Planner API Terminology

- **Application:** the cloud application provided by the user, which consists of classes cModule, cQoSRequirement, and cTopology.
- **CloudService:** the cloud offerings advertised by cloud providers.
- **CandidateList:** linked list, which is used to store the candidate cloud offerings for each module.
- **ResultedSolution:** represents the resulted (optimal) solution(s) that is composed of pairs of module and its corresponding cloud offering.
- **CurrentSolution:** the solution that is used currently.
- **NewSolution:** after replanning, an alternative solution, similar to the current one, is generated. The ResultedSolution, CurrentSolution and NewSolution are with the same type (i.e., cSolution, represents the solutions that is composed of pairs of modules and its corresponding cloud offerings after planning).

4.2.2. Planner API Specification

Method	Params	Output	Description
GET CandidateList	Application, CloudService	CandidateList	Gets the list of candidate clouds offerings for each module
GET Solution	Application, CloudService, CandidateList	Solution	Gets the solutions and returns
GET SimilarSolution	Application, CloudService, CandidateList, CurrentSolution	Solution	Gets a similar solution from the current used, and this will be useful for replanning
POST ModuleProfile	Module Profile		The application will be deployed
POST ReconfigurationConfirmation	Module Profile		The reconfiguration plan is approved

Table 3: Planner API specification

4.3. Deployer API

The Deployer API provides the resources to manage plans and modules. This API contains the method for managing the application deployments, e.g. deploying a Deployable Application Model, removing an Application, removing and undeploy the Application Modules.

4.3.1. Deployer API Terminology

- **Effector**: Mechanism to interact with the Application Modules (eg. restart module).
- **ConfigParameters:** Configuration parameters for an Application Module (see [2]).
- **Location:** Providers used to deploy the application modules.

4.3.2. Deployer API Specification

Method	Params	Output	Description
GET Applications	----------	List of Application	Returns the list of deployed applications
POST createAndStartApplication	Deployable Application Model	ApplicationId	Deploys new application
DELETE Application	ApplicationIdOrName	ApplicationId	Removes a running application
GET ConfigurationParameters	ApplicationIdOrName, ModuleIdOrName	ConfigParameters	Returns the available ConfigParameters in an Application Module
GET AvailableEffectors	ApplicationIdOrName, ModuleIdOrName	List of Effectors	Returns the list of available effectors for an Application Module
POST CallEffector	ApplicationIdOrNAme, ModuleIdOrName, Effector, EffectorParameterList		Triggers an effector in the desired module.

GET ApplicationStatus	ApplicationIdOrName	ApplicationStatus	Returns the Application status (stopped, running, error…)
GET ComponentStatus	ApplicationIdOrNameD, ModuleIdOrName	ComponentStatus	Returns the Application Module status
GET ApplicationTree	ApplicationId	ApplicationTree	Returns the Application Module hierarchy
GET SupportedLocations	----------	List of Supported Locations by the deployer	It retrieves the available locations in the deployer
POST createNewLocation	locationSpec	Location	Creates a new location
GET Dependencies	ApplicationId, ModuleId	List of Application Modules dependencies	It retrieves a list which contains the dependences among the ModuleID and the rest of the Application Modules

Table 4: Deployer API specification

4.4. Monitor API

The Monitor API provides the functionality related with the monitoring tasks, such as retrieving information from monitoring agents or managing the active policies.

4.4.1. Monitor API Terminology

- **MonitoringAgent:** describes the monitoring data provider related to a registered application.
- **Metric:** each of the basic monitored data that describes different properties generated from the managed applications.
- **MetricCatalog:** complete list of the metrics being retrieved from a certain module.

- **Policy:** represents a simple QoS property over *Metrics* data (e.g. ResponseTime < 10.0).
- **PolicyConfig:** parameters to define a *Policy.*
- **PolicyStatus:** shows the status of the property that is represented by a *Policy,* such as "OK" or "Violated".

4.4.2. Monitor API Specification

Method	Params	Output	Description
POST Application	ApplicationId	ApplicationId	Registers an application in the Monitor Component
POST MonitoringAgent	ApplicationId, MonitoringAgent	ApplicationId	Attaches a MonitoringAgent to a previously registered application
GET MonitoringAgentsBy Application	ApplicationId	Monitoring Agents	Retrieves the list of monitoring agents being used on a registered Application
DELETE MonitoringAgent	ApplicationId, MonitoringAgent	ApplicationId	Removes the MonitoringAgent attached to some application and the metrics retrieved by it
GET AvailableMetrics	ApplicationId, ModuleId	MetricCatalog	Gets the list of available metrics from a certain application module
GET MetricValue	ApplicationId, ModuleId, Metric	Value or values	Fetches the current value of the selected metric
POST AttachPolicy	ApplicationId, ModuleId, PolicyConfig	PolicyId	Attaches a policy to an existing entity
POST StartPolicy	ApplicationId, ModuleId, Policy	PolicyId	Starts a policy

POST StopPolicy	ApplicationId, ModuleId, Policy	PolicyId	Stops a policy
DELETE Policy	ApplicationId, ModuleId, Policy	PolicyId	Removes a policy from being associated with an entity
GET Policy	PolicyId	Policy	Fetches the policy details
POST subscribeToPolicy	ApplicationId, ModuleId, PolicyId, Subscriber	PolicyId	Subscribes a listener component to the events generated from a policy

Table 5: Monitoring API specification

4.5. SLA Service API

The SLA Service API provides the necessary functionality to manage the SLAs defined by the administrator.

4.5.1. SLA Service API 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.
- **Guarantee term**: Term that expresses service guarantees in an agreement, defines how guarantees are assessed and which compensation methods apply in case of meeting or violating the service guarantees.
- **Enforcement**: Process that evaluates whether the guarantee terms are being fulfilled.
- **Enforcement Job**: Entity associated to an agreement that stores if an enforcement process is enabled, and date executions of the enforcement.
- **QoB (Quality of Business)**: expresses a constraint over business-related metrics and the penalties and recovery actions that are applied in case this constraint is violated.

4.5.2. SLA Service API Specification

Method	Params	Output	Description
GET Agreement	Id	Agreement	Retrieves the agreement identified by its id

GET Agreements	Filter (provider, resource)	Agreement[]	Gets the list of agreements that match the filter
POST Agreement	Application, Module, Description	Agreement	Creates an agreement for a given application. The agreement is generated using a simplified structure, i.e.: id, provider, qos, qob. Return agreement in YAML format.
PUT Agreement	Id, Description	Agreement	Updates an existing agreement
DELETE Agreement	Id		Deletes an existing agreement
GET AgreementStatus	Id	AgreementStatus	Retrieves the status (violated, not violated) of service level objectives and the overall agreement
GET Enforcement	AgreementId	Enforcement	Retrieves the enforcement job of an agreement
GET startEnforcement	AgreementId		Starts the enforcement of an agreement
GET stopEnforcement	AgreementId		Stops the enforcement of an agreement
GET Provider	Id	Provider	Retrieves a provider
POST Provider	Description	Provider	Creates a provider
PUT Provider	Description	Provider	Updates an existing provider
DELETE Provider	Id		Deletes an existing provider
GET QoSViolations	Filter (AgreementId, Application, Module, Resource, Provider, Dateinterval, …)	QoSViolation[]	Gets a list of QoS violations that match the filter

GET QoBViolations	Filter (agreementId, Application, Module, Resource, Provider, DateInterval, …)	QoBViolation[]	Gets a list of QoB violations that match the filter
POST HealingPolicyDone	AgreementId, PolicyId, QoSViolationId		Notifies the SLA Service that the Policy Action was already done.

Table 6: SLA Service API specification

5. SeaClouds Dashboard proposal

In this section, we describe a very early version of the proposal of the Unified Dashboard, in which SeaClouds is working (and whose objective is to present more in detail for the next deliverable, D4.5 Unified dashboard and revision of Cloud API), that will be used to access to the platform services.

The organization of the Dashboard is divided into several sections (Figure 2):

- **Home.** SeaClouds Dashboard welcome screen. Shows information about the applications and Seaclouds project.
- **Module Profile Designer.** GUI to define the application abstract architecture. It is not available yet.
- **DAM generator.** Cloud provider selector, it generates the DAM based on the user choices / Discoverer suggestions. It is not available yet.
- **Deploy new Application.** Takes a DAM from a YAML encoded text, and deploys the application.
- **Monitor applications.** Monitor dashboard entry point. It provides graphical information about the deployed application parameters.

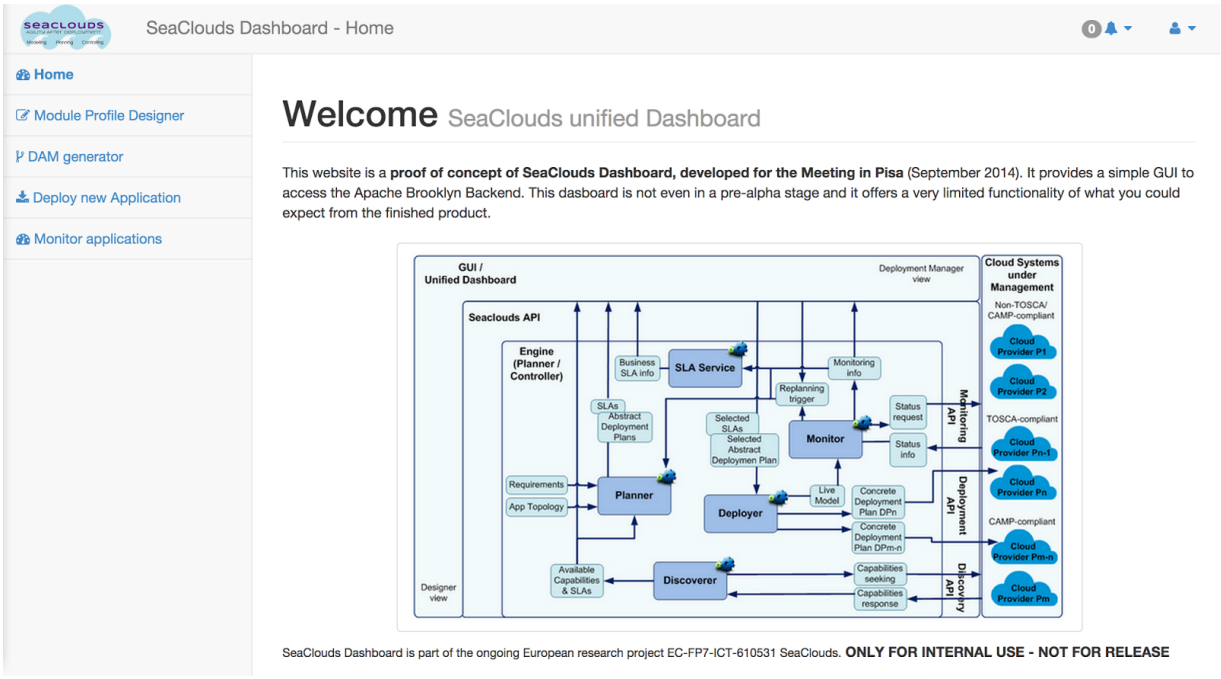

Figure 2. SeaClouds Dashboard home screen

The dashboard is designed to provide a clear and simple way to interact with SeaClouds in all stages of the life-cycle of an application, starting from the application module definition (module profile) to the analysis of application's SLA violations. In order to establish a common GUI across the whole SeaClouds platform, we need to specify the framework that we are going to use during the development process of the dashboard. In this release we present a basic frontend using HTML5 and JavaScript libraries. In addition, for this proposal we use Brooklyn

API [6] to gather the necessary information, instead of the SeaCloud API which will be used in next releases.

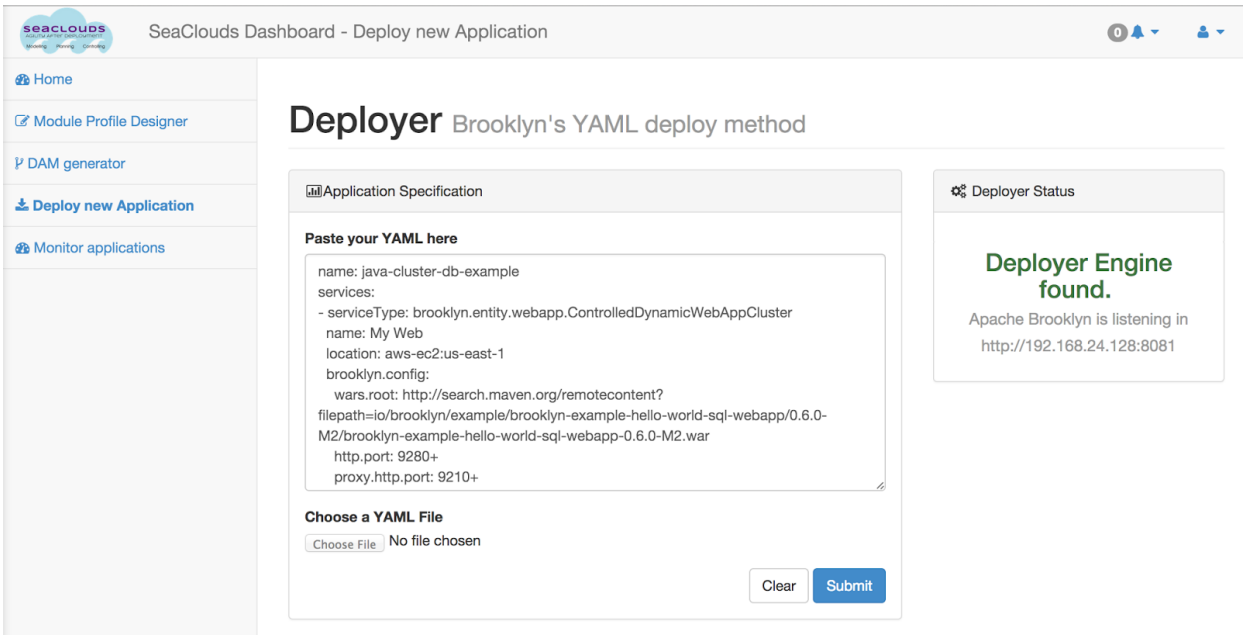

Figure 3. Deploying an application using SeaClouds

The functionality provided in this proposal allows to deploy an application by using a YAML file with Brooklyn Syntax (Figure 3). Finally, Figures 4 and 5 depict the application overview, showing the deployed application status and allowing the user to retrieve more detailed information about the main metrics from the entities using different kind of graphs according to the different nature of the metrics and the expected use and comparison with other data.

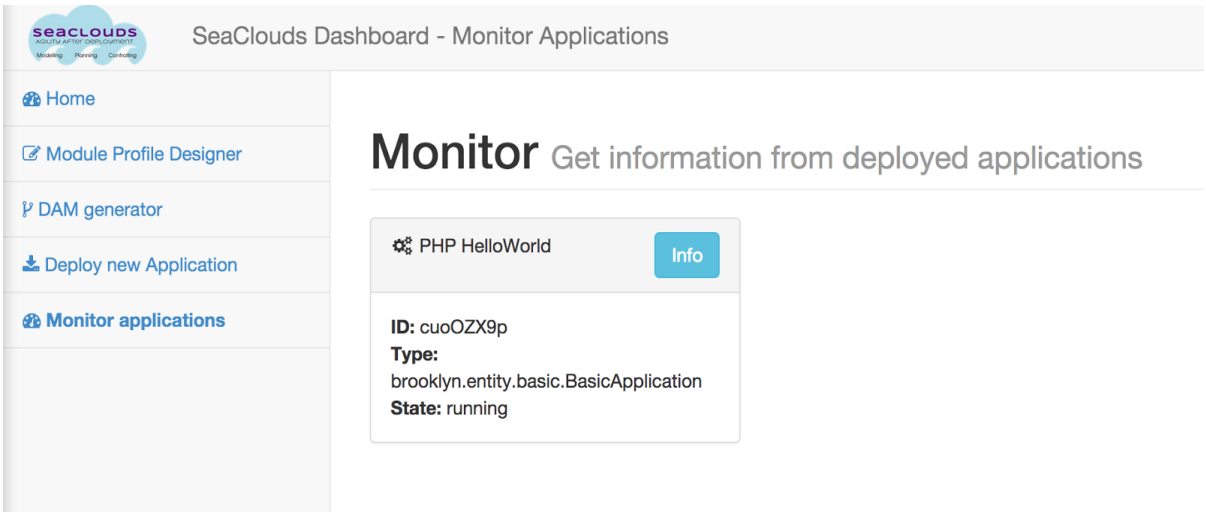

Figure 4. Application status shown at SeaClouds Dashboard

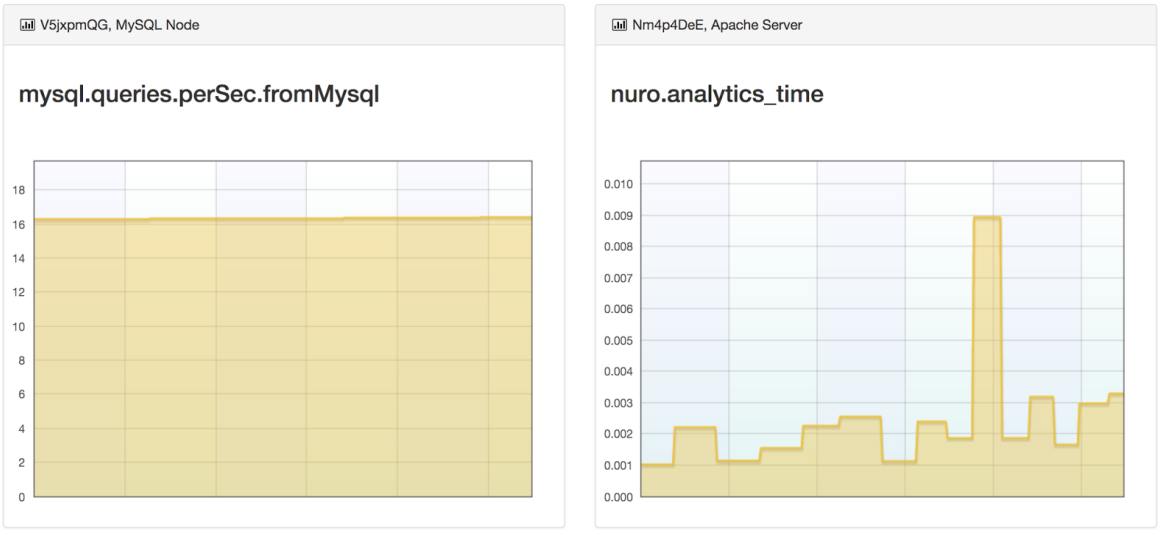

Figure 5. Live monitor of the application metrics

6. Conclusions

In this deliverable, we have presented the first design of the SeaClouds Application Programming Interface (API) structure, describing the common language used by the SeaClouds components to interact among them, and thus, including methods for deploying, stopping, starting, and updating applications on cloud. The API has been defined as a generic application and platform management independent of the language, framework, and platform. This deliverable has also presented a set of examples of interactions between components and a very early proposal of the dashboard.

In the next deliverable, the final design of the unified management dashboard, including the set of commands to administrate services distributed between clouds, and a revision of the Cloud Application Programming Interface will be delivered, so in that deliverable, also we will review the unified API.

References

[1] Deliverable D3.1 “Discovery design and orchestration functionalities: first specification”.

[2] Deliverable D4.1 “Definition of the multi-deployment and monitoring strategies”.

[3] Deliverable D5.4.1 “Definition of the multi-deployment and monitoring strategies”.

[4] Deliverable D2.2 “Initial architecture design SeaClouds Platform to Review”.

[5] Deliverable D2.1 “Requirements for the SeaClouds Platform".

[6] Brooklyn API definition https://brooklyn.incubator.apache.org/v/0.7.0-M1/use/api/?help-doc.html