seaclouds
AGILITY AFTER DEPLOYMENT
Modelling Planning Controlling

# SeaClouds Project
# D2.4 Final SeaClouds Architecture

| | |
|---|---|
| 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 | WP2. Requirements Analysis, overall Architecture and Standardization |
| Deliverable code | D2.4 |
| Deliverable Title | Final SeaClouds Architecture |
| Nature | Report |
| Dissemination Level | Public |
| Due Date: | M16 |
| Submission Date: | 16th February 2015 |
| Version: | 1.0 |
| Status | Final |
| Author(s): | Dionysis Athanasopoulos (POLIMI), Miguel Barrientos (UMA), Jose Carrasco (UMA), Javier Cubo (UMA), Francesco D’Andria (ATOs), Elisabetta Di Nitto (POLIMI), Adrián Nieto (UMA), Román Sosa (ATOS), Christian Tismer (NURO), PengWei Wang (UPI) |
| Reviewer(s) | Dionysis Athanasopoulos (POLIMI), Elisabetta Di Nitto (POLIMI), Andrea Turli (Cloudsoft) |

## 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 | 19/01/15 | ToC and deadlines defined | Javier Cubo |
| 0.2 | 26/01/15 | First version all sections | Javier Cubo |
| 0.3 | 30/01/15 | First contributions | Javier Cubo, Francesco D'Andria, Adrián Nieto, PengWei Wang, Román Sosa, Christian Tismer, Elisabetta Di Nitto |
| 0.4 | 11/02/15 | Revision of first contributions and second draft | Javier Cubo, Francesco D'Andria, Román Sosa, PengWei Wang, Christian Tismer, Elisabetta Di Nitto, Dionysis Athanasopoulos, Miguel Barrientos, Jose Carrasco, Adrián Nieto |
| 0.5 | 13/02/15 | Review of a more stable version | Dionysis Athanasopoulos, Elisabetta Di Nitto, Andrea Turli |
| 1.0 | 13/02/15 | Stable version after the reviews | Javier Cubo, Adrián Nieto |

## Table of Contents

## List of Figures

## List of Tables

## Executive summary

This deliverable presents the final and stable architecture of the SeaClouds platform. SeaClouds provides the foundation for allowing “Agility After Deployment” providing necessary tools and a framework for Modelling, Planning and Controlling Cloud Applications. In this document, the SeaClouds components are described both at design time and at run-time. Also, a mapping between the SeaClouds Open Reference Architecture and the case studies is considered in this deliverable.

# 1. Introduction

The final architecture of the SeaClouds platform is presented in this document, following the initial architecture presented in Deliverable D2.2 [1]. Prior to describing the final SeaClouds architecture, we briefly provide the scope of SeaClouds and the objectives met by its architecture.

## 1.1 Scope and objectives

How to design, deploy and manage, in an efficient and adaptive way, complex applications across multiple heterogeneous cloud platforms is one of the problems that have emerged with the cloud revolution. In this document, we present the Final SeaClouds Architecture.

Considering an agile execution approach both at design and runtime, SeaClouds works towards giving organizations the capability of ***"Agility After Deployment"*** for cloud-based applications (see Figure 1), following an agile execution approach. The approach is based on the concept of service orchestration and designed to fulfill functional and non-functional properties over the whole application. Applications will be dynamically reconfigured by changing the orchestration of the services they use when the monitoring will detect that such properties are not respected. So, **SeaClouds' main goal is the development of a novel platform which performs a seamless adaptive multi-cloud management of service-based applications**, with four specific objectives. For each objective, the SeaClouds consortium plans to tackle a set of challenges, which are described in depth in Deliverable D2.2, related to Initial architecture and design of the SeaClouds platform [1].

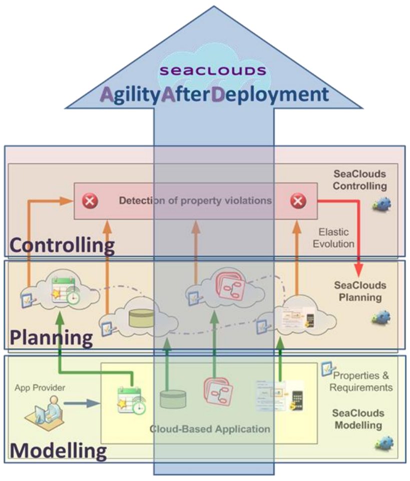


Figure 1. "Agility After Deployment" strategy in SeaClouds.

Objectives of SeaClouds:

- *Orchestration and adaptation of services distributed over different cloud providers.*
- *Offer unified application management of services distributed over different cloud providers.*
- *Monitoring and run-time reconfiguration operations of services distributed over multiple heterogeneous cloud providers.*
- *Compliance with major standards for cloud interoperability.*

## 1.2 Overview of the document

The structure of the rest of this document is the following. Section 2 presents the approach and functionalities of SeaClouds. It introduces the final architecture of the platform and illustrates the SeaClouds approach using a storyboard based on the NURO Cloud Gaming, one of the SeaClouds case studies. Section 3 describes the main components: Discoverer, Planner, Deployer, Monitor, SLA, and GUI and Dashboard. Section 4 maps the SeaClouds architecture components to the case studies. Section 5 closes the deliverable.

## 1.3 Glossary of Acronyms

| 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 |
| OASIS | Organization for the Advancement of Structured Information Standards |
| TOSCA | Topology and Orchestration Specification for Cloud Applications |
| CAMP | Cloud Application Management for Platforms |
| GUI | Graphical User Interface |
| API | Application Programming Interface |
| APP | Application |
| DB | Database |
| AAM | Abstract Application Model |
| DAM | Deployable Application Model |
| DevOps | Development and Operations |
| DC | Data Collectors |
| DDA | Deterministic Data Analyzer |
| KB | Knowledge Base |
| ObjS | Object Store |
| AMSoC | Application Management System over Cloud |
| QoE | Quality of Experience |
| C-AP | Customer - Application Provider |

Table 1. Acronyms.

## 2. Final SeaClouds Architecture

This section presents the SeaClouds approach, functionalities and architecture, illustrating the approach using a storyboard.

### 2.1. The SeaClouds Approach

The **SeaClouds** project aims to develop a new open source framework which performs Seamless Adaptive Multi-Cloud management of service-based applications. The framework consists of an Application Management System over Cloud (AMSoC) at different levels, IaaS and PaaS that implements a ***DevOps*** approach for continuous software delivery. This approach enables application providers to mitigate business risks and reduce time to market and customer feedback.

SeaClouds allows developers to design, deploy, manage and configure **complex applications** across **multiple and heterogeneous clouds**, something unfeasible hitherto.

Figure 2 shows the cloud architecture situation before (top) and after SeaClouds (bottom). Without SeaClouds, services can only be deployed, managed and monitored across multiple clouds as standalone applications, and not as part of composite applications. This has the consequence that there is no support for synchronized deployment and unified monitoring, which implies that the QoS of the entire application is difficult to monitor. There is no support for migrating one service and reconfiguring the rest of the application to use the migrated service, in case a provider does not respect its SLA.

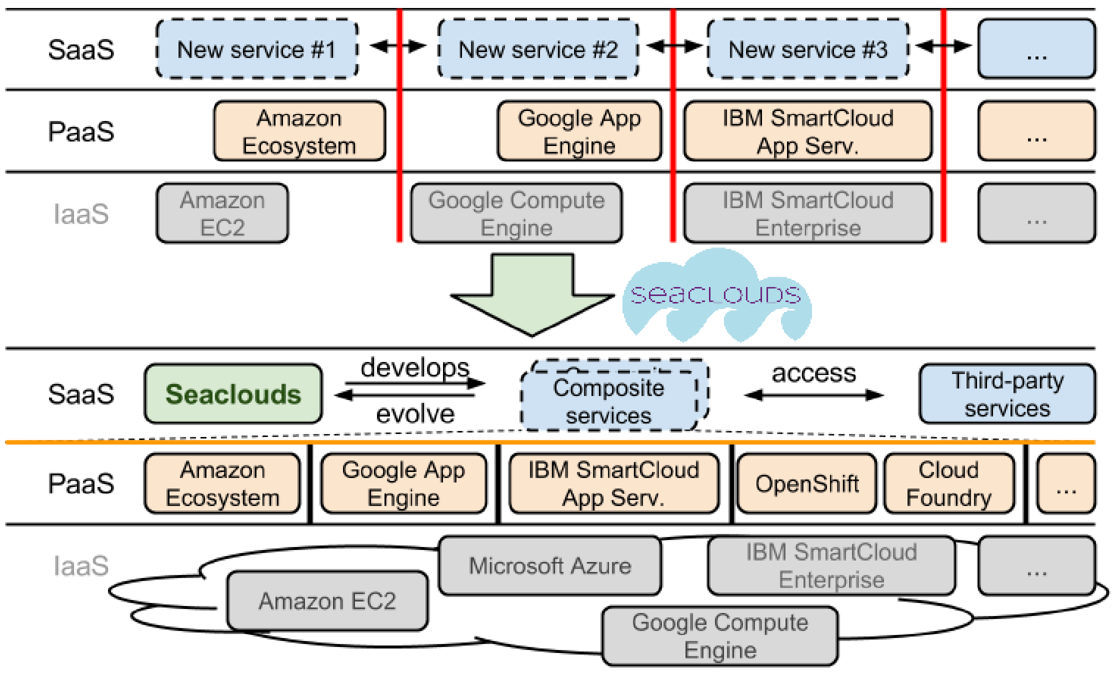


Figure 2. Cloud architecture before and after SeaClouds.

SeaClouds aims at homogenizing the management over different providers and at supporting the sound and scalable orchestration of services across them. Moreover, systems developed based on the SeaClouds approach will inherently support the evolution of their constituent services, so as to easily cope up with needed changes, even at runtime. The development, monitoring and reconfiguration via SeaClouds include a unified management service, where services can be deployed, replicated, and administered by means of standard harmonized APIs, such as the ones provided by the Cloud4SOA project and the one that can be produced based on the CAMP specification.

In the following, we list some of the current problems and barriers, related to the cloud that will be solved by the main results expected from SeaClouds.

1. ***Support for application deployment and migration to different providers***. SeaClouds will provide support for deploying and migrating applications composed of several services taking care of the synchronization of the services and their reconfiguration, without requiring the user to manually intervene.

2. ***Management and monitoring of underlying providers***. Properties over application and services deployed on multiple clouds can be ensured and managed in a standardized way by using unified metrics and automated auditing.

3. ***Increased availability and higher security***. The usage of formal models to support the management of service-based applications over multi-clouds environments gives more flexibility to reconfigure the distribution when a SLA violation occurs.

4. ***Performance and cost optimization***. The framework gives users freedom to distribute application requirements over different cloud offerings by using needed options in a flexible manner. Organizations can take advantage of useful and powerful services provided by each platform and avoiding its weaknesses. Optimization requirements can also be modelled to consider cost as the main decision parameter.

5. ***Low impact on the code and user-friendly interface***. SeaClouds will tackle different problems for developers and administrators of cloud applications thanks to the proposed orchestration model as follows. First, the development process is simplified by using the SeaClouds tools and framework that require minor code changes. Second the management of already deployed complex cloud applications is simplified thanks to the SeaClouds dashboard.

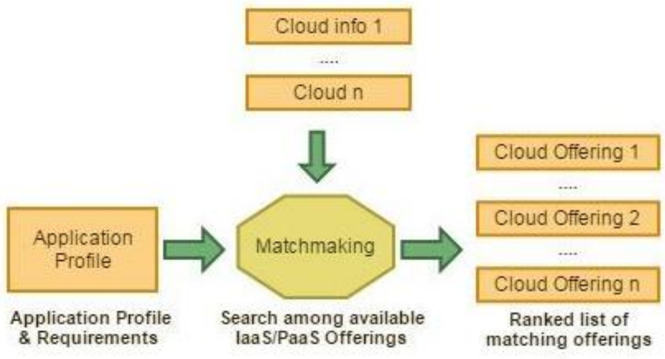
## 2.2. The SeaClouds Functionalities

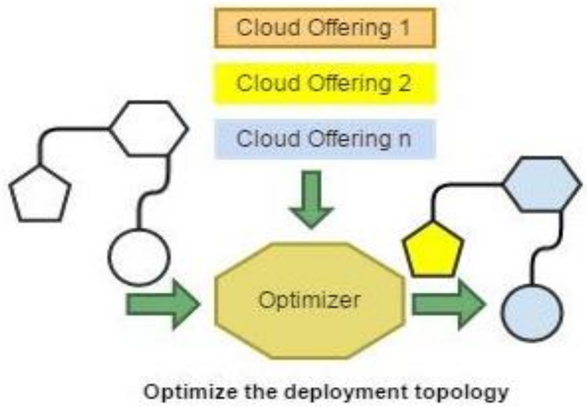
When we talk about SeaClouds, the basic question we should answer is: ***What is SeaClouds?***

- ***SeaClouds*** is a software platform which makes more efficient the "design, development, planning and management" of complex business applications distributed on multi-cloud environments.
- ***SeaClouds*** provides an enterprise capability with continuous software delivery that enables independent application vendors to mitigate risks and reduce time and cost to market and customer feedback.
- ***SeaClouds*** orchestrates services, platforms and infrastructures to ensure they directly and dynamically, meet the needs of cloud applications.
- ***SeaClouds*** provides an integrated and based on standards multi-cloud application management system that follow the DevOps approach with seven basic capabilities delivered to the developer via an innovative Graphical User Interface (GUI).

The seven basic functionalities/capabilities of SeaClouds are the following:

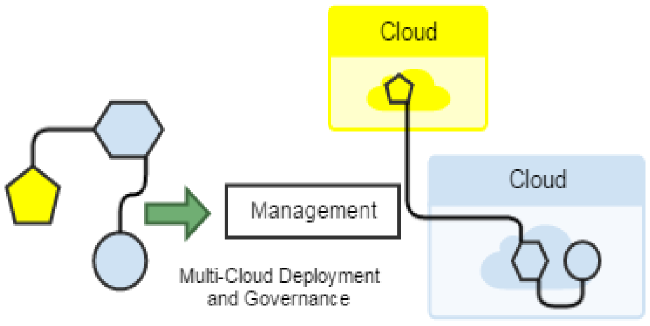
***Matchmaking*** allows querying or browsing available cloud offerings (PaaS and IaaS) via the SeaClouds GUI for determining suitable ones based on application requirements. The latter includes information about the technology of each application module and a suite of non-functional requirements such as: Location, Quality of Service, SLA, Cost, etc.



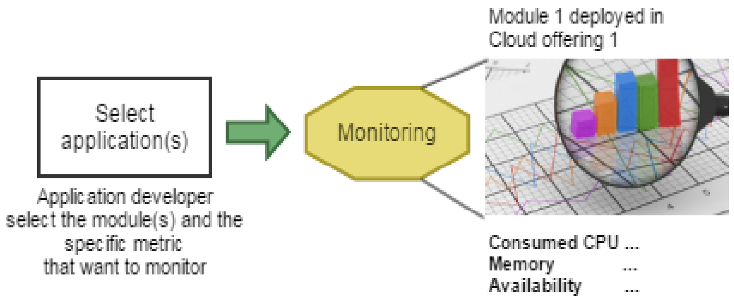


***Cloud Service Optimizer***: It optimizes the deployment topology of an application across multiple clouds to address non-functional requirements by following strategies like the maximization of the Cost or the Quality of Service and possibly following location policies like "following the sun".

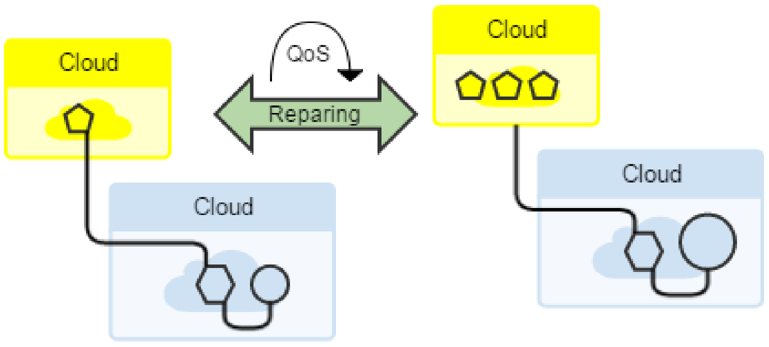
***Application Management (Application Deployment and Governance)***. It supports efficient deployment and multi-cloud governance of a complex application on various cloud offerings (IaaS and PaaS) leveraging cloud harmonized APIs and platform-specific adapters.

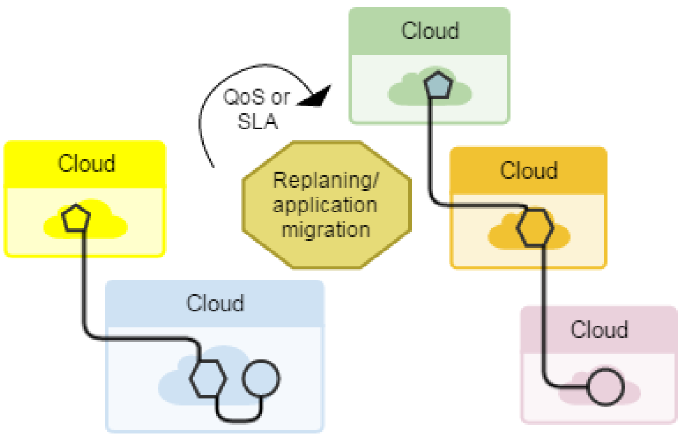


***Monitoring and SLA enforcement***. It provides an independent Cloud monitoring functionality based on unified "cloud offering" independent metrics, such as latency/Response time, application status, etc., to allow operators to proactively monitor the health and the performance of business-critical applications, hosted across multiple Clouds.


Select application(s)
Application developer select the module(s) and the specific metric that want to monitor
Monitoring
Module 1 deployed in Cloud offering 1
Consumed CPU ...
Memory ...
Availability ...

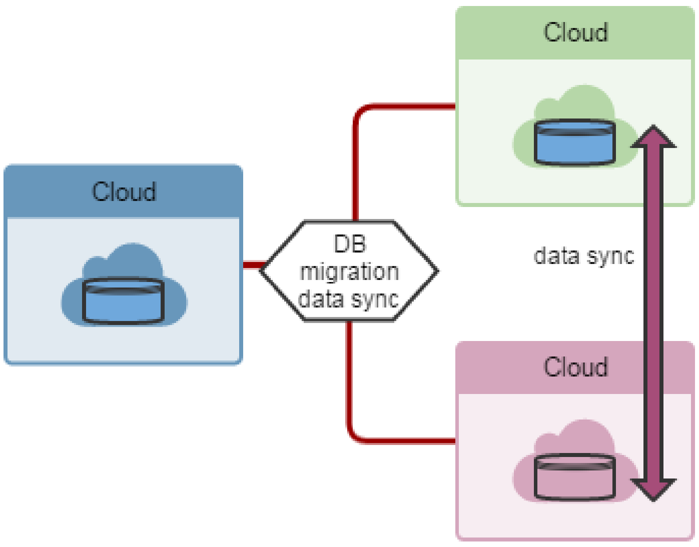
***Repairing*** (cloud resource scaling). It scales horizontally and vertically cloud resources to maximize the performance of each module of an application; it restarts and replaces failed components.





***Replanning and Application migration***: The SeaClouds software platform aims to support a seamless migration of the software modules of an application between dissimilar (but compatible) clouds, dealing with the semantic interoperability conflicts to allow the full portability of an application.

***Database migration and data synchronization:*** It enables the portability of data between Databases (on different Clouds) in an automatic way to minimize the amount of effort and time for application developers operational teams to set up and process critical data updates between different and in some cases dissimilar databases. It ensures that data are not lost nor damaged in the process and that the application exploiting the data is not negatively affected by this movement.



The aforementioned functionalities are provided by the following software components, introduced in the next subsection, in which the SeaClouds platform is presented. The details about the architecture of the SeaClouds platform are given in Section 3.

- The ***Matchmaking*** functionality is implemented by the Planner component.
- The ***Cloud Service Optimizer*** functionality is implemented by the Planner component as well.
- The ***Application Management*** functionality is implemented by the Deployer component.

- The ***Monitoring and SLA enforcement*** functionality is implemented jointly by the Deployer, Monitoring and SLA Service components.

- The ***Repairing*** functionality is implemented jointly by the Deployer and Monitoring and components.

- The ***Replanning and Application migration*** functionality is implemented jointly by the Deployer and Monitoring and Planner components.

- The ***Database migration and data synchronization*** functionality is inherited from MODAClouds and is integrated with the other elements of SeaClouds so that they can activate it.

## 2.3. The SeaClouds Architecture

This section presents the reference architecture and the design of the SeaClouds platform, and discusses its novel aspects compared to other existing initiatives and efforts. Figure 3 depicts the reference architecture of the SeaClouds platform.

Before describing the core components of the architecture of the SeaClouds platform, it is worth observing that the platform features a **Graphical User Interface (GUI)** for two user roles (Designers and Deployment Managers), and that Cloud Provider Systems are considered. The main **stakeholders for the SeaClouds platform** are the following:

- ***Application Designer*** (or ***Developer***) exploits the GUI to provide a description of the topology of the application to be deployed together with a set of requirements. These requirements can include QoS properties and technology requirements for the application modules, and the maximum acceptable cost for the entire deployment.

- ***Deployment Manager** (or **Application Manager**)* exploits the GUI through the **Unified Dashboard** that allows them to supervise the deployment and the monitoring of the application.

- ***Cloud Providers*** offer Cloud Resources (which further offer some Cloud Capabilities). They do not necessarily interact directly with the SeaClouds platform, but the services offered are exploited by the platform to run service compositions.

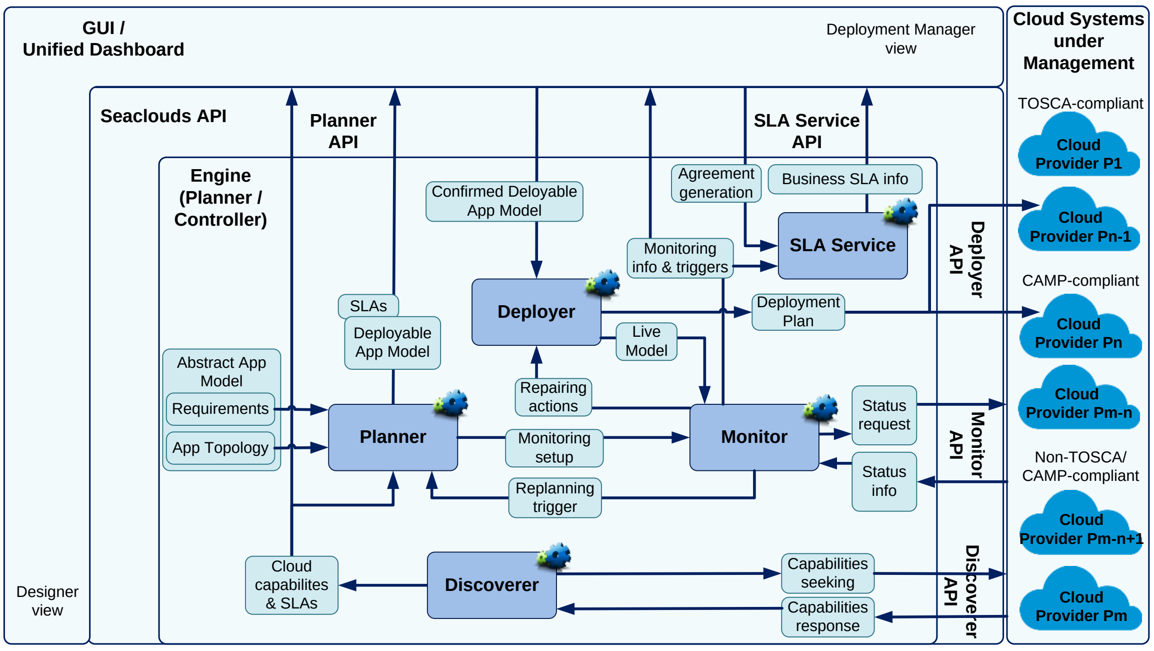


Figure 3. Initial Architecture of the SeaClouds Platform.

From **SeaClouds platform functionalities** standpoint, we can identify five major components in the architecture, plus a RESTful **harmonized and unified SeaClouds API** layer used for the deployment, management and monitoring of simple cloud-based applications through different and heterogeneous cloud providers.

- ***SeaClouds Discoverer***: it's in charge of discovering available capabilities and add-ons offered by available cloud providers using the Discoverer API.
- ***SeaClouds Planner:*** it's in charge of generating the DAM, using the Planner API.
- ***SeaClouds Deployer***: it's in charge of executing the DAM, using the Deployer API.
- ***SeaClouds Monitor***: it's 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, using the Monitor API. It is also in charge of generating the reconfiguration suggestions (if needed) to be passed as inputs to the Planner component to trigger the generation of a new adaptive orchestration plan.
- ***SeaClouds SLA Service***: it's 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.

SeaClouds follows the Application Model Lifecycle depicted in Figure 4.

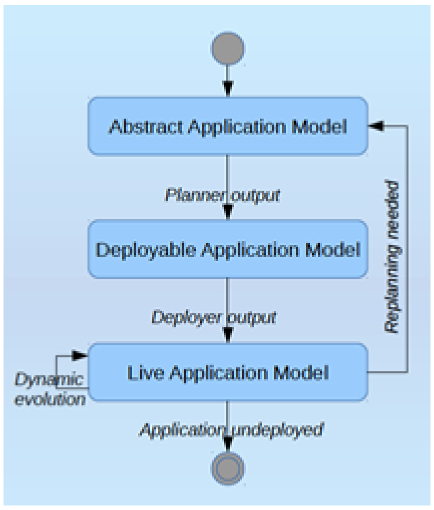


Figure 4. Application Model Lifecycle.

Figure 5 represents the necessary steps to carry out an application deployment from the initial stage where the Application Developer (end-user) provides the Application Model consisting of the Module Profile and the Topology representing the connections among the modules of the cloud application to be deployed (other elements as the SLA restrictions and policies are considered by SeaClouds), as described in detail in Deliverable D3.1 [2], related to the design-time, to the deployment, monitoring and reconfiguration of the application, related to the run-time. A brief explanation of this process is given in the following.

After the Abstract Application Model (AAM) has been specified (step 1 in Figure 5), SeaClouds kicks off the Discovering and Planning stage. Once the cloud providers have been discovered (step 2), the Planner acts with two subprocesses: Matchmaking and Optimizer (described in D3.1 [2]).

The Planner generates a Deployable Application Model (DAM), which specifies the concrete cloud services used to distribute the application (step 3). The Deployer component, detailed in Deliverable D4.1 [3], executes the confirmed DAM (step 4), while the monitoring is configured with the corresponding monitoring rules taken from the user requirements (step 5) and the agreements are initialized with the user inputs (step 6). Also, the SLA service is subscribed to rules or alerts in connection with the Monitor, enforcing the policies of the agreements (step 7). The Deployer allows the deployment of the application's modules over heterogeneous IaaS and PaaS (step 8), and a Live Application Model, which tracks the dynamic evolution of the deployment and management of the application modules themselves.

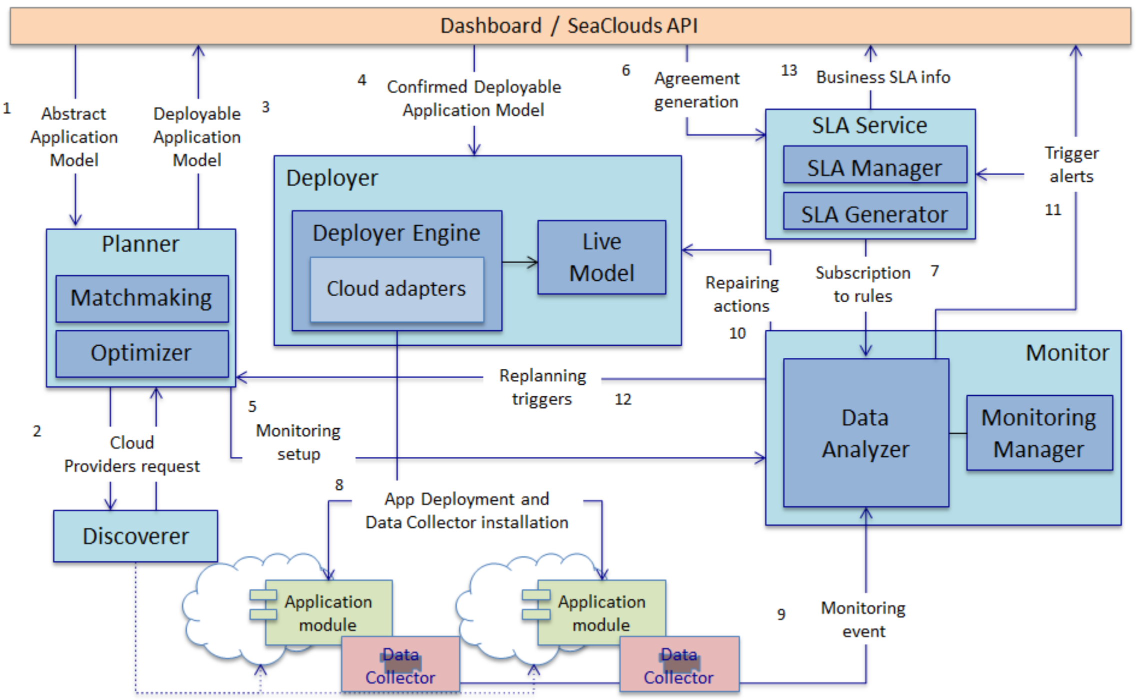


Figure 5. Interaction flow between the SeaClouds components.

Once the application is deployed, the Deployer manages it and instruments the Monitor. In particular, the Deployer installs DataCollectors in the cloud machine(s), in which the application modules have been deployed. A DataCollector component gathers raw monitoring data and pushes them to the Monitor (step 9). The latter component interacts with the SLA service to manage violations of properties, QoS and QoB. A Live Application Model maintains a track of the dynamic evolution of the deployment and management of the application modules. Whether a violation issue occurs, and it can be fixed without replanning, then the Monitor and the Deployer interact to repair the issue (step 10). Otherwise, then the Monitor interacts with the Planner (steps 11, 12). In this case, the Planner generates a new plan, which will be executed by the Deployer. Business SLA info is exposed by means of the Dashboard (step 13).

A distinguishing aspect of the SeaClouds architecture is that it builds on top of two OASIS standards initiatives: **TOSCA** (Orchestration Specification for Cloud Applications) [4] and **CAMP** (Cloud Application Management for Platforms) [5]. On one hand, adopting TOSCA to represent application topologies can be automatically processed by any TOSCA-compliant platform. On the other hand, adopting CAMP, which proposes standardized artifacts and APIs that need to be offered by PaaS clouds to manage the building, running, administration, monitoring and patching of applications in the cloud. It is however worth noting that the Deployer does not require cloud providers to be TOSCA or CAMP compliant, and it actually could generate deployment plans for non-TOSCA/CAMP compliant providers as needed.

Section 3 describes more in detail the components and services, their functionalities, interactions, and the inputs/outputs of the SeaClouds platform, but before we illustrate the SeaClouds proposal with a storyboard using the NURO Cloud Gaming case study.

## 2.4. The SeaClouds StoryBoard

Let’s imagine Christian is a video game developer and he wants to use the SeaClouds platform to be able to configure, deploy and manage his video games on multiple and heterogeneous clouds.

Christian is dealing with applications constituted of a multiple modules: a client, typically installed on the customer’s device, and a server (software layer plus database). Christian needs to find a good balance between availability, response time and cost of the server part. Moreover, he needs to guarantee that the server is able to handle a workload with significant variations over time, both in terms of number of requests per second (the number of players can grow/decrease depending on the marketing campaigns the company organizes, the exhibitions in which the video game is shown, the time of the day,…) and in terms of geographic provenance (video games played in US and India will experience a workload mainly originated by USA in the timeframe 18.00-22.00 EST and mainly originated by India in the timeframe 8.30-12.30 EST, corresponding to 18.00-22.00 IST). The cost of hot instances on clouds may vary with the time of the day, Christian realizes that it is convenient for his company to exploit the cloud that offers the best price (and other features) at each time. Overall, Christian’s requirements are met if careful configuration, deployment, and management of the system are performed. This is the reason for which Christian decides to use SeaClouds.

Christian has already developed the client and server for his application. Moreover, he has set up a MySQL database for storing the application data. The application (written in PHP) interacts with the database connecting to MySQL compatible databases.

From the SeaClouds **Dashboard** and using the **GUI**, Christian starts defining the **Abstract Application Model** (AAM). The Dashboard assists him in:

- Identifying the Base Modules Game Application and Analytics Application that are PHP components;
- Identifying the containers to be used to run them, the PHP workers offered as services by PaaS or IaaS providers;
- Mapping the application databases, Game DB and Log DB to the same MySQL service;
- Defining as first Quality of Service requirement Fast Response, and associate to it two Management Policies called BOOM and Burst, respectively, that will be activated as soon as the monitoring system realizes that the Fast Response requirement is violated;

Christian will use the same Dashboard to follow the application lifecycle.

At this point, SeaClouds, through its **Discoverer**, checks a list of different cloud providers. For each provider, SeaClouds Discoverer:

- gathers technical properties of its cloud - in order to satisfy Christian's application technical requirements;
- checks their exposed availability and other issues (like public prices);
- tests the performance of different alternatives offered by the provider (SeaClouds gets help from CloudHarmony[1] for this task);

Using the Abstract Application Model that Christian created and the gathered information from cloud providers, SeaClouds invokes the execution of its **Planner**. The Planner will transform the AAM into a **Deployable Application Model** (**DAM**). For doing this, it passes through two stages.

In the first stage, some existing cloud offers are discarded for the execution of modules in the Abstract Application Model. The decision to discard some cloud offers is performed by a **Matchmaker** and is based on the matching between technology/technical properties they offered and technology/technical requirements expressed in the Abstract Application Model. Therefore, in this stage, for each application module in the Abstract Application Model there are only kept the cloud offers that are technically able to execute it.

In the second stage of the planning activity, it is performed an **optimization** process in which, for each module in the AAM, it is selected a cloud offer among the ones that can technically execute it. This optimization process is guided by the quality requirements stated in the AAM (i.e., the required availability, performance and cost of Christian's application). Once a cloud offer is selected for each module, they are saved as Concrete Service object in the AAM.

Using performance evaluation techniques, the optimizer now deals with the generation of information useful for the runtime scalability/elasticity decisions. It does so by computing the threshold values for system characteristics (e.g., monitored system response time) above/below which application modules should scale up/down. It also provides the maximum limit of replicas of each module that the application can afford without budget overrun (i.e. not exceeding the application cost requirement). At this point the Planner work is finished. The result of this step, is a specification of the Christian's application compatible with major standards on cloud interoperability, which the advantages of being able to deploy the application in multiple clouds.

This optimized Abstract Application Model together with the information regarding when to scale up/down and the scaling limit conforms the DAM.

As soon as the DAM is ready, Christian selects the deploy option and waits until the application is deployed and ready to operate, which is done by the SeaClouds

[1] https://cloudharmony.com/

**Deployer**. Then it starts the application execution and makes it available to his customers.

By means of the **Monitor**, the SeaClouds monitoring system continuously oversees the execution of the system and at a certain point realizes that the Fast Response requirement is violated.

Christian is satisfied of the way the system works, but he knows that in the current stage still it does not fulfill his actual requirement of being able to achieve a good balance between availability, response time and cost.

Christian then opens back the Application Model editor and introduces the new requirement about the balance plus the information about the system workload that he has collected during the initial operation of the system. The SeaClouds Planner analyses the new requirement and comes us with a variation of the DAM containing the following **reconfiguration** action:

```
if (timeOfTheDay >= 9.00 EST AND timeOfTheDay <= 13.00 EST)
        migrate system to IndianCloud
```

He then asks SeaClouds to put the system back in operation.  In terms of **SLA Service**, from the point of view of the application, Christian wants to offer some guarantee to his users that the platform is responsive enough to offer a nice playing experience. So he adds a business SLA constraint to the agreement that represents the service offered by Christian to the players: if the runtime of the platform is greater than 2 seconds, he offers discounts for in-app purchases to the users that are playing at that time.

The cloud providers where each module is deployed offer their own fixed SLA. But Christian wants to consult the SLA of the current providers in a centralized way and in a uniform format. So, he accesses the dashboard where he can consult in a YAML format the SLA of the providers. Moreover, he wants to be notified when SeaClouds evaluates that the provider SLA has not being fulfilled. For example, HP Cloud guarantees a monthly uptime of 99.95%, with variable discounts depending of the actual uptime. Christian wants to check if the discounts are applied at the end of the month by the provider.

Christian also could indicate some actions previously defined to repair situations that could be solved by using techniques to repair without need of replanning (create a new plan), and also he could request these actions at run-time, by means of the dashboard.

## 3. SeaClouds Components

In this section, the components from the final architecture of SeaClouds are detailed. It is worth mentioning each component offers its functionalities through a REST interface and the interactions between modules must be performed using these REST interfaces. A first design of the API of the modules was provided in Deliverable D4.2 [6], and the next and final version will be explained deeply in Deliverable 4.5 [7].

### 3.1. Discoverer Component

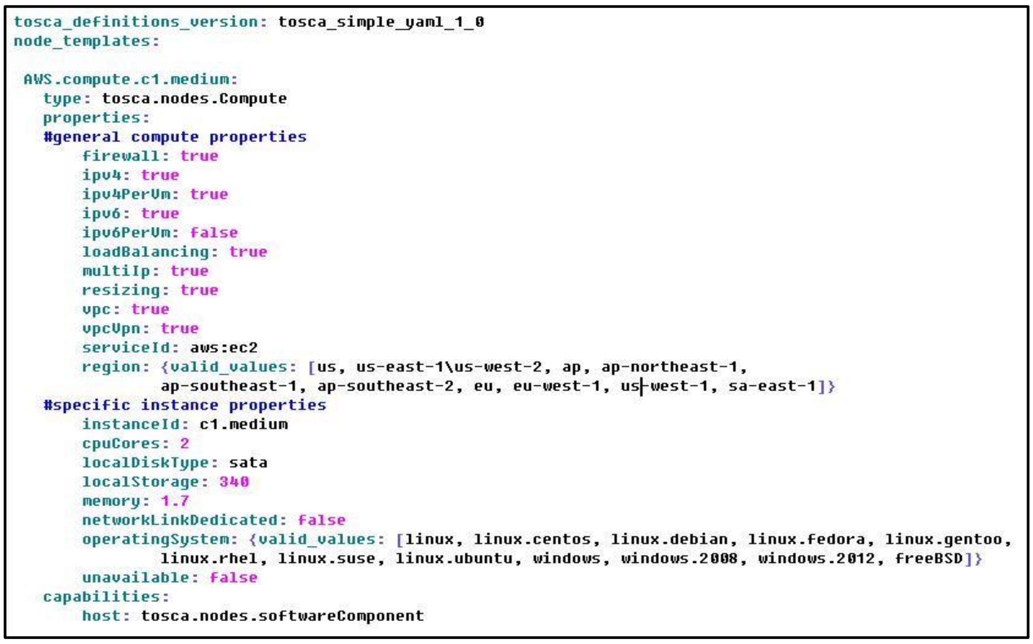
The main functionality of the Discoverer component is described in Table 2.

| **Component** | ***Discoverer*** |
|---|---|
| **Description/ Functionality** | The ***Discoverer*** component is in charge of discovering available capabilities offered by cloud providers. The description of such capabilities includes technology aspects (e.g., programming languages, development frameworks, runtime environments, add-ons), QoS properties (e.g., availability, reliability), along with the associated SLAs (including the cost associated to each provided service). The Discoverer will utilize some existing directories (e.g., CloudHarmony) to collect cloud providers and services, transform them into the forms based on the related model defined in SeaClouds, and then build the repository, which is accessible to the Planner component as well as the Dashboard. |
| **Inputs** | A list of cloud providers and a set of desired SLAs. |
| **Outputs** | A repository of available cloud provider capabilities (specified in TOSCA YAML) and cloud providers SLAs to be sent to the Planner and the Dashboard. |
| **Interactions and Interfaces** | This component interacts with the Planner and the Dashboard. The Planner will access the repository generated by the Discoverer to perform a matchmaking and optimization process so as to decide where to deploy each application module according to its QoS and technology requirements. The GUI/Dashboard will present the result of the discoverer to the end-user. Optionally, this |

| | |
|---|---|
| | component could also receive automatic updates from cloud providers. |

Table 2. Discoverer component description

The Discoverer (see the architecture and interaction with other components in Section 3.2, Figures 7 and 8, where is represented together with the Planner) is responsible for collecting the capabilities and services advertised by different cloud providers (both at IaaS and PaaS levels), generating the SeaClouds repository of cloud providers and services, and also maintaining and updating this repository continuously, hence to support the Matchmaking step in the Planner component to determine suitable cloud services from this repository for each module. In addition, we have proposed a general cloud profile model [2] to represent the services offered by different cloud providers so as to provide a uniform description of them, and mapped this model into TOSCA representations. Therefore, in the SeaClouds repository generated by the Discoverer, all the cloud services are written in TOSCA YAML format, which greatly facilitate our use in SeaClouds platform, e.g., a TOSCA-based matchmaking in the Planner component. Based on our cloud profile model and TOSCA YAML format, a simplified example for *AWS.compute.c1.medium* from Amazon Web Services in our repository is shown in Figure 6.



```
tosca_definitions_version: tosca_simple_yaml_1_0
node_templates:

 AWS.compute.c1.medium:
   type: tosca.nodes.Compute
   properties:
   #general compute properties
       firewall: true
       ipv4: true
       ipv4PerVm: true
       ipv6: true
       ipv6PerVm: false
       loadBalancing: true
       multiIp: true
       resizing: true
       vpc: true
       vpcVpn: true
       serviceId: aws:ec2
       region: {valid_values: [us, us-east-1\us-west-2, ap, ap-northeast-1,
               ap-southeast-1, ap-southeast-2, eu, eu-west-1, us-west-1, sa-east-1]}
   #specific instance properties
       instanceId: c1.medium
       cpuCores: 2
       localDiskType: sata
       localStorage: 340
       memory: 1.7
       networkLinkDedicated: false
       operatingSystem: {valid_values: [linux, linux.centos, linux.debian, linux.fedora, linux.gentoo,
               linux.rhel, linux.suse, linux.ubuntu, windows, windows.2008, windows.2012, freeBSD]}
       unavailable: false
   capabilities:
       host: tosca.nodes.softwareComponent
```

Figure 6. An example from Amazon Web Services

## 3.2. Planner Component

The Planner description and architecture is presented in Table 3. Planner component description and Figure 7, respectively.

| **Component** | ***Planner*** |
|---|---|
| **Description/ Functionality** | The ***Planner*** component receives from the application designer an Abstract Application Model (AAM) - a description of the application topology together with a set of requirements, including QoS properties and technology requirements - and it determines (by consulting the repository of cloud offerings) how the application modules can be distributed over the available clouds without violating the given set of requirements. The Planner is first triggered by the inputs from the application designer and then by replanning triggers generated by the Monitor component (possibly filtered by the Deployment Manager). The Planner will generate a Deployable Application Model (DAM) that specifies the concrete cloud services where to distribute the application modules. This is implemented in three steps (Matchmaking, Optimization, and DAM generator), and the outputted DAM will include the concrete services associated with each Base module and the policies to manage the scaling mechanism of each module. |
| **Inputs** | Abstract Application Model (AAM), repository of cloud offerings and related SLAs, replanning trigger, and Live Model. |
| **Outputs** | Deployable Application Model (DAM) and related Monitoring Configuration (Monitoring rules, SLAs…). |

| | |
|---|---|
| **Interactions and Interfaces** | The Planner receives the AAM including requirements and application topology from the Application Designer, by means of the GUI/Dashboard. It interacts with the Discoverer component to acquire the available capabilities and SLAs. It interacts with the SLA Service to generate the SLA agreements that will evaluate at runtime the Quality of Business of the application. It generates the DAM (returned to the Deployment Manager), so it connects also with the Deployer. It setups the Monitor from where it receives replanning triggers from the Monitor component, and also from the Deployment Manager (connected through the GUI/Dashboard). When a replanning is needed, it will ask the Deployer for the Live Model so as to collect enough information to do the replanning. |

Table 3. Planner component description

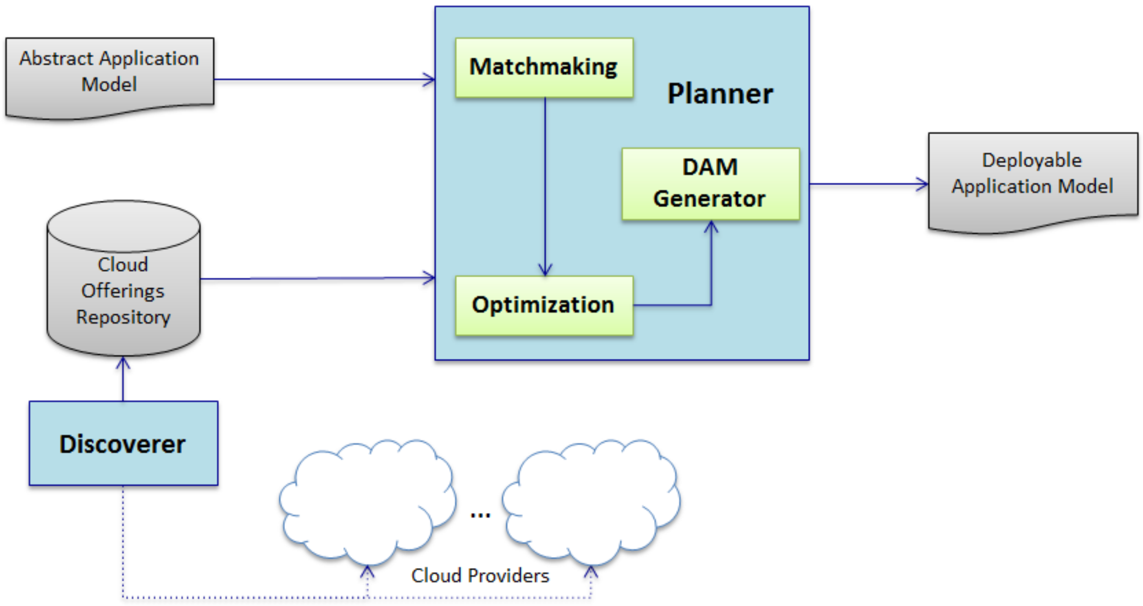

Abstract Application Model
Matchmaking
Planner
DAM Generator
Deployable Application Model
Cloud Offerings Repository
Optimization
Discoverer
...
Cloud Providers

Figure 7. Architecture of the Discoverer and Planner components

As depicted in Figure 7, the Discoverer is a single component connected to the Cloud Providers and manages the Cloud Offerings Repository, while the Planner consists of three sub-components:

- Matchmaking: this first step aims to identify, for each module, a set of cloud services that satisfy the technology requirements specified by the user. In this step, the matchmaking does not take into account quality requirements and possible relationships among modules.

- Optimization: once a set of suitable cloud services have been identified for each module, an optimization process can be performed. This activity decides, among all the options that satisfy the technology requirements provided by the matchmaking activity, a suitable combination of cloud services that allows the application to satisfy its quality requirements and the relationships among modules, at the same time that considers the utilization of cloud elasticity.
- DAM generator: following the matchmaking and optimization, this step will finalize the DAM through adding the policies like “Following-the-sun”, so as to include enough information for the Deployer.

The Planner is responsible for transforming this AAM into a DAM (Deployable Application Model) so that the Deployer can deploy the application actually. This transformation will be completed by three steps in the Planner.

First, for each module of the application, the Matchmaker will check the repository and do a matchmaking according to the technology requirements expressed by the user in the AAM, and discard the cloud offerings that cannot satisfy the technology requirements. Therefore, after this step, for each application module in the AAM, only the cloud offerings that are technically able to execute it are selected.

Second, based on the results from the Matchmaking step, the Optimization step will try to find a suitable combination of cloud offerings that satisfy the quality requirements of the application and/or its modules. This process is guided by the quality requirements specified by the user in the application (i.e., the required availability, performance and cost), and also considers the utilization of cloud elasticity. By using performance evaluation techniques, the optimizer deals with the generation of information that is useful for the runtime scalability/elasticity decisions. It does so by computing the threshold values for system characteristics (e.g., monitored system response time) above/below which application modules should scale up/down. It also provides the maximum limit of replicas of each module that the application can afford without budget overrun (i.e., not exceeding the application cost requirement).

Third, based on the Application Model generated by modifying the original AAM by Matchmaking and Optimization, the DAM generator will add the needed policies like “Follow-the-sun” or “autoscaling” into the model so as to generate the final DAM.

Figure 8 depicts the interaction of the Discoverer and Planner components with the rest of components of the SeaClouds Platform.

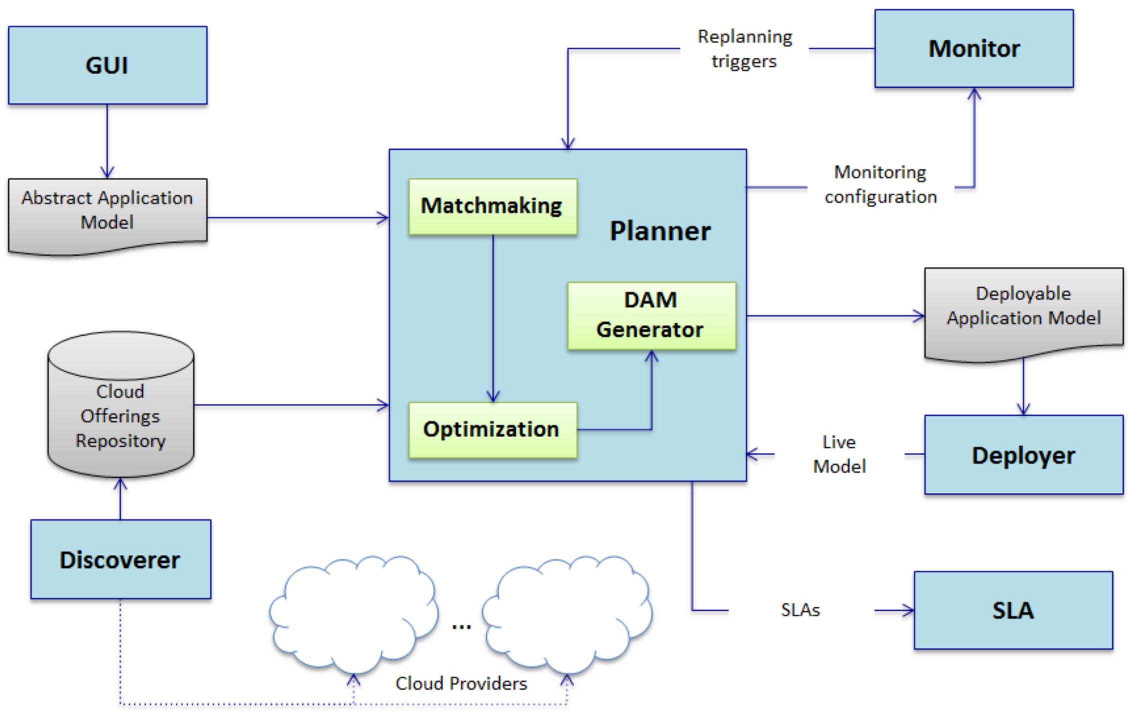


Figure 8. Discoverer and Planner strategy: interaction between components

## 3.3. Deployer Component

Table 4 describes the main functionality of the Deployer component, whose architecture is presented in Figure 9.

| **Component** | ***Deployer*** |
|---|---|
| **Description/ Functionality** | 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, the relationships, the dependencies, features, constraints, the target providers, etc. Therefore, the DAM includes an embedded deployment plan for carrying out the deployment based on the modules dependencies and relations.<br><br>The Deployer processes the DAM and uses the necessary operations that allow to manage the target locations and the cloud resources in a homogeneous way. Then, the |

| | |
|---|---|
| | application components are distributed and the relation are established, archiving the desired application behavior.<br><br>The Deployer provides the necessary operations or methods, through the REST API, allowing the post-deployment management like changing the application status (e.g., stop, pause and restart) or performing entity-specific actions (e.g., scale up/down).<br><br>Moreover, the Deployer maintains the model of the current application deployment status called Live Application Model (LAM). |
| **Inputs** | Deployable Application Model (DAM) |
| **Outputs** | It could be considered as the output of the Deployer the deployment itself, and internally is maintained the Live Application Model (LAM) |
| **Interactions and Interfaces** | The Deployer receives the DAM from the Planner, and by means of the GUI/Dashboard confirms the DAM. It interacts with the target platforms, cloud providers, using the needed services to deploy the application modules. The Deployer is directly connected with the Monitor to check when a repairing action is required and giving information to the Monitor about the Live Application Model. This component also provides a REST API, as it was mentioned above, to ensure the post-deployment management of application components and cloud resources. |

Table 4. Deployer component description

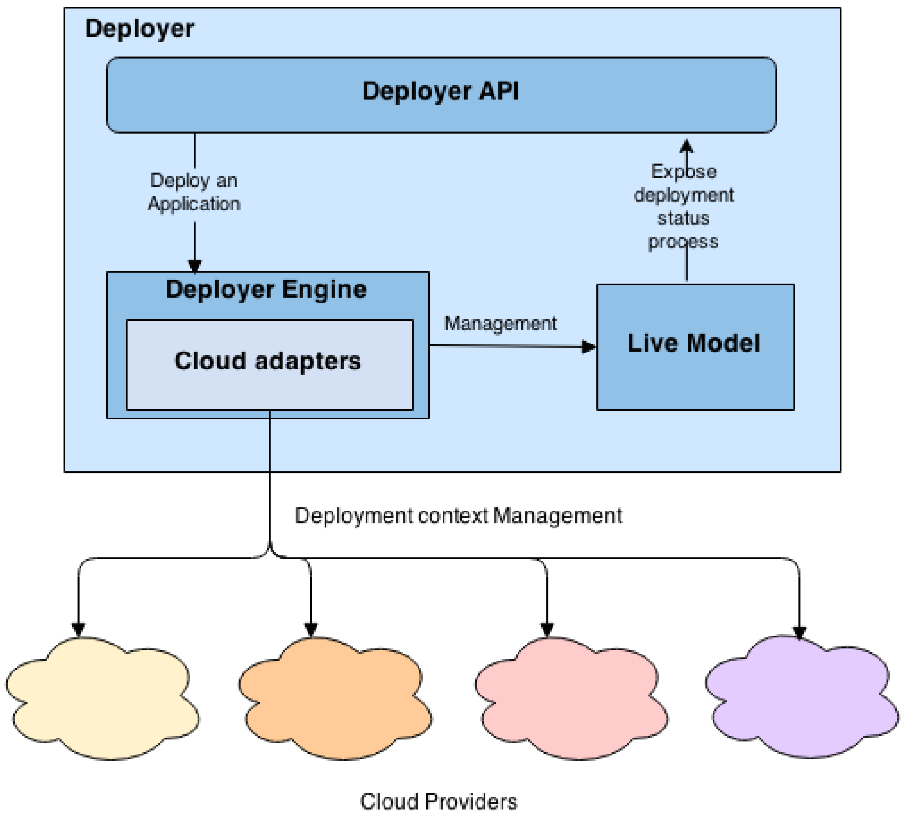


Figure 9. Architecture of the Deployer component

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

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. 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. Whether a violation occur, then a reconfiguration process is required. This process is detailed in Deliverable 4.3 [11].

Figure 10 shows the steps performed during the deployment of a cloud application using the SeaClouds Deployer, considering the interaction with the rest of components (the full description can be found in [3]).

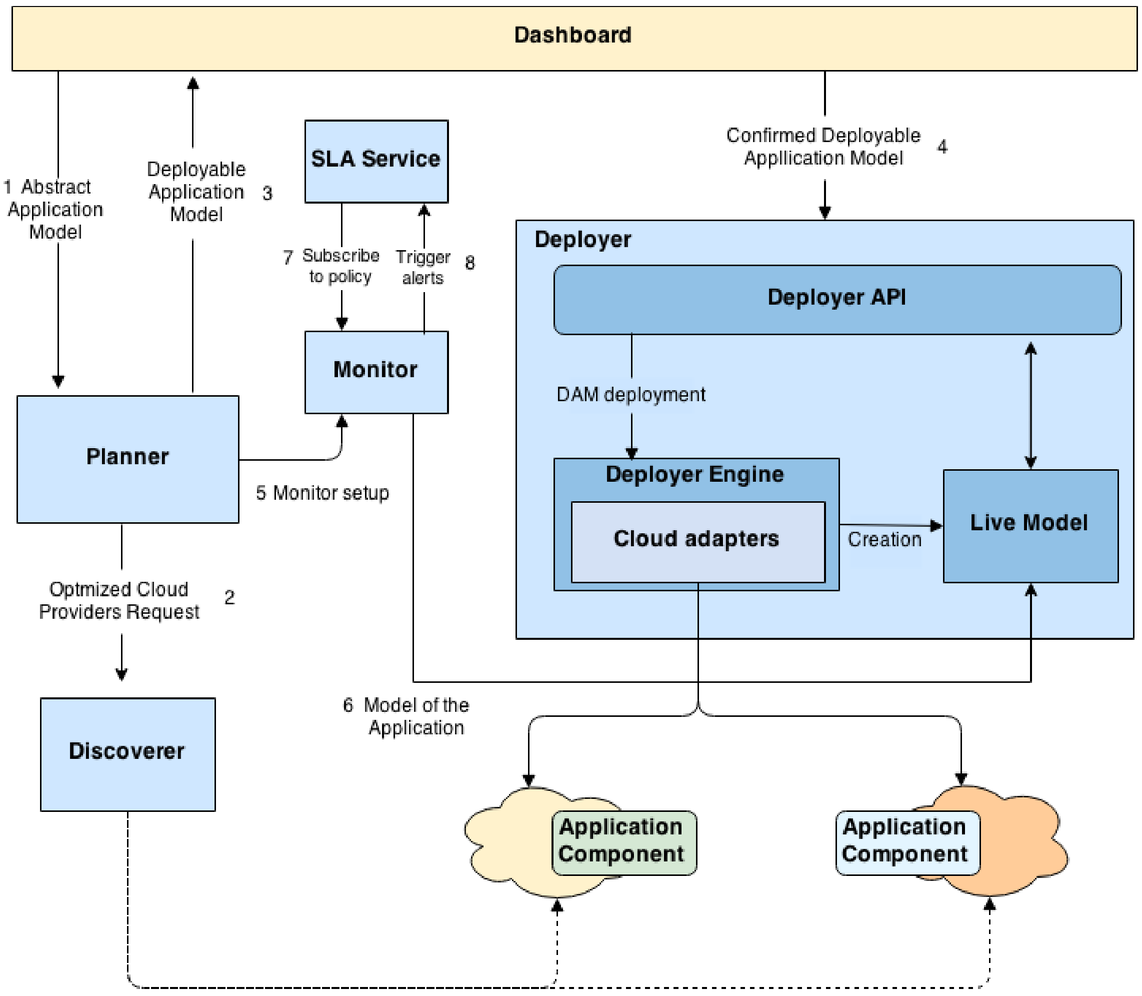


Figure 10. Deployment strategy: interaction between components

## 3.4. Monitor Component

In Table 5 and Figure 11 are presented the functionality and the architecture, respectively, of the Monitor component.

| **Component** | ***Monitor*** |
|---|---|
| **Description/ Functionality** | The ***Monitor*** component receives the set 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.<br><br>The monitoring component offers a REST interface to install monitoring rules and configure the monitoring subcomponents, in particular, the data collectors. Moreover, through the same REST interface, the monitoring component allows other components to register themselves as *observers* to the stream of data corresponding to some monitoring rule. The Monitoring component exploits the MODAClouds monitoring platform as its core element. |
| **Inputs** | 1. A set of monitoring rules<br>2. The live model. This should be described according to the monitoring meta-model and should be kept updated through the provided API at runtime. |
| **Outputs** | The monitoring component offers the possibility to connect external observers to its data streams corresponding to some monitoring rules. This way, such observers can be aware of the monitored metrics and trigger repairing or replanning actions when some of the monitoring rules are violated. |
| **Interactions and Interfaces** | The monitoring platform offers its services through a REST interface. Moreover, it assumes that components that want to be alerted of the occurrence of some issue register themselves as observers of some monitoring rule.<br><br>The interaction with data collectors on one side and observers on the other occurs by sharing data streams with them. |

Table 5. Monitor component description

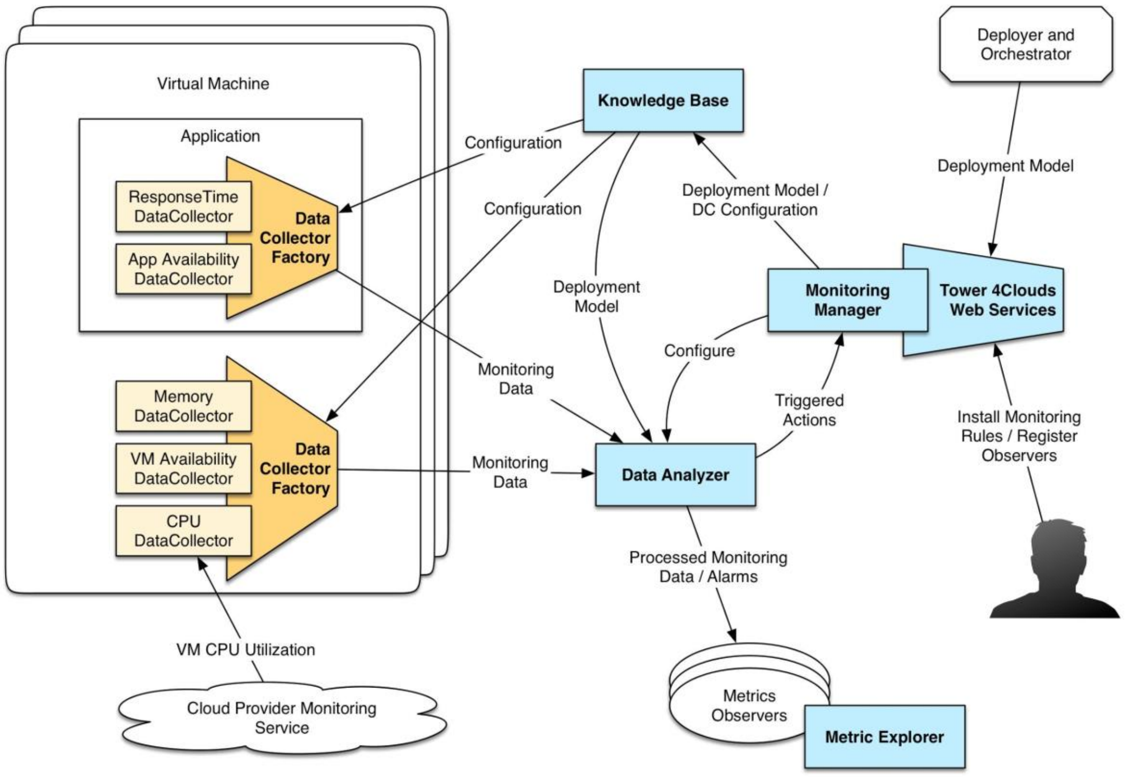


Figure 11. Architecture of the Monitor component

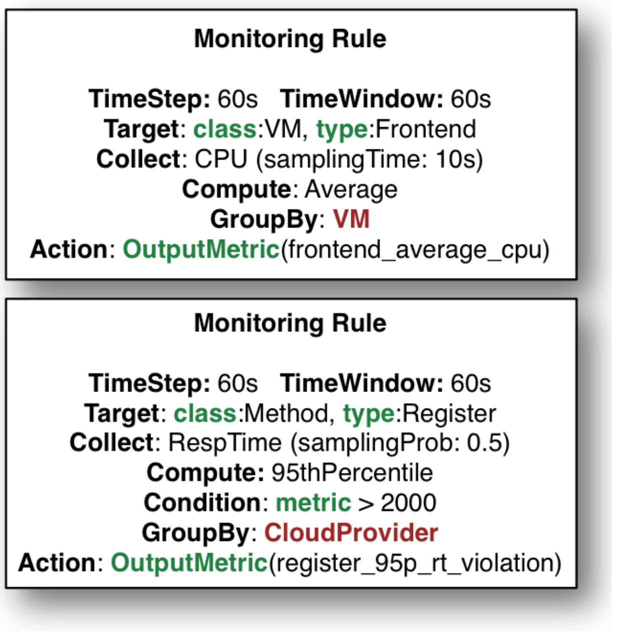
Figure 11 shows the architecture of the Monitoring component. The system is based on the idea of using Data Collectors to acquire monitoring data from various sources. In the figure we have as an example five different types of Data Collectors in charge of interacting with different components, either 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 starting from the Data Collector Factory library that is available here [12]. 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. KB (Knowledge Base) interface as in the KB, data collectors factories will retrieve the configuration of their data collectors. ObjS (Object Store) interface as from ObjS they retrieve the URL of the KB

All above interfaces are described in the MODAClouds deliverable D6.3.2 [13] and available on GitHub at the URLs indicated in the deliverable.

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 Ontology stored in the Knowledge Base. Thanks to this, it can aggregate and filter data at various levels of abstraction. For instance, in Figure 12, you can see the pseudocode of two rules.



**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 12. Monitoring rules pseudocode

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).

```xml
<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 13 XML code of a simple monitoring rule.

Figure 13 shows the full code of the first monitoring rule in Figure 12. MODAClouds deliverable D5.2.2 [14] provides more details on this language.

A component willing to observe an OutputMetric generated by the DDA is called Metric Observer. It should expose a REST endpoint and will receive monitoring data serialized as RDF JSON. A Java library that simplifies the creation of a REST endpoint and data de-serialization is available.

The Monitoring Manager is responsible for the correct configuration of the monitoring platform and abstracts all its complexity to the external users. It offers a REST interface for:

- Installing/uninstalling monitoring rules
- Attaching/detaching observers to metrics
- Updating the live model (described according to the monitoring meta-model) of the system being monitored.

Figure 14 shows an overall diagram of the interaction between the Monitor and the rest of components in SeaClouds. The interaction with the Monitor occurs through its REST interface. More specifically:

- The Dashboard registers as an observer for all monitoring rules generating events that should be shown to the user.
- The Deployer registers as an observer for all monitoring rules generating events that should trigger some repairing actions. Moreover, the Deployer passes to or updates into the Monitor the live model each time a new application is deployed or it is replanned (either through replanning or through repairing), respectively.
- The Planner registers as an observer for the monitoring rules generating the replanning event. This is generated by the monitoring platform upon reception of an event that is acquired by the Data Collector that monitors the application Live Model managed by the Deployer.
- The SLA Service registers as an observer for all monitoring rules generating events indicating an SLA violation.

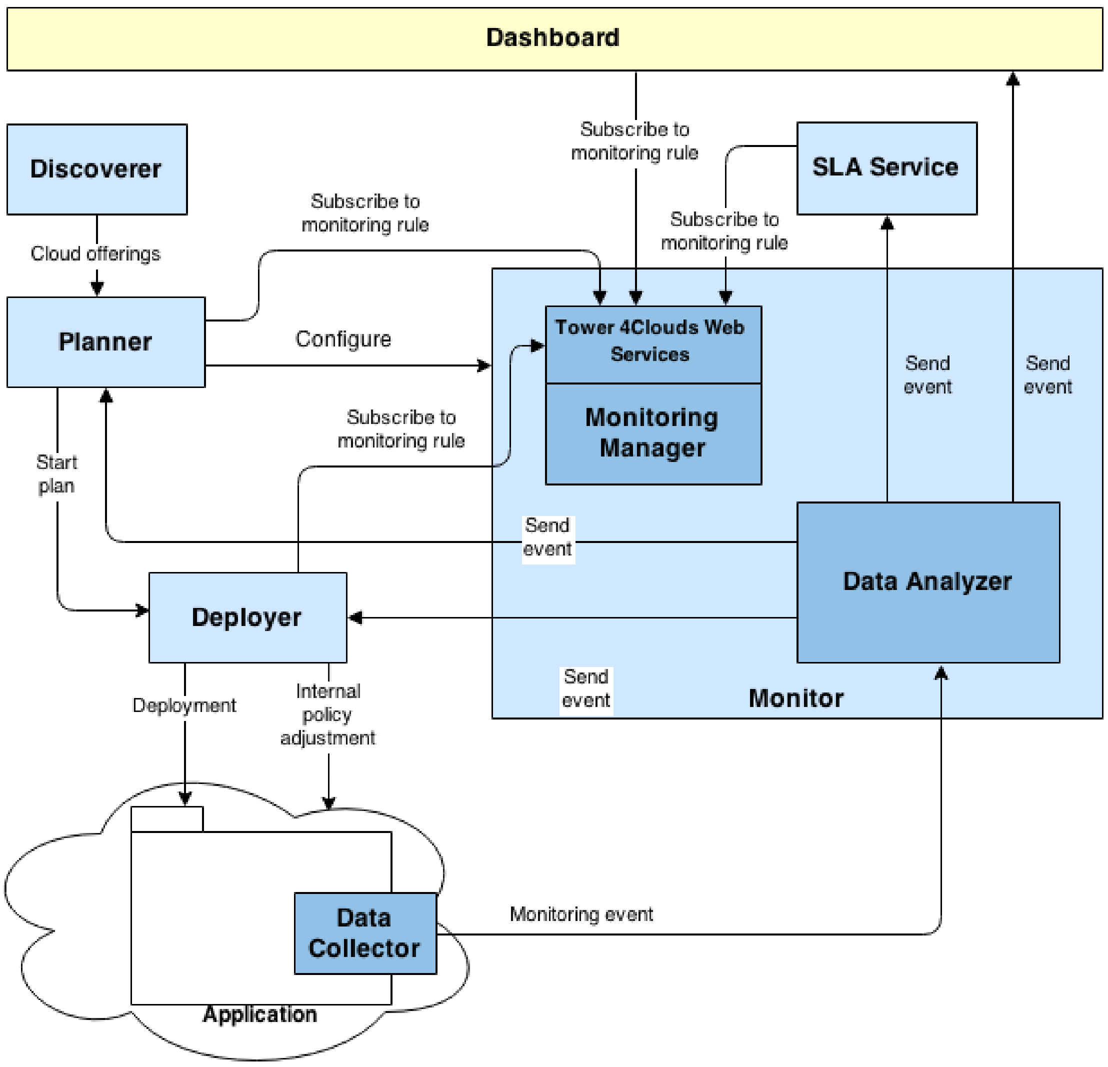


Figure 14. Monitoring strategy: interaction between components

## 3.5. SLA Service

As regards the SLA Service, its functionality and architecture are described in Table 6 and Figure 15 respectively.

| **Component** | ***SLA Service*** |
|---|---|
| **Description/ Functionality** | The SLA Service represents the component responsible for generating and storing the formal documents describing electronic agreements between the parties involved in SeaClouds: customers, application providers and cloud |

| | |
|---|---|
| | providers.<br><br>However, in the scope of the project, SLAs do not aim at representing a contractual relationship between the customers consuming a service and the vendors that provide them. SLAs describe the service that is delivered, the functional and non-functional properties of the resource, and the duties of each party involved.<br><br>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.<br><br>The SLA Service exposes the basic functionalities of handling providers, templates and agreements, and searching of violations and penalties, through a REST interface. It also offers the possibility to push events, such as violations and penalties, to subscribed components. |
| **Inputs** | In order to generate an agreement, the SLA Service uses the Deployable Application Model and the set of business rules defined by the application designer.<br><br>While the Planner component provides the inputs to create the SLA Agreements, the Deployer component will trigger the start of the SLA enforcement at runtime, once the application has been deployed.<br><br>In order to enforce an agreement, the SLA Service will be subscribed to the Monitor Component, which notifies about QoS violations. |

| | |
|---|---|
| **Outputs** | The component outputs:<br>● The agreement between an end user and the application provider, and the agreements between the application provider and each cloud provider where the application is going to be deployed. These agreements correspond to the two levels of SLA explained below,<br>● The notification of raised violations and penalties (including replanning triggers). |
| **Interactions and Interfaces** | The SLA Service offers a REST interface to interact with. The expected interactions are:<br>● GUI: consulting of agreements status, violations and penalties by the application administrator.<br>● Accounting/billing component: an external component may interact with SLA Service in order to obtain the business penalties that have occurred. |

Table 6. SLA Service description

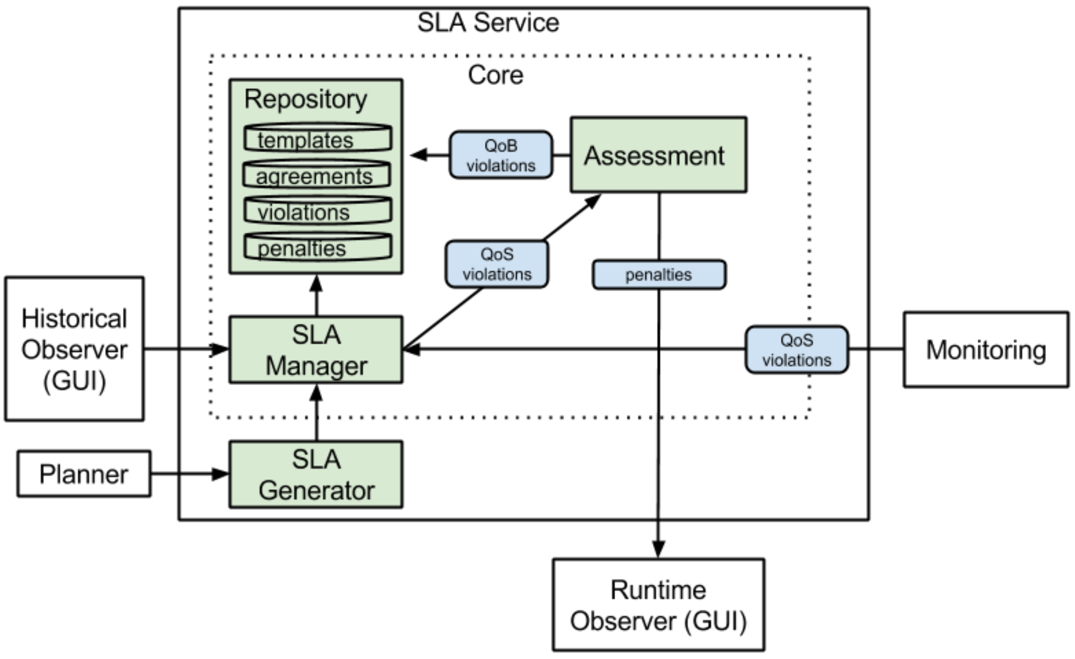


Figure 15. Architecture of the SLA Service

Figure 15 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 [15], which defines schemas for SLA Templates and SLA Agreements. A summary of the format of agreements and templates can be located at [16].

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.

A QoB rule is defined like this:

- A constraint over a metric provided by sensors (e.g. runtime < 2000ms). A non-fulfilled constraint is considered a breach.
- A time window where the number of breaches must be below a threshold (e.g. 5 breaches /day). If this time window is violated, a QoB violation is raised, and the business values take place (discount, migrations, downrating, other recovery actions…)
- The business action that takes place in the case of violation. This action may also be defined inside a time window. For example, 3 violations of a constraint

in a day are penalized with a discount, and 5 violations in a day are penalized with a trial of a service during one month.

In SeaClouds, we identify two levels of SLA, depending of the involved parties of the agreement:

- Customer (End user) - Application Provider SLA (C-AP SLA)
- Application Provider - Cloud Provider SLA (AP-CP SLA)

The Customer-Application Provider level describes the service offered by the Application Provider to their users. In this SLA level, there is one template per full application.

When generating the agreement from the template, the customer party in the agreement may be not an actual customer, but a generic one; this makes sense on applications that offer a free service to their users, not requiring a previous registration.

The guarantee terms in this C-AP SLA should only watch observable metrics by the end user. The purpose is to measure the Quality of Experience (QoE) that the user perceives when he/she is using the application. Examples of suitable metrics are: application request times, availabilities, continuity of video streaming... The application provider have the possibility to add business values to the guarantee terms, in order to describe the penalties that the application provider will incur if the term is violated. A possible penalty is a discount in the monthly fee.

One interesting fact about this level is that it will take the output of the Optimizer Module, where an optimum QoS for the application has been calculated, to generate this C-AP SLA.

The Application Provider-Cloud Provider level describes the service offered by the cloud provider to the Application Provider. For example, the Amazon offers are described in a set of templates, where the QoS assured and the corresponding penalties if not fulfilled are stored. These templates are filled with the SLA offers advertised by each cloud provider.

In this SLA level, there is one agreement for each cloud provider where the application is deployed.

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 cannot 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.

This SLA level performs a second assessment of the actual SLA enforced by the cloud provider, although monitored by the Monitor component. In this way, the application provider can check at the end of the billing cycle the QoS violations incurred by the cloud provider, and verify that the corresponding discounts have been applied.

In Figure 16 is shown an overview of the flow of interactions with the rest of components of SeaClouds.

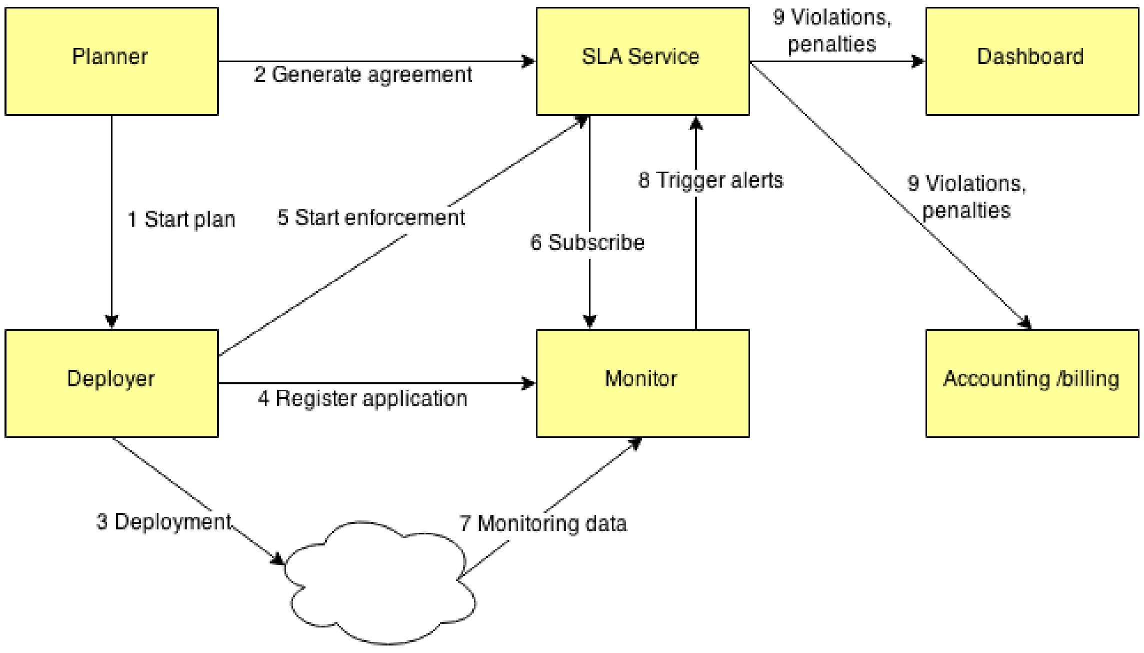


Figure 16. SLA Service strategy: interaction between components

## 3.6 GUI and Dashboard

The intention of the SeaClouds dashboard is to interact with all the SeaClouds components with an integrated and composed single view of the platform.

Table 7 details the main functionalities per each component UI.

| **Component** | ***GUI*** |
|---|---|
| **Description/ Functionality** | The GUI is focused on the implementation of an easy way to use topology editor. The GUI constitutes the uppermost layer in the SeaClouds architecture. |
| **Inputs** | The user input |

| | |
|---|---|
| **Outputs** | Abstract Application Model |
| **Interactions and Interfaces** | The GUI includes UIs for all the SeaClouds Components. |

Table 7. GUI description

In the SeaClouds GUI, the Discoverer UI allows the user to discover available capabilities offered by cloud providers, including technology aspects, QoS properties, SLAs. It also allows the use of existing directories to collect cloud providers and services.

The Planner UI, the application designer models the application topology, i.e. the modules that compose the application, the relationships between them, along with the desired QoS. As per the objectives of the Planner component, this UI allows the topology designer to specify a simple topology of the application.

After this step, the application administrator can specify the adaptation and business policies, and asks for an initial AMM. The Planner UI is a decision-support tool, and it will produce a number of feasible AMM that satisfy the application administrator requirements.

The application administrator has to select the AMM plan which fits better his expectations and needs. If none of them looks fine, he can change some/all of the requirements and generate a new set of suggestions.

Once the application is deployed, the application administrator can monitor the application performance.

The Deployer UI is used to verify the deployment of the application. Once deployed, the application administrator can check the status of all the modules that comprise the application. The administrator can also manage lifecycle of the application (start or stop a module).

The Monitor UI is where the application administrator can review the application performance, choosing to view real-time data and/or historical data of the monitored metrics.

The SLA Service UI is where the application administrator can review the behavior of the application in terms of violations of QoS and QoB, and the associated penalties. With this info, the administrator may decide to replan the application.

The Dashboard functionalities are defined in Table 8.

| **Component** | ***Dashboard*** |
|---|---|
| **Description/ Functionality** | The dashboard provides an easy way to interact with the SeaClouds Platform. It allows a final user to follow the entire application lifecycle (topology definition, discovering the best providers, deploying the application, monitoring the application metrics/SLA and triggering reconfigurations).<br><br>The Dashboard Component is a Web application which runs on any browser. It relies on the REST API to interact with the other components. |
| **Inputs** | User inputs |
| **Outputs** | User triggered actions and the Abstract Application Model |
| **Interactions and Interfaces** | The Dashboard "consumes" the information provided by the SeaClouds components by using a REST API exposed on each component. |

Table 8. Dashboard description

The intention of the SeaClouds dashboard is to interact with all the tools that have been implemented in the platform, in order to facilitate the use of the software capabilities.

We try to move away from the typical full integrated management application, focusing on innovative aspects. To achieve this, there is a strong effort in developing a client side written in JavaScript that interacts with the needed modules consuming their REST interfaces, in order to obtain an easy to use and responsive application.

The development of the dashboard will produce a set of modules, each one interacting with a different SeaClouds capability: Application Topology Design, Management, SLA, Monitoring, etc. The implementation of these modules will prioritize the functionality, leaving the goal of obtaining a uniform look as a less important topic.

## 4. Mapping the SeaClouds functionalities to the Case Studies

SeaClouds provides an integrated and open source “DevOps” multi-cloud application management system with seven basic capabilities (listed in Table 9) that span the Design and Runtime Management of Complex Business Applications. SeaClouds will be delivered to application developers and operators via an innovative graphical user interface (GUI).

The next table summarizes the contribution we expect from the SeaClouds tools to the development of the ATOS Social-Networking and NURO Cloud Gaming Case Study.

The functionalities offered by the Case Studies are characterized by grades in a 1 to 5 scale (1 = less important to 5 = very important) concerning the level of the importance of the development of the functionalities.

| **SeaClouds basic Functionalities** | **ATOS Social-Networking Case Study** | **NURO Cloud Gaming Case Study** |
|---|---|---|
| ***Matchmaking:*** Querying and browsing for the best-fit cloud offering based on the application requirements. | **[Grade = 4]**<br>- Functional and non-functional requirements ***(Cost, Location, Response Time and Availability)***<br>- The QoS design / modelling (based on WS-Agreement) allows refining the design of the architecture of the application leveraging on real QoS requirements from the application provider and application customer point of view. | **[Grade = 3]**<br>- The SeaClouds Matchmaking considers some aspects for the NURO use case, such as, limited budgets, technical requirements, QoS properties |
| **Cloud Service Optimizer:** Optimizing the deployment of an application across Clouds to address functional requirements. | **[Grade = 2]**<br>- Application topology Optimization based on the maximization of the benefit for the Application provider in terms of Cost or following a QoS strategy | **[Grade = 4]**<br>- Performance optimization considering the application topology |
| ***Application Management (application Deployment and Governance):*** Streamlined “multi-cloud” management of applications across | **[Grade = 5]**<br>- The SeaClouds Runtime tools will allow reducing the operational overhead with multi-cloud application management providing unified API to govern the application (included the | **[Grade = 4]**<br>- Some issues that the SeaClouds Runtime environment can manage in the case study are: save system |

| | | |
|---|---|---|
| Clouds. | deployment and the un-deployment of the modules of a complex application) unified metrics across all deployed applications allowing comparing and contrasting between applications' performance across Clouds infrastructures | administrational effort, unified cost controlling, unified billing |
| ***Monitoring and SLA:*** Monitoring of applications deployed on several Clouds, using universal metrics and user-defined SLA policies. | **[Grade = 5]**<br>- SeaClouds runtime tools reply to the need to verify/enforce the Application SLA at runtime. When the system detects a violation of an SLA agreement the system properly reacts triggering user policies. | **[Grade = 5]**<br>- As regards the monitoring and SLA: minimizing operational effort, and automated alerts and actions |
| ***Repairing (Cloud resource scaling).*** Scale cloud resources to maximize the performance of the application; restart and replace failed components | **[Grade = 4]**<br>- This functionality will ensure the application will remain responsive and its performance is affected by, i.e. increased stream data traffic or by complex analysis involved, since the system will be self-adaptive to these cases by scaling out and down, depending on resources needed. | **[Grade = 5]**<br>- In the case of this case study, automated scaling is very important in real scenarios |
| ***Replaning and Application migration:*** Enables portability of applications between dissimilar clouds (but functionally compatible) | **[Grade = 3]**<br>- The functionality may be useful in the case of a "disaster recovery plan". SeaClouds supports in a semi-automated way the portability of the application between dissimilar clouds overcoming the vendor lock-in. | **[Grade = 4]**<br>- This functionality beyond vendor lock-in, migrating modules as required in the case study |
| ***Database migration and data synchronization:*** Enables portability of data between dissimilar Clouds | **[Grade = 3]**<br>- When the Application migration happens, the Persistent Layer of the Application should be migrated as well, maintaining its data synchronized. | **[Grade = 3]**<br>- Using this functionality, the case study could benefits of simplifying failover |

Table 9. SeaClouds functionalities matching with the Case Studies

## 5. Conclusions

In this document, we have presented the ***Final SeaClouds Architecture***. The project aims at providing an open source framework to address the problem of deploying, managing and reconfiguring complex applications over multiple and heterogeneous clouds.

The SeaClouds approach works towards achieving *"Agility After Deployment"* by tackling the problem from the service orchestration perspective, applying an agile execution approach, stressing both the design time with the development and planning, and the runtime with the deployment, monitoring and reconfiguration of cloud applications. Following the architecture proposed, first, the exploitation of the best available offering for each application component at any time is performed. Then, a complex application, which consists of modules and (technological and QoS) requirements, is provided as input to the SeaClouds planner. The latter generates the orchestration by assigning (groups of) modules to different cloud providers. Such orchestration is then deployed and monitored according to standard metrics. If requirements are violated, then the SeaClouds monitor will generate reconfiguration information which leverages the creation of a different orchestration of the application***.***

Therefore, the proposed architecture can well support this process of deploying and managing a cloud application over multiple clouds.

Thanks to the seamless distribution over several different platforms and infrastructure clouds, applications developed in SeaClouds will also take advantage of higher availability (via inter-cloud redundancy), higher security (via inter-cloud data partition) and higher throughput (via inter-cloud load balancing).

A key ingredient in our proposal is the use of two OASIS standards initiatives for cloud interoperability, namely CAMP and TOSCA, which allow us to describe the topology of user applications independently of cloud providers, provide abstract plans, and discover, deploy/reconfigure, and monitor our applications independently of the particularities of the cloud providers.

## References

1. SeaClouds Project. Deliverable D2.2 Initial Architecture and design of the SeaClouds Platform (SeaClouds Consortium), http://seaclouds-project.eu/deliverables/SeaClouds-D2_2-Initial_architecture_and_design_of_the_SeaClouds_platform.pdf, 2014.

2. SeaClouds Project. Deliverable D3.1 Discovery, Design and Orchestration Functionalities: First Specification (SeaClouds Consortium), http://seaclouds-project.eu/deliverables/SEACLOUDS-D3.1-Discovery_Design_and_Orchestration_Functionalities%20_First_Specification.pdf, 2014.

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

4. OASIS. TOSCA 1.0 (Topology and Orchestration Specification for Cloud Applications), Version 1.0, http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf, 2013.

5. OASIS. CAMP 1.1 (Cloud Application Management for Platforms), Version 1.1, http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.pdf, 2014.

6. SeaClouds Project. Deliverable D4.2 Cloud Application Programming Interface (SeaClouds Consortium), http://seaclouds-project.eu/deliverables/SEACLOUDS-D4.2-Cloud_Application_Programming_Interface.pdf, 2014.

7. Deliverable 4.5. Unified dashboard and revision of Cloud API, coming on the next months.

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 D4.3 Design of the run-time reconfiguration process (SeaClouds Consortium), coming on the next months.

12. MODAClouds Data Cololector Factory https://github.com/deib-polimi/modaclouds-data-collector-factory

13. MODAClouds deliverable D6.3.2 http://www.modaclouds.eu/wp-content/uploads/2012/09/MODAClouds_D6.3.2_MonitoringPlatformFinalRelease.pdf

14. MODAClouds deliverable D5.2.2 http://www.modaclouds.eu/wp-content/uploads/2012/09/MODAClouds_D5.2.2_-MODACloudMLQoSAbstractionsAndPredictionModelsSpecificationFinalVersion.pdf

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

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