SeaClouds Project

D5.4.3 - Final version of sw platform

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	**WP5 Integration, infrastructure delivery and GUI**
Due Date:	M29
Submission Date:	M30
Version:	1.1
Status	Final
Author(s):	Andrea Turli (Cloudsoft) Diego Pérez (Polimi) Javier Cubo (UMA)
Reviewer(s)	Román Sosa González (Atos), Elisabetta Di Nitto (Polimi)

Dissemination Level

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

Table of Contents

Executive Summary 4
1. Introduction 5
1.1. Glossary of Acronyms 5
2. Short overview of the final SeaClouds platform (Cloudsoft + UMA) 7
3. Implementation of repairing features (UMA) 8
4. Installation and deployment 10
4.1. Run SeaClouds 10
5. Detailed documentation of the platform 14
5.1. SeaClouds Dashboard 14
5.1. SeaClouds Deployer 15
5.1. SeaClouds Planner 15
5.1. SeaClouds Discoverer 16
5.2. SeaClouds Monitor 16
6. The SeaClouds testbed 19
7. Conclusions 20

Executive Summary

The objective of this document is to give an overview of the final SeaClouds integrated prototype, which is the actual deliverable D5.4.3, as outcome of the implementation work done in the technical work packages WP3 and WP4.

The document provides a short overview of the final SeaClouds architecture and of the repairing functionality that was the last feature implemented in the SeaClouds platform. Moreover, it provides instructions to deploy and configure the software, presents the detailed documentation of all components and describes the testbed that the consortium has been using to support the integration and evaluation phases.

1. Introduction

This document is accompanying the final release of the SeaClouds platform. Such a release can be downloaded from the SeaClouds github repository https://github.com/SeaCloudsEU.

The document is structured as follows:

- Section 2 is an overview of the SeaClouds final architecture.
- Section 3 provides an overview of the repairing feature.
- Section 4 explains how to deploy and configure the final version of the SeaClouds platform.
- Section 5 provides the detailed documentation of all components.
- Section 6 describes the SeaClouds testbed that has been used in the integration and evaluation phases.
- Finally, Section 7 draws the conclusions.

This deliverable references other deliverables where appropriate to avoid repetitions.

1.1. Glossary of Acronyms

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

Table 1: Glossary of acronyms

2. Short overview of the final SeaClouds platform

This section provides an overview of the software architecture of the SeaClouds platform in order to make this deliverable self-contained. It concentrates on the illustration and description of components that are relevant for the installation and deployment of the platform. For a detailed description of SeaClouds software architecture, readers are referred to *Final SeaClouds Architecture* deliverable [1].

The software platform is composed of six main components, namely *Dashboard, Deployer, Discoverer, Monitor, Planner and SLA Service*. Figure 1 shows a Component and Connector view of the software architecture. Figure 1 also illustrates the software packages that are relevant for the deployment and installation of each component.

At first, SeaClouds user interacts with the *Dashboard* to describe the application to deploy and specify the Quality of Service with which the application should run. The result of this step is sent to the *Planner* component. Then, the Planner creates a set of deployment proposals in TOSCA language, which will be sent back to the *Dashboard* in order to allow the user to choose his/her preferred deployment. The *Planner* is able to generate this set of proposals by using the *Discoverer* functionality[1], which, by querying public repositories as cloudharmony.com or paasify.it, finds available cloud resources, both IaaS and PaaS, and their properties and stores them in a mongoDB database [2].

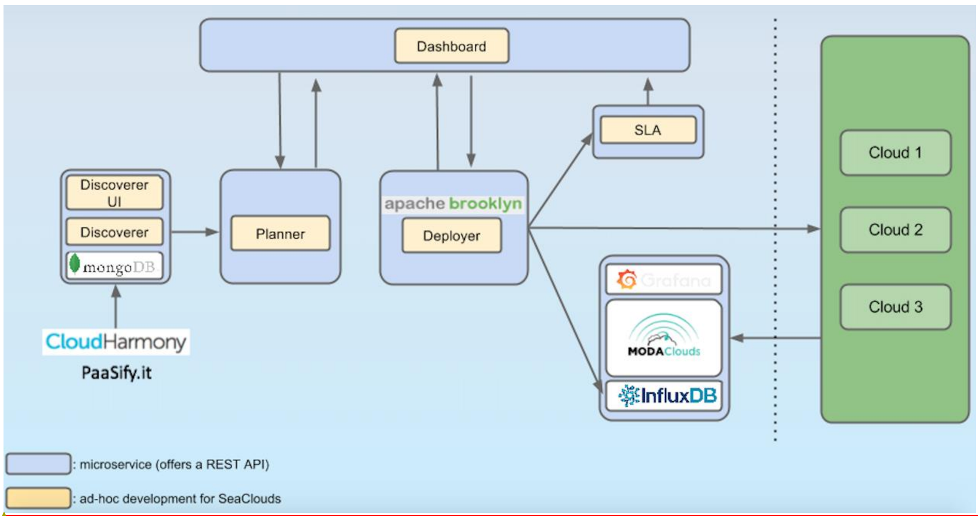
After the user has chosen one of the proposals, the *Planner* is invoked again, this time to automatically create a Deployable Application Model (DAM), which will be a TOSCA description of the application that will already contain all the necessary information for its automatic deployment, installation and monitoring. Once the DAM is created, it is sent back to the *Dashboard*, where the user can confirm the proposed deployment.

The *Dashboard* sends the confirmed DAM to the *Deployer.* In that moment, the *Deployer* starts the deployment and installation of the application through the Apache Brooklyn [3] engine that has been enhanced by SeaClouds team. The application installation process also involves the configuration of the elements relevant the monitoring and SLA control tasks. These tasks are executed by *Monitor* and *SLA service* components, respectively. The *Monitor* component requires the installation of an extended version of ModaClouds Tower4Clouds [4][4] monitoring engine, InfluxDB [5]to store the monitored data, and Grafana [6] to graphically show monitored information.

Once the user application is running, the *Deployer* is in charge of its runtime management, as this component also includes application repair features through

[1] The Discoverer is also equipped with a web-based GUI (called DrACO), which permits retrieving the TOSCA-based representation of available cloud offerings.

autoscaling techniques. Next section details the implementation of these repairing features.



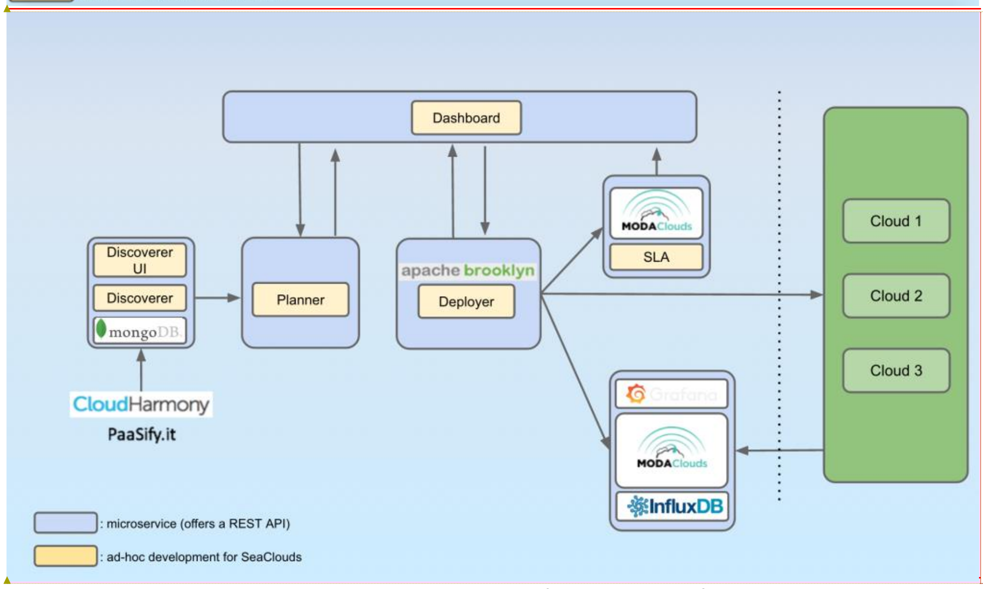


Figure 1: Components, modules and connectors of the SeaClouds software architecture

Formatted: Font: (Default) Times New Roman, Font color: Auto

Comment [ED1]: From Roman: remove the MODAClouds logo from SLA

Comment [JS2]: At the level of the Dashboard, we would need a box representing the DrACO interface, which is connected with the Discoverer.

Formatted: Font: (Default) Arial, 11 pt, Font color: Black

3. Implementation of repairing features

In SeaClouds, the Deployer Engine is mainly in charge of deploying the plan (which can consist of multiple components that need to be configured and integrated across multiple machines, but it is also responsible (if necessary) of scaling to meet demand

and restarting failed components. We refer to this last task as `repairing`, and it was initially described in [D4.3][2].

The Deployer Engine is allowed to modify the deployed application whenever this is needed. According to the runtime inputs and the policies defined at the level of the entire application and/or components, Deployer Engine is capable to adjust the application using the instructions (rules or policies) generated from the user requirements. For example, the Deployer Engine can add more resources to a deployed application to meet the growing demand.

For example, the following YAML represents an Abstract Deployable Profile (ADP). It includes part of the plan corresponding to a Java web server (TomcatServer) description, and an autoscaling policy is added to the group just composed by the afore-mentioned node template. This policy is added because the user marked as active the autoscale option when the topology and module requirements were introduced through the SeaClouds GUI. In the same way, the kind of metric was chosen by the Optimizer according to the node template type. Moreover, the rest of values (e.g. minimum and maximum pool sizes, or upper and lower bounds) were inferred from the user inputs and the capabilities and requirements of selected offerings. Notice that most of the information included in this ADP is cloud agnostic. That is, no information about specific technologies is included. For example, the autoscaling policy only includes information about the type of metrics (requests per node), independently of the node template type.

```
node_templates:
  Softcare_dashboard:
    type: seaclouds.nodes.webapp.tomcat.TomcatServer
    properties:
      language: JAVA
      autoscale: true
    artifacts:
    - wars.root: https://s3-eu-west-1.amazonaws.com/atos-
paas/v3/softcare-gui.war
      type: tosca.artifacts.File
    requirements:
    - host: Cloud_Foundry
      instancesPOC: 2

groups:
   operation_Software_dashboard:
     members:
     - Softcare_dashboard
     policies:
     - autoscaling:
         type: seaclouds.policies.autoscaling.AutoScalerPolicy
         autoscaler.resizeDownStabilizationDelay: 120000
         metric: seaclouds.metrics.requestPerNode
         minPoolSize: 1
         maxPoolSize: 5
         metricUpperBound: 20.104166666666664
```

[2] [D4.3] SeaClouds consortium. Deliberable D4.3: Design of the runtime reconfiguration process. July 2015.

```
        metricLowerBound: 10.052083333333332
```

From this information, a Deployable Abstract Model (DAM) is obtained by the DAMGenerator, which is accepted by the Deployer Engine, based on Brooklyn-TOSCA. In this case, the type of the metric is converted to an appropriate available metric according to the node template supported by the Deployer. The same is applicable to the policy type, an appropriate and available policy in the Deployer is chosen.

```
node_templates:
  Softcare_dashboard:
    type:
org.apache.brooklyn.entity.cloudfoundry.webapp.java.JavaCloudFoundryPa
asWebApp
    properties:
      language: JAVA
      autoscale: true
      application-url: https://s3-eu-west-1.amazonaws.com/atos-
paas/v3/softcare-gui.war

groups:
 operation_Software_dashboard:
   members: [ Softcare_dashboard ]
   policies:
     - autoscaling:
         type: org.apache.brooklyn.policy.autoscaling.AutoScalerPolicy
         metric: app.server.requestpersecond
         minPoolSize: 1
         maxPoolSize: 5
         metricUpperBound: 20.104166666666664
         metricLowerBound: 10.052083333333332
```

4. Installation and deployment

This paragraph explains how to install SeaClouds platform using Apache Brooklyn.
Install the prerequisites

Vagrant

Virtualbox

Download latest SeaClouds Platform release from github.com:

```
$ wget https://github.com/SeaCloudsEU/SeaCloudsPlatform/archive/1.0.0.tar.gz
```

Untar the package

```
$ tar -zxf SeaCloudsPlatform-1.0.0.tar.gz
$ cd SeaCloudsPlatform-1.0.0/byon
```

Run vagrant

```
$ vagrant up
```

Point your favourite browser at http://10.10.10.100:8081

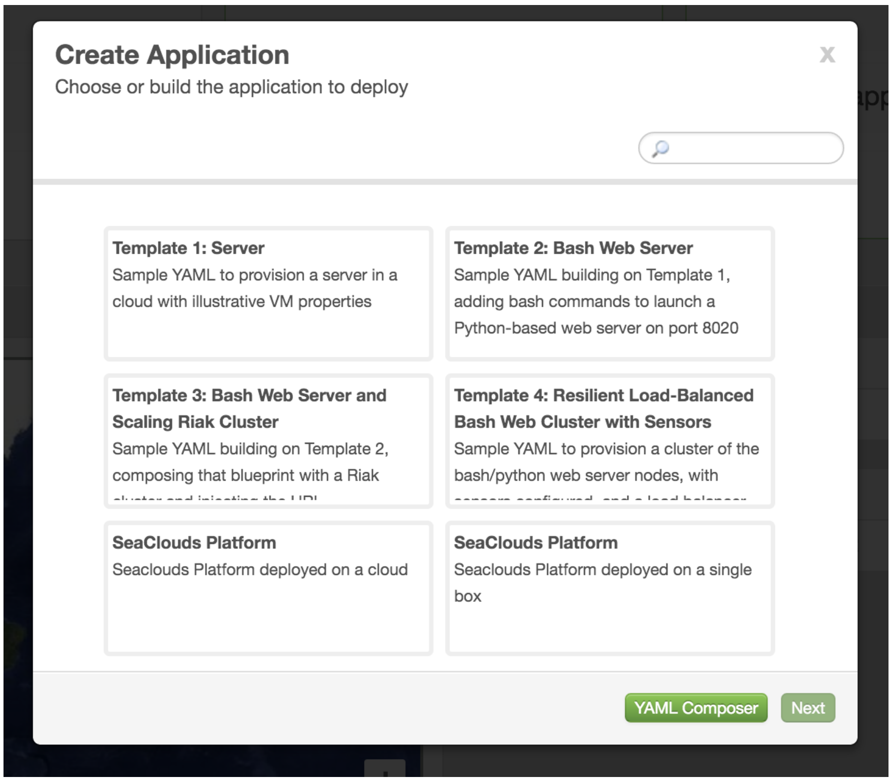
4.1. Run SeaClouds

Once you have the SeaClouds installer (aka Apache Brooklyn) running, you will have 2 new applications available.

Run SeaClouds on BYON

By selecting, "SeaClouds Platform on BYON" you can deploy SeaClouds on a single node. BYON stands for Bring Your Own Node, and requires the user to specify the IP addresses of the target nodes that will host the SeaClouds platform, in this particular case it will always be 10.10.10.100, if you are running the installer inside Vagrant.
Steps:

1. Select the SeaClouds Platform on a single box
2. Click on YAML Composer. A YAML blueprint is shown (see Figure 3). You don't need to do anything here.
3. Finally Click Deploy button
4. After few minutes you will see something like


Create Application
Choose or build the application to deploy

Template 1: Server
Sample YAML to provision a server in a cloud with illustrative VM properties

Template 2: Bash Web Server
Sample YAML building on Template 1, adding bash commands to launch a Python-based web server on port 8020

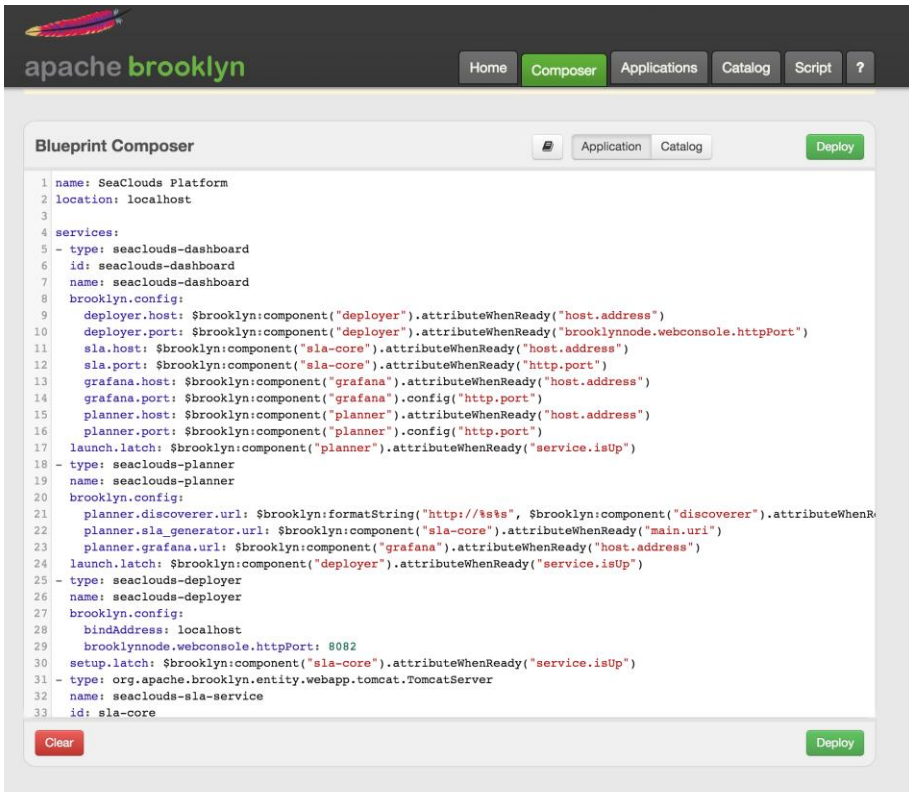
Template 3: Bash Web Server and Scaling Riak Cluster
Sample YAML building on Template 2, composing that blueprint with a Riak

Template 4: Resilient Load-Balanced Bash Web Cluster with Sensors
Sample YAML to provision a cluster of the bash/python web server nodes, with

SeaClouds Platform
Seaclouds Platform deployed on a cloud

SeaClouds Platform
Seaclouds Platform deployed on a single box

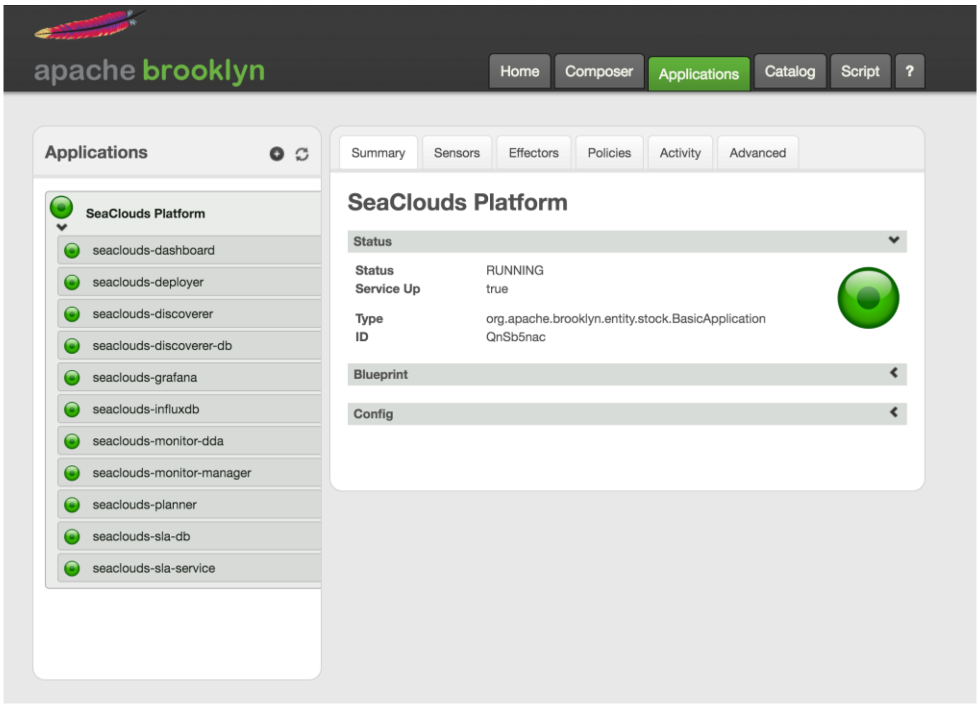
Figure 2: Deploy SeaClouds on your a single node



```yaml
name: SeaClouds Platform
location: localhost

services:
- type: seaclouds-dashboard
  id: seaclouds-dashboard
  name: seaclouds-dashboard
  brooklyn.config:
    deployer.host: $brooklyn:component("deployer").attributeWhenReady("host.address")
    deployer.port: $brooklyn:component("deployer").attributeWhenReady("brooklynnode.webconsole.httpPort")
    sla.host: $brooklyn:component("sla-core").attributeWhenReady("host.address")
    sla.port: $brooklyn:component("sla-core").attributeWhenReady("http.port")
    grafana.host: $brooklyn:component("grafana").attributeWhenReady("host.address")
    grafana.port: $brooklyn:component("grafana").config("http.port")
    planner.host: $brooklyn:component("planner").attributeWhenReady("host.address")
    planner.port: $brooklyn:component("planner").config("http.port")
  launch.latch: $brooklyn:component("planner").attributeWhenReady("service.isUp")
- type: seaclouds-planner
  name: seaclouds-planner
  brooklyn.config:
    planner.discoverer.url: $brooklyn:formatString("http://%s%s", $brooklyn:component("discoverer").attributeWhenR
    planner.sla_generator.url: $brooklyn:component("sla-core").attributeWhenReady("main.uri")
    planner.grafana.url: $brooklyn:component("grafana").attributeWhenReady("host.address")
  launch.latch: $brooklyn:component("deployer").attributeWhenReady("service.isUp")
- type: seaclouds-deployer
  name: seaclouds-deployer
  brooklyn.config:
    bindAddress: localhost
    brooklynnode.webconsole.httpPort: 8082
  setup.latch: $brooklyn:component("sla-core").attributeWhenReady("service.isUp")
- type: org.apache.brooklyn.entity.webapp.tomcat.TomcatServer
  name: seaclouds-sla-service
  id: sla-core
```

Figure 3: A YAML blueprint in the Composer



SeaClouds Platform
- seaclouds-dashboard
- seaclouds-deployer
- seaclouds-discoverer
- seaclouds-discoverer-db
- seaclouds-grafana
- seaclouds-influxdb
- seaclouds-monitor-dda
- seaclouds-monitor-manager
- seaclouds-planner
- seaclouds-sla-db
- seaclouds-sla-service

SeaClouds Platform

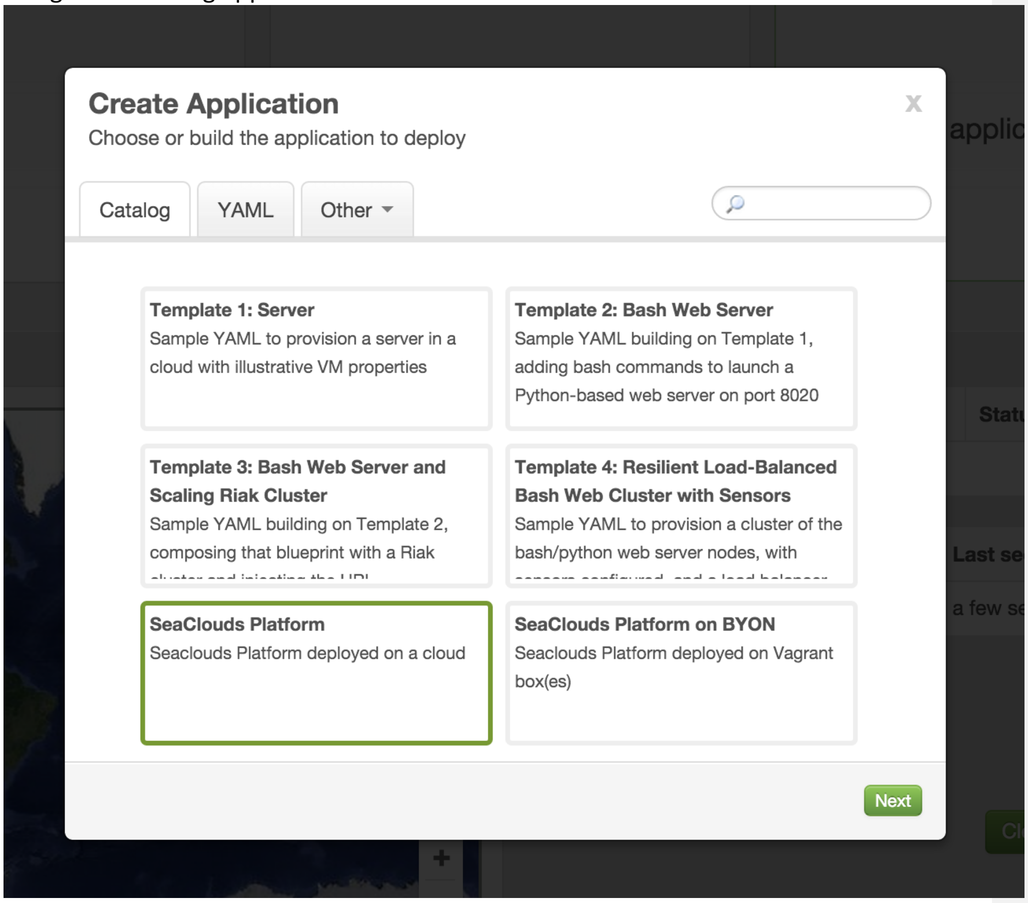
Status

Status	RUNNING
Service Up	true
Type	org.apache.brooklyn.entity.stock.BasicApplication
ID	QnSb5nac

Figure 4: The SeaClouds platform appears to be running.

Run SeaClouds on a cloud

Similarly, to the previous case, you can easily deploy SeaClouds platform to AWS EC2 using the following application item:


Create Application

Choose or build the application to deploy

Template 1: Server
Sample YAML to provision a server in a cloud with illustrative VM properties

Template 2: Bash Web Server
Sample YAML building on Template 1, adding bash commands to launch a Python-based web server on port 8020

Template 3: Bash Web Server and Scaling Riak Cluster
Sample YAML building on Template 2, composing that blueprint with a Riak

Template 4: Resilient Load-Balanced Bash Web Cluster with Sensors
Sample YAML to provision a cluster of the bash/python web server nodes, with

SeaClouds Platform
Seaclouds Platform deployed on a cloud

SeaClouds Platform on BYON
Seaclouds Platform deployed on Vagrant box(es)

Figure 5: Select deployment on a cloud provider

Be sure to edit the `identity` and the `credential` fields with the AWS "Access Key ID" and "Secret Access Key", respectively and finally click the Deploy button.

5. Detailed documentation of the platform

This section reports the detailed documentation that describes how to install and run each individual component of the SeaClouds platform, organized per component. All the requirements and configuration files, where needed, will be explicated.

5.1. SeaClouds Dashboard

This component provides an easy way to interact with SeaClouds Platform by using an Angular.js web application.

Running SeaClouds Dashboard requires Java 7 (or greater) installed on the target machine and the following other requirements:

- SeaClouds Planner and SeaClouds Deployer up and running on an accessible endpoint.
- 1 free TCP port.
- A config.yml configuration file with the following parameters:
  - server.connector.port: A positive number which will be used by Dropwizard to expose the Dashboard. Required. Eg. 8000.
  - planner.host: SeaClouds Planner IP. Required. Eg. 127.0.0.1.
  - planner.port: SeaClouds Planner Port. Required. Eg. 1234.
  - deployer.host: SeaClouds Deployer IP. Required. Eg. 127.0.0.1.
  - deployer.port: SeaClouds Planner Port. Required. Eg. 8081.
  - deployer.username: SeaClouds Deployer Username. Optional. Eg. user.
  - deployer.password: SeaClouds Deployer Password. Optional. Eg. password.
  - monitor.manager.host: SeaClouds Monitor (Tower4Clouds Monitoring Manager) IP. Required. Eg. 127.0.0.1.
  - monitor.manager.port: SeaClouds Monitor (Tower4Clouds Monitoring Manager) Port. Required. Eg. 8710.
  - monitor.grafana.host: SeaClouds Monitor Dashboard (Grafana) IP. Required. Eg. 127.0.0.1.
  - monitor.grafana.port: SeaClouds Monitor Dashboard (Grafana) Port. Required. Eg. 3000.
  - sla.host: SeaClouds Deployer IP. Required. Eg. 127.0.0.1.
  - sla.port: SeaClouds SLA Port. Required. Eg. 8080.

To install SeaClouds Dashboard in standalone mode, one can download the latest artifact from http://search.maven.org/remotecontent?filepath=eu/seaclouds-project/dashboard/1.0.0/dashboard-1.0.0.jar and then run:

```
java -jar dashboard.jar server path/to/config.yml
```

5.1. SeaClouds Deployer

The main goal of the SeaClouds Deployer component is to deploy the application in a multi-cloud environment.

Install and runnning SeaClouds Deployer requires an Apache Brooklyn 0.9.0 installation plus a couple of additional artifacts available at http://search.maven.org/remotecontent?filepath=eu/seaclouds-project/deployer/1.0.0/deployer-1.0.0.jar which adds PaaS support and some new Brooklyn entities and developed for SeaClouds.

SeaClouds deployer jar has to be added to `$BROOKLYN_HOME/bin/lib/dropins` folder.

5.1. SeaClouds Planner

This component provides design time functionalities for the SeaClouds Platform.
It provides planning and DAM generation.

Running SeaClouds Planner requires Java 7 (or greater) installed on the target machine and:

- A SeaClouds Discoverer, Sla, Monitor component services up and running since they will be called via HTTP by the planner.
- A plannerconf.yml configuration file.
- A config.yml configuration file with the following parameters:
    - planner port (1234 by default)
    - discovererURL is the Discoverer Component URL and port in the format http://{ip}:{port}/
    - monitorGeneratorURL is the Monitor Component URL (http://{ip})
    - monitorGeneratorPort is the Monitor Component port ({number})
    - slaGeneratorURL: is the SLA Component URL and port in the format http://{ip}:{port}/
    - deployableProviders is the list of providers that the Deployer component is able to deploy (by default ["openstack-nova","openstack-keystone","openstack-nova-ec2", "byon", "sts", "elasticstack", "cloudstack", "rackspace-cloudidentity","aws-ec2","gogrid","elastichosts-lon-p","elastichosts-sat-p","elastichosts-lon-b","openhosting-east1","serverlove-z1-man","skalicloud-sdg-my","go2cloud-jhb1","softlayer","hpcloud-compute","rackspace-cloudservers-us","rackspace-cloudservers-uk","azurecompute","google-compute-engine","CloudFoundry"])
    - filterOfferings is a boolean flag that enables the filter of non deployable providers for the matching process (default is false)

All the configuration info are required.

To start the SeaClouds Planner, once you have fulfilled the requirements you only need to run on JRE (>=1.7): java -jar planner-service.jar server path/to/config.yml.

5.1. SeaClouds Discoverer

The SeaClouds Discoverer module is able to provide information about cloud offerings through its RESTful API.

Those information are a TOSCA YAML representation (following the SeaClouds Discovery design and orchestration functionalities specification) of the offerings retrieved by the crawlers from https://cloudharmony.com and http://www.paasify.it.

Running SeaClouds Discoverer requires Java 7 (or greater) installed on the target machine and:

- A discovererconf.yml configuration file
- A running MongoDB service, used as a permanent layer to store information about cloud offerings used by SeaCloudsPlatform
- A config.yml configuration file with the following parameters:
    - discoverer port (1236 by default)
    - activeCrawlers is the list of crawlers to use (currently we only support "CloudHarmonyCrawler" and "PaasifyCrawler")
    - databaseURL is the URL of the MongoDB that will be used to store offerings
    - databasePort

All the configuration variables are required.

Once you have fulfilled the requirements you only need to run on JRE (>=1.7): java -jar discoverer.jar server path/to/config.yml.

5.2. SeaClouds Monitor

The SeaClouds monitor module is based on Tower4Clouds, a monitoring platform for multi-clouds application developed in the context of the MODAClouds FP7 European Project.
The released platform has been extended and customized in order satisfy specific SeaClouds monitoring requirements. More specifically, since Tower4Clouds relies on the concept of Data Collector (DC), or the components responsible to collect

monitoring metrics, as main the mechanism to extend the Platform, a number of new Tower4Clouds's Data Collectors has been developed in SeaCloud.

Moreover, being Tower4Clouds already integrated with some suitable external tools enabling visualization of monitoring data, like InfluxDB and Graphite, we decided to exploit this integration as a way to provide application metrics visualization in SeaClouds.

In the following, we are going to provide instructions on how to install each component belonging to the SeaClouds Monitor module starting from a fresh installation of Ubuntu 14.04.

SeaClouds currently use Tower 4Clouds version 0.2.3. In order to work Tower 4Clouds needs Java 7 installed. In order to download Tower 4Clouds Data Analyzer v0.2.3 and Tower 4Clouds Monitor Manager v.0.2.3 you can run the following command:

wget https://github.com/deib-polimi/tower4clouds/releases/download/v0.2.3/data-analyzer-0.2.3.tar.gz
wget https://github.com/deib-polimi/tower4clouds/releases/download/v0.2.3/manager-server-0.2.3.tar.gz

The only required configuration is that the Data Analyzer need to know the public IP of the Monitor Manager. In order to do that you can export the following environmental variable:

export MODACLOUDS_TOWER4CLOUDS_DATA_ANALYZER_ENDPOINT_IP_PUBLIC=<MONITOR-MANAGER-PUBLIC-PORT>

After that, you need to untar the downloaded files and start the the two services using their own starter scripts. You could use the following script:

```
    tar -xvzf data-analyzer-0.2.3.tar.gz
    tar -xvzf manager-server-0.2.3.tar.gz
    cd data-analyzer-0.2.3
    rm -f tower4clouds-data-analyzer.log
    nohup bash tower4clouds-data-analyzer > tower4clouds-data-analyzer.log 2>&1 &
    cd ..
    cd manager-server-0.2.3
    rm -f tower4clouds-manager.log
    nohup bash tower4clouds-manager > tower4clouds-manager.log 2>&1 &
```

After that you can check that the platform is up and running by connecting to http://<MONITOR-MANAGER-HOST-IP>:8170/webapp to see the Monitor Manager web console.

Finally, according with the current status of Tower 4Clouds, it is necessary to install an initial static monitoring rule enabling the monitoring of the response time for application modules. You can use the following curl in order to do that, properly replacing the Monitoring Manager endpoint information (IP and port):

```
    curl    -X    POST    -H    "Content-type:    application/xml"
http://${MONITOR_MANAGER_HOST}:${MONITOR_MANAGER_PORT}/v1/monito
ring-rules                        -d                            '
    <ns2:monitoringRules>
        <ns2:monitoringRule         id="internalComponentRTRule"
timeStep="2"                                       timeWindow="2">
            <ns2:monitoredTargets>
                <ns2:monitoredTarget            class="Method"/>
            </ns2:monitoredTargets>
            <ns2:collectedMetric
metricName="EffectiveResponseTime">
                <ns2:parameter
name="samplingProbability">1</ns2:parameter>
            </ns2:collectedMetric>
            <ns2:metricAggregation
groupingClass="InternalComponent"   aggregateFunction="Average"/>
            <ns2:actions>
                <ns2:action                  name="OutputMetric">
                    <ns2:parameter
name="metric">AverageResponseTimeInternalComponent</ns2:paramete
r>
                    <ns2:parameter
name="value">METRIC</ns2:parameter>
                    <ns2:parameter
name="resourceId">ID</ns2:parameter>
                </ns2:action>
            </ns2:actions>
        </ns2:monitoringRule>
    </ns2:monitoringRules>
    '
```

###InfluxDB

InfluxDB will use ports 8083, 8086, 8090, and 8099. Once you install you can change those ports and other options in the configuration file, which is located at either /opt/influxdb/shared/config.toml or /usr/local/etc/influxdb.conf"

Tower 4Clouds comes integrated with InfluxDB version 0.8. In order to install InfluxDB one can refer to the InfluxDB installation documentation available at the following [link](https://influxdb.com/docs/v0.8/introduction/installation.html#ubuntu-debian).

In particular it is sufficient to run the following commands to download and install InfluxDB:

```
    #for 64-bit systems
    wget
http://get.influxdb.org.s3.amazonaws.com/influxdb_0.8.9_amd64.de
b
    sudo dpkg -i influxdb_0.8.9_amd64.deb
```

In order to start InfluxDB just run the following:

```
    sudo service influxdb start
```

In order to install Grafana you need to download and install Grafana the following commands need to be executed:

```
    wget
https://grafanarel.s3.amazonaws.com/builds/grafana_2.5.0_amd64.d
eb
    sudo apt-get install -y adduser libfontconfig
    sudo dpkg -i grafana_2.5.0_amd64.deb
```

To start Grafana just run the following:

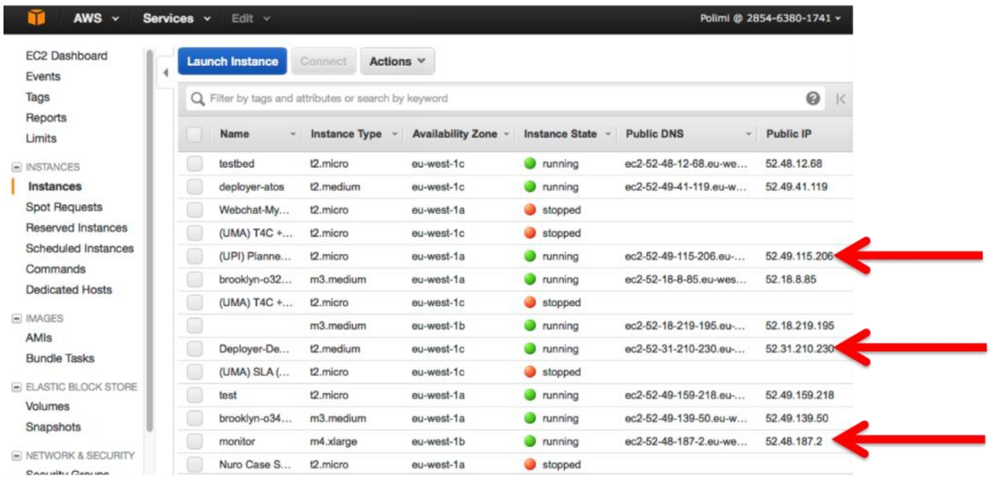
```
     sudo service grafana-server start
```

6. The SeaClouds testbed

This section describes the platform integration testbed, as an update of the testing platform Amazon Machine Image (AMI) explained in [D5.4.2][3].

The goal of the updated testbed is to facilitate the integration testing of SeaClouds components. The testbed is composed of a set of Virtual Machines on AWS. Figure 6 depicts a view of EC2 Dashboard with the instances used for components development and integration testing. Concretely, red arrows point to the VMs that are part of the testbed environment.

[3] [D5.4.2] Seaclouds consortium. Deliverable D5.4.2. Second version of the SW platform. September 2015.

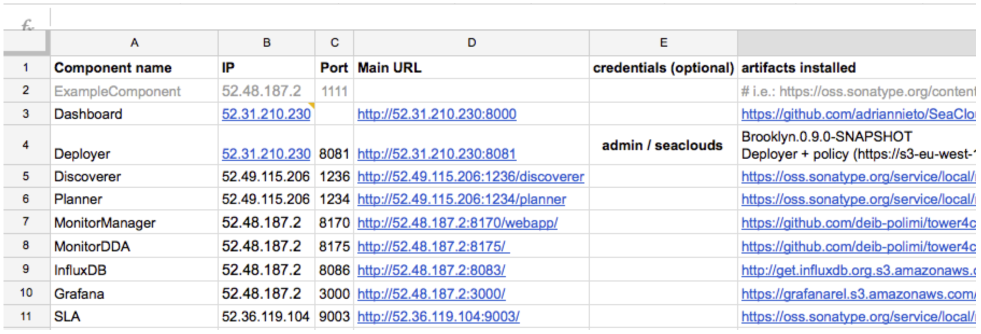


Name	Instance Type	Availability Zone	Instance State	Public DNS	Public IP
testbed	t2.micro	eu-west-1c	● running	ec2-52-48-12-68.eu-we…	52.48.12.68
deployer-atos	t2.medium	eu-west-1c	● running	ec2-52-49-41-119.eu-w…	52.49.41.119
Webchat-My…	t2.micro	eu-west-1a	● stopped		
(UMA) T4C +…	t2.micro	eu-west-1c	● stopped		
(UPI) Planne…	t2.micro	eu-west-1a	● running	ec2-52-49-115-206.eu-…	52.49.115.206
brooklyn-o32…	m3.medium	eu-west-1a	● running	ec2-52-18-8-85.eu-wes…	52.18.8.85
(UMA) T4C +…	t2.micro	eu-west-1c	● stopped		
	m3.medium	eu-west-1b	● running	ec2-52-18-219-195.eu-…	52.18.219.195
Deployer-De…	t2.medium	eu-west-1c	● running	ec2-52-31-210-230.eu-…	52.31.210.230
(UMA) SLA (…	t2.micro	eu-west-1c	● stopped		
test	t2.micro	eu-west-1a	● running	ec2-52-49-159-218.eu-…	52.49.159.218
brooklyn-o34…	m3.medium	eu-west-1a	● running	ec2-52-49-139-50.eu-w…	52.49.139.50
monitor	m4.xlarge	eu-west-1b	● running	ec2-52-48-187-2.eu-we…	52.48.187.2
Nuro Case S…	t2.micro	eu-west-1a	● stopped		

Figure 6: VMs of the SeaClouds testbed from the AWS console.

The software programs running in the each of the instances used for the testbed are updated while new revisions of SeaClouds components are built.

The responsible of each SeaClouds component that offers its functionality through a REST API is responsible for setting the service up in the AWS instance and updating the information to the rest of the consortium developers in order to reach the REST service. We use a spreadsheet to store the information that each developer who is responsible of a REST service has to share with the rest of developers for the integration testing. Figure 7 shows the spreadsheet with the descriptive information that developers have to include about their services. For each service, it is required the IP address and port where the software listens to connections, the main URL to reach the service, credentials to access the service in case that they were necessary and the URL from where anybody can download the software that executes the service.



	A	B	C	D	E	
1	Component name	IP	Port	Main URL	credentials (optional)	artifacts installed
2	ExampleComponent	52.48.187.2	1111			# i.e.: https://oss.sonatype.org/content
3	Dashboard	52.31.210.230		http://52.31.210.230:8000		https://github.com/adriannieto/SeaClo
4	Deployer	52.31.210.230	8081	http://52.31.210.230:8081	admin / seaclouds	Brooklyn.0.9.0-SNAPSHOT Deployer + policy (https://s3-eu-west-
5	Discoverer	52.49.115.206	1236	http://52.49.115.206:1236/discoverer		https://oss.sonatype.org/service/local/
6	Planner	52.49.115.206	1234	http://52.49.115.206:1234/planner		https://oss.sonatype.org/service/local/
7	MonitorManager	52.48.187.2	8170	http://52.48.187.2:8170/webapp/		https://github.com/deib-polimi/tower4c
8	MonitorDDA	52.48.187.2	8175	http://52.48.187.2:8175/		https://github.com/deib-polimi/tower4c
9	InfluxDB	52.48.187.2	8086	http://52.48.187.2:8083/		http://get.influxdb.org.s3.amazonaws.
10	Grafana	52.48.187.2	3000	http://52.48.187.2:3000/		https://grafanarel.s3.amazonaws.com/
11	SLA	52.36.119.104	9003	http://52.36.119.104:9003/		https://oss.sonatype.org/service/local/

Figure 7: Internal excel sheet used to keep track of the configuration of all SeaClouds components.

7. Conclusions

This document accompanies the final release of the SeaClouds platform and describes the main technologies and the processes that SeaClouds development team relied on to develop the SeaClouds platform.

The development process has been highly influenced by Agile methodologies, XP programming and some of the most popular and appreciated tools in the OSS community, which has proven effective in a variety of geographically distributed development teams.
It is well-known that the process is just a way to mitigate the risk. SeaClouds had a lot of benefits from this process, but, along the way, many assumptions have been proven wrong. Successful agile teams rely on effective communication and not on tools, although they may help.

The SeaClouds consortium decided to adopt Continuous Integration to significantly reduce integration problems, allowing a team to develop cohesive software more rapidly, but the responsibility of the integration was not equally understood and postponed or ignored for the most part of the project.
CI is a software development practice where members of a team integrate their work frequently, leading to multiple integrations per day. This is possible because the consortium maintains a Single Source repository, hosted at GitHub where each developer Commits to the Mainline as soon as she has a bug fix or a new feature. This process is guarded by the so called GitHub Flow. This needed to be protected more along the way, as giving the full permission on the repository to everybody was not working as expected.
Each integration is verified by an automated build (including test) done by TravisCI to detect integration errors as quickly as possible and evaluated by Codeconv, an hosted service that is able to calculate the test coverage of the project. SeaClouds scored a decent 48% an the end of the project.

8. References

[1]. SeaClouds deliverable D2.4. *Final SeaClouds Architecture.* January 2015. Available at: http://www.seaclouds-project.eu/deliverables/SEACLOUDS-D2.4-Final_SeaClouds_Architecture.pdf

[2]. https://www.mongodb.com

[3]. https://brooklyn.apache.org

[4]. http://www.modaclouds.eu/software/open-source-repositories/

[5]. https://influxdata.com

[6]. http://grafana.org