SeaClouds Project

D5.4.2 - Second 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:	M2
Submission Date:	13th October 2015
Version:	1.1
Status	Final
Author(s):	Andrea Turli (Cloudsoft) Michele Guerriero (Polimi) Diego Pérez (Polimi)
Reviewer(s)	Roman Sosa Gonzales (Atos), Christian Tismer (NURO)

Dissemination Level

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

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	29/9/2015	Initial draft	Andrea Turli
0.5	10/10/2015	Polimi contributions	Diego Perez, Michele Guerriero
1.0	13/10/2015	Applying official template	Andrea Turli
1.1	13/10/2015	Addressed review comments	Andrea Turli, Roman Sosa Gonzales

1. Table of Contents

2. Executive Summary

The objective of this deliverable is to give an overview of the final SeaClouds integrated prototype as outcome of the implementation work done in the technical work packages WP3 and WP4, where the main activities so far are concentrated in Deliverables D3.1, D3.2 D4.1 and D4.2.

The document overviews the tools implemented by M22 and describe all the necessary process to install and configure the system.

3. Introduction

This deliverable describes the software development process and tools adopted to implement the SeaClouds platform.

This deliverable references other deliverables where appropriate to avoid repetitions.

This document is structured as follows:

- Section 4 is an overview of the SeaClouds components that constitutes the entire platform.
- Section 5 contains the overview of the development tools and pipelines adopted by the consortium.
- Section 6 explains how to install SeaClouds platform.
- Section 7 describes the SeaClouds testbed.

3.1. 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

4. Short overview of the SeaClouds platform

This section gives an overview of the current state of the software architecture of the SeaClouds platform in order to make this deliverable self-contained. For a detailed description of the platform architecture, readers are referred to deliverable D5.1.3 [D513].

The software platform is composed of six main components, which are subdivided into modules, namely *Dashboard, Deployer, Discoverer, Monitor, Planner and SLA Service*. Figure 1 shows the modules of each component and the communication messages between components.

SeaClouds user interacts with the *Dashboard* to describe the application to deploy and specify the Quality of Service with which the application should run. Result of this step is the abstract application model (AAM) which is given to the Planner. Then, the *Planner* creates a Deployable Application Model (DAM) that contains all the information necessary to deploy an application and where each module of the user's application has been assigned to a convenient cloud resource. In order to unveil the possible cloud offers onto which the modules of the user application can be deployed, the *Planner* uses the *Discoverer* functionality, that finds different public cloud resources together with both their technical (e.g., amount of RAM) and quality properties (e.g., availability of the resource). During the DAM generation, *SLA Service, Monitor* and *Deployer* components are requested to generate the part of the DAM regarding the application SLA, the monitoring rules and the repairing policies, respectively. The modules that offer these "generation" functionalities have been included in the architecture recently in order to improve the inter-component communication. A more detailed description of these modules, their dependencies, interfaces, used libraries and programing languages is given in the deliverable of the Final Integrated Platform [D513].

The proposed DAM by the *Planner* is passed to the Dashboard, where the user can confirm the proposed deployment. Then, the DAM is passed to the *Deployer* which deploys the application through the Apache Brooklyn engine enhanced in SeaClouds project with capabilities to support PHP applications and CloudFoundry and OpenShift PaaS (the later is currently under development). Again, a more detailed description of these new deployment capabilities can be found in Deliverable 5.1.3 [D513]. The *Deployer* is also able to repair it through scaling techniques and it monitors the behavior of some parameters of the application. During the application runtime, the *Monitor* collects data of its behavior, QoS and resource consumption. The *SLA Service* is subscribed to the *Monitor* and therefore it is informed of violations of the quality requirements of the application. Both the information from the *Monitor* and the *SLA Service* is passed to the *Dashboard*, hence the user can see graphically the QoS offered by its application to clients, the resource consumption of the application and the SLA violations.

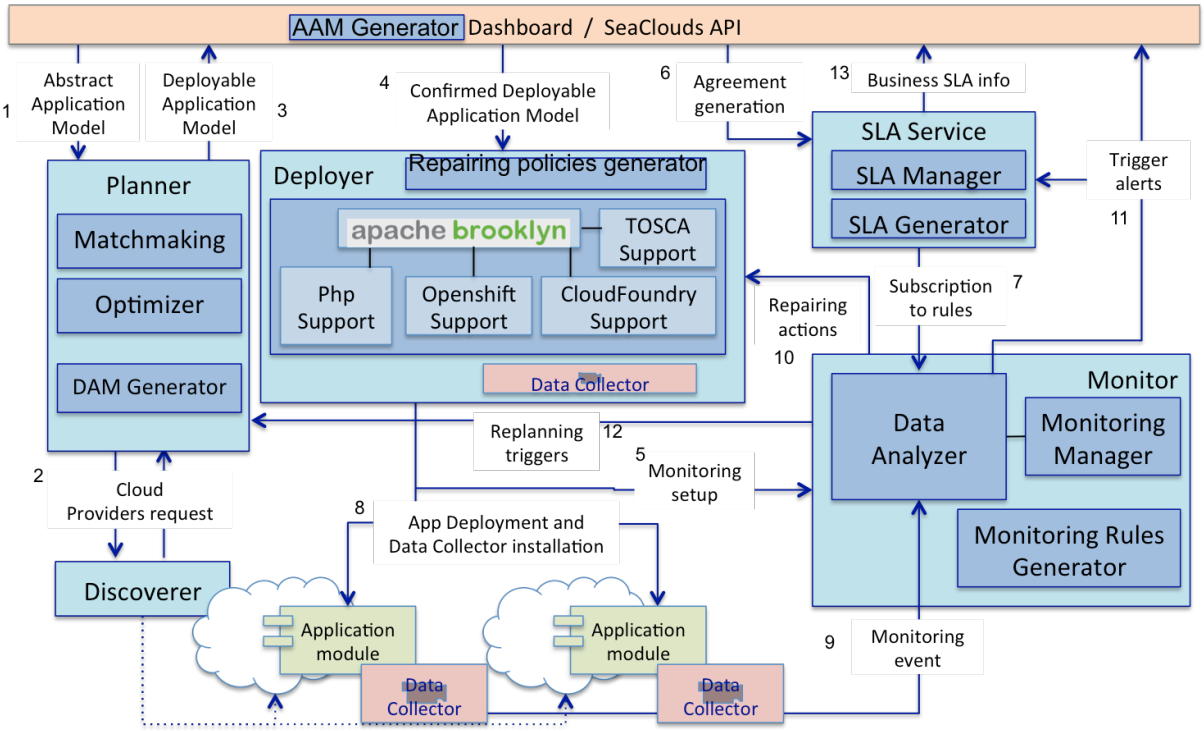

Figure 1: Components, modules and connectors of the SeaClouds software architecture

5. The SeaClouds framework for continuous integration and deployment

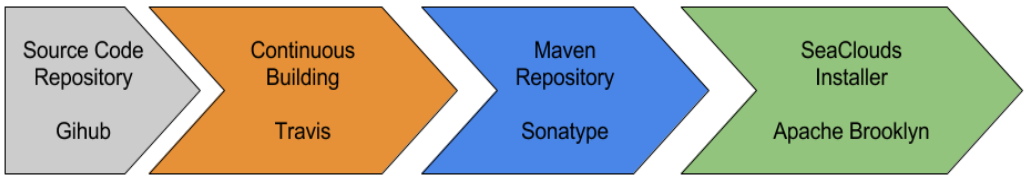

Figure 1: Development pipeline

5.1. Development cycle

The official repository for source code is hosted at https://github.com/SeaCloudsEU/SeaCloudsPlatform and it contains all the code developed for the SeaClouds Platform.

SeaClouds consortium decided to not rely on formal code review, which can be difficult and inaccurate, but it relies on a lightweight code review. In this context, for lightweight we mean that reviews are often done as part of the normal development process, but because the development team is highly distributed we rely on collaborative tool to follow techniques recommended in Extreme Programming, like

Pair Programming and/or usage of tool-assisted code review such as GitHub Pull Requests, gist and gitter.im.

5.2. Github flow

Because of the agreement on the development workflow, we get inspired by GitHub Flow https://guides.github.com/introduction/flow/ where each developer forks on its own GitHub space the official repository. A fork is a copy of the original repository. Forking a repository allows you to experiment with changes without affecting the original project. Once the developer is happy with the status of the implementation of a new feature or of a bug fixing, he opens a **pull request** (PR) and asks for a code review from another developer or the product owner. If the developer doesn't specify a reviewer, the board of project owners nominate at least one developer (different from the PR owner) as reviewer. Finally, if the project owner likes your work, they might pull your fix into the original repository
The code review happens publicly on discussions on the Github PR or gitter.im and in order to be accepted a PR needs to get at least a `+1` (as usual in Apache Software Foundation projects) or a `lgtm` (look good to me)

In parallel with the code review, a continuous building system has been set up to checkout the code contained in the PR, compile it, test it, and report on the Github PR page the result of the build. This happens for all the PRs and mitigate the risk to checkout code that breaks the build of the main project. SeaClouds is using https://travis-ci.org/ as many other successful opensource projects to implement the Continuous Building pipe. Travis easily sync your GitHub projects and is able to test the code in minutes by simply configuring the Github repository with a .travis.yml file.

5.3. Deploy Maven artifacts to Sonatype

SeaClouds is a java-based project built using Apache Maven. Travis-CI takes care of the build but also takes care of the deployment of the generated artifacts to a public Maven repository. In particular, the build is able to produces SNAPSHOT and RELEASE artifacts, so any time a new commit is added to the master branch of SeaCloudsEU/SeaCloudsPlatform repository, all the generated artifacts from a successful build are pushed to Sonatype OSS. This space has been formally requested on behalf of the SeaClouds consortium and it stores the last 5 SNAPSHOT releases of SeaClouds platform.
Once the consortium agrees on cutting a new official release, it is then pushed manually to Maven Central.

5.4. How to release SeaClouds

The production of a SeaClouds official release has been kept manual for obvious reasons. In order to release a new version, those are the main steps needed:

`mvn clean install` If everything is ok:

`mvn -DdryRun=true release:prepare -DreleaseVersion=0.7.0-M19 -Dtag=0.7.0-M19 -DdevelopmentVersion=0.8.0-SNAPSHOT` and wait for a message like `Release preparation simulation complete.`

Then, run proper the maven release command
$ mvn release:clean
$ mvn release:prepare -DreleaseVersion=0.7.0-M19 -Dtag=0.7.0-M19 -DdevelopmentVersion=0.8.0-SNAPSHOT
$ mvn release:perform

Finally test the staging repository just created, and promote the release, if everything looks good.

6. Installation and deployment scripts

This paragraph explains how to install SeaClouds platform using Apache Brooklyn. If you are running on a *nix machine, the following commands will be useful.

Download latest Apache Brooklyn release from brooklyn.io
```
$ wget https://www.apache.org/dyn/closer.lua/incubator/brooklyn/apache-brooklyn-0.8.0-incubating/apache-brooklyn-0.8.0-incubating-bin.tar.gz
```
Untar Apache Brooklyn
```
$ tar -zxf apache-brooklyn-0.8.0-incubating-dist.tar.gz
$ cd apache-brooklyn-0.8.0-incubating
```
Download seaclouds-catalog.bom
```
$ wget http://git.io/vCJE9
```
Run Apache Brooklyn with seaclouds catalog
```
$ bin/brooklyn launch --catalogAdd seaclouds-catalog.bom
```

where bom is Brooklyn Object Model.

6.1. Run SeaClouds

Once you run Apache Brooklyn with the seaclouds's catalog.bom added, you will have 2 new applications available

Run SeaClouds on BYON
By selecting, "SeaClouds Platform on BYON" you can deploy SeaClouds on 2 nodes that Apache Brooklyn can reach. 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.

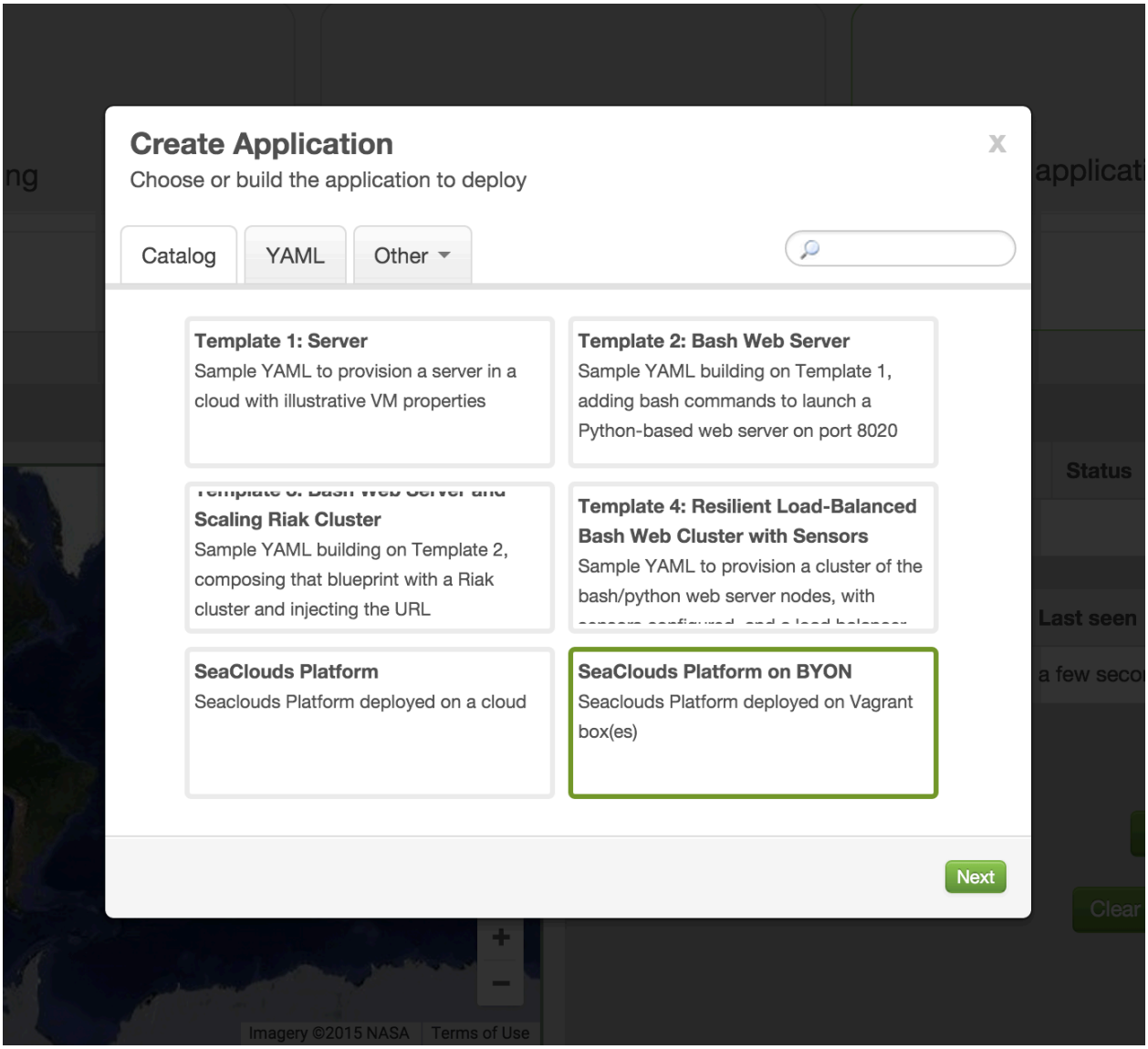

Figure 2: Deploy SeaClouds on your own nodes

For the user's convenience, SeaClouds provides a Vagrantfile to provision 2 Ubuntu 12.04 64bit servers that can be used to deploy SeaClouds on BYON.

Checkout SeaClouds distribution

```
$ git clone git@github.com:SeaCloudsEU/SeaCloudsPlatform.git
```

Move to byon folder

```
$ cd SeaCloudsPlatform/byon
```

Launch Vagrant

```
$ vagrant up
```

Run Apache Brooklyn from byon using:

```
$BROOKLYN_HOME/bin/brooklyn launch --catalogAdd ../seaclouds-catalog.bom
```

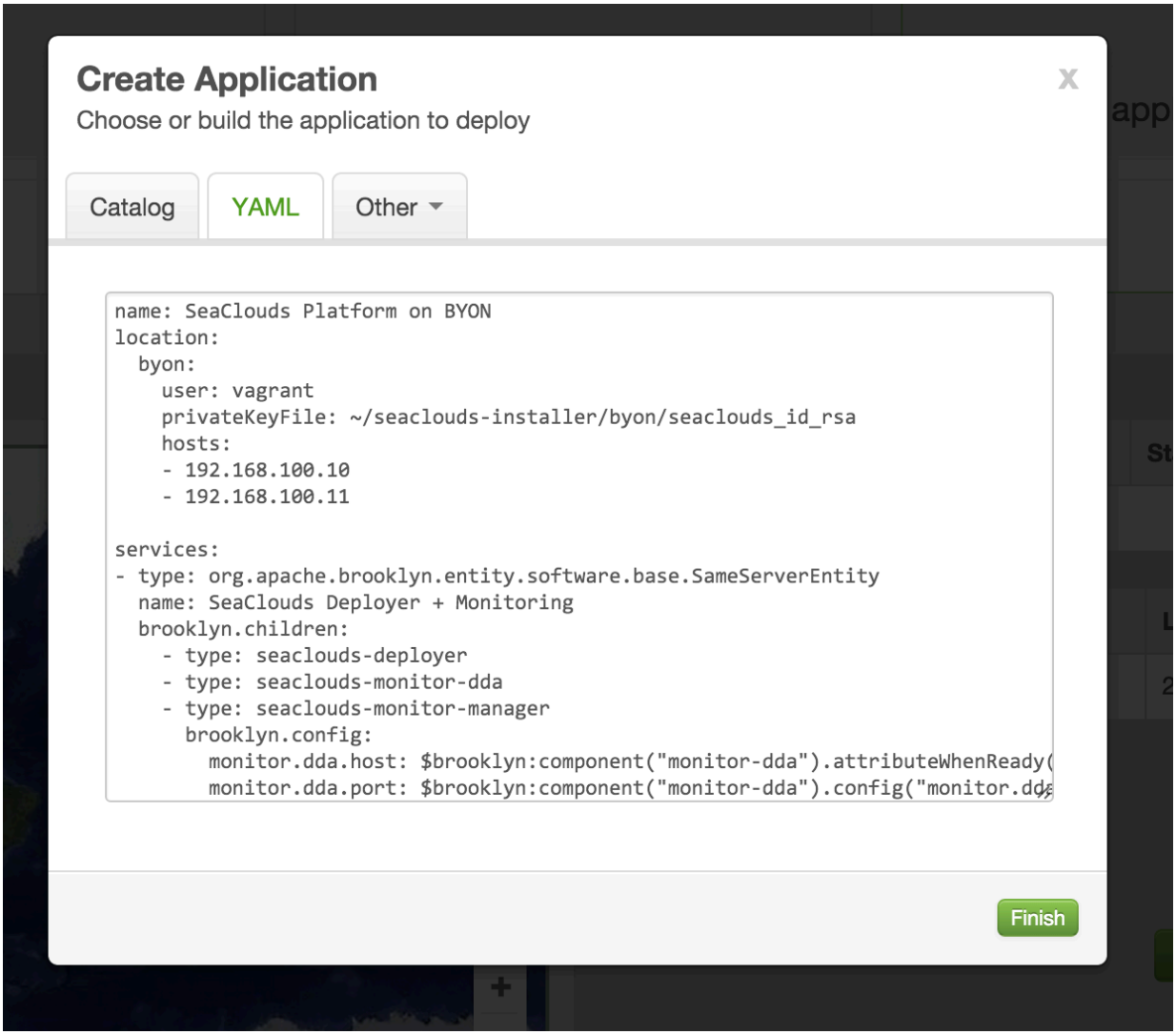

Figure 3: Confirm the default values for BYON

Run SeaClouds on a cloud

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

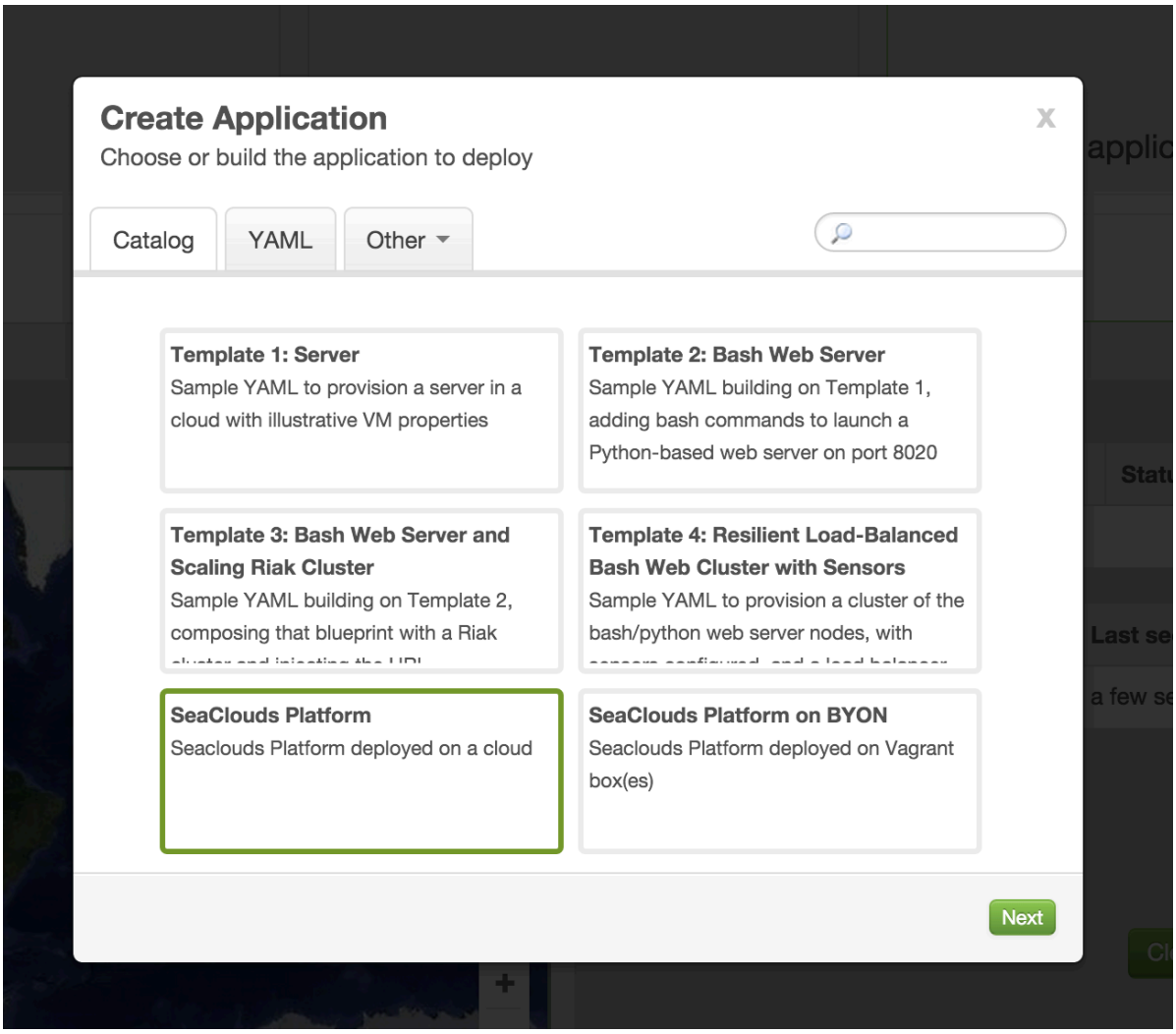

Figure 4: Select deployment on a cloud provider

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

7. The SeaClouds testbed

In this Section we describe the SeaClouds testbed, which has been set up with the aim of having a testing platform available during the development, in particular during the last phases in which, in parallel with the last developments, the process of components integration started.

The main goal of the SeaClouds testbed is to provide a way to test each component, or a set of integrated components, in an isolated fashion. The ideal users of the SeaClouds testbed are mainly the SeaClouds developers and case studies.
The SeaClouds testbed has been accomplished by creating an Amazon Machine Image (AMI) on Amazon EC2, already equipped with all the necessary software dependencies and with a set of utility scripts helping the user to easy set up her own testbed,according to the SeaClouds components she want to test together.
The AMI comes with Ubuntu 14.04 installed and is publicly accessible. It can be found by filtering the public AMI list from the AMI menu of the EC2 console, looking for its name (**SeaClouds Testbed**), as shown in the following picture:

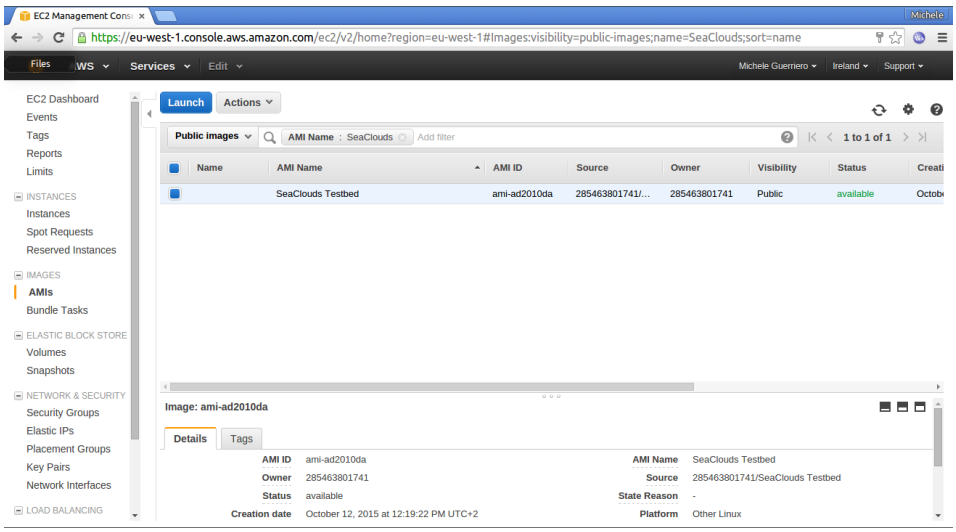

Figure 5: Provisioning of SeaClouds AMI on AWS EC2

When a virtual machine instance is launched from the SeaClouds testbed AMI, it comes with number of folders located in the root of the **ubuntu** user. Each folder is dedicated to a single SeaClouds component and contains all the requirements for running it, from the component artifact to any other further required dependency. Each folder also contains starter and stopper scripts for the associated component. Ideally the SeaClouds testbed user just have to play with these scripts in order to set up her own testbed.
The following picture shows the output of **tree -L 2** issued from the home of the **ubuntu** user.

```
ubuntu@ip-172-31-6-169:~$ tree -L 2
.
├── dashboard
│   ├── config.yml
│   ├── dashboard.jar
│   ├── dashboard.log
│   ├── startDashboard.sh
│   └── stopDashboard.sh
├── deployer
│   ├── brooklyn
│   ├── brooklyn.debug.log
│   ├── brooklyn-dist-0.7.0-incubating
│   ├── brooklyn.info.log
│   ├── deployer.log
│   ├── startDeployer.sh
│   └── stopDeployer.sh
├── discoverer
│   ├── bin
│   ├── lib
│   ├── startDiscoverer.sh
│   └── stopDiscoverer.sh
├── planner
│   ├── plannerconf.yml
│   ├── planner-service-0.8.0-SNAPSHOT.jar
│   ├── startPlanner.sh
│   ├── stopPlanner.sh
│   └── temp
├── sla-service
│   ├── bin
│   ├── lib
│   ├── share
│   ├── sla.log
│   ├── startSlaService.sh
│   └── stopSlaService.sh
└── tower4clouds
    ├── data-analyzer-0.2.3
    ├── manager-server-0.2.3
    ├── startTower4Clouds.sh
    └── stopTower4Clouds.sh

15 directories, 22 files
ubuntu@ip-172-31-6-169:~$
```

An example of a SeaClouds testbed use case, which actually happened during the development, is the ATOS case study which at a given point wants for some reasons to test its application against just the runtime environment of the SeaClouds platform. In this case the developer can launch an instance from the available testbed AMI, access the instance via ssh and run the starter scripts of each desired component; in particular for having the runtime platform running, she will start the SeaClouds SLA, Monitor and Deployer. At this point, each component is up and running and the developer can run its test.

8. Conclusion

This deliverable describes the main technologies but most importantly 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.

The SeaClouds consortium decided to adopt Continuous Integration to significantly reduce integration problems, allowing a team to develop cohesive software more rapidly.
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 maintain 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.
Each integration is verified by an automated build (including test) done by TravisCI to detect integration errors as quickly as possible.

Key practices to keep the CI effective are also:

- fix Broken Builds **immediately**: this is a responsibility of everyone in the team
- keep the Build fast: which is continuously improved with a continuous serie of refinement steps

For more details about those practices, please start from the great http://www.martinfowler.com/articles/continuousIntegration.html