Secure Provisioning of Cloud Services
based on SLA Management

SPECS Project - Deliverable 1.6.2

SPECS Testbed Definitive

Version no.1.0
30 April 2016

The activities reported in this deliverable are partially supported
by the European Community's Seventh Framework Programme under grant agreement no. 610795.

Deliverable information

Deliverable no.:	D1.6.2
Deliverable title:	SPECS Testbed Definitive
Deliverable nature:	Prototype
Dissemination level:	Public
Contractual delivery:	30 April 2016
Actual delivery date:	
Author(s):	Silviu Panica (IeAT)
Contributors:	Alessandra De Benedictis (CeRICT)
Reviewers:	Valentina Casola (CeRICT), Madalina Erascu (IeAT)
Task contributing to the deliverable:	T1.6
Total number of pages:	22

Executive summary

This deliverable is associated with the final implementation of the SPECS Testbed (Task 1.6) whose goal is to provide and to maintain a running platform to host the SPECS services.

The SPECS Testbed provides the SPECS Enabling Platform as already illustrated in Deliverables D1.2, D1.1.1, D1.1.2 and D1.6.1; it offers all the components needed to provide and to manage all SPECS services in their life cycle.
It is currently deployed on the servers of the IeAT partner, being accessible to all partners during the project life. After the end of the project, SPECS Launcher will be still offered for three years as a hosted service using IeAT partner resources. All the other IeAT partner resources will no longer be available.

The design of the SPECS Enabling Platform module is shown in details in the deliverable D1.1.2, the goal of this document is to: (i) report the status of implementation activities, (ii) give the final instructions on how to access and use the Testbed.

Table of contents

Index of figures

Index of tables

1. Introduction

This deliverable reports on the status of development activities related to the SPECS Testbed, and illustrates related installation and usage guides.
The *SPECS Testbed* represents the supporting infrastructure that provides the runtime environment for the *SPECS SLA Platform* and the *Core Services*. In the SPECS Project, the Testbed represents the SPECS *Enabling Platform,* which consists of a software layer (operating system, bootstrap services, management services) responsible for the initial start-up of the SPECS Platform and for the automatic management of the SPECS services, and in a physical infrastructure composed of virtualized resources where the software layer is executed.
As already discussed in D1.6.1, the SPECS Testbed software layer is composed of three main components:

- *Custom Operating System* - a multi-purpose operating system, that runs on top of the infrastructure;
- *Bootstrapper*, a collection of services that bootstrap and customize a custom OS instance for a specific deployment template;
- *Cluster Manager*, a deployment service that is able to automatically bootstrap any service required by the SPECS Platform.

For what regards the physical resources hosting the Enabling Platform services, they are currently provided by two of the SPECS partners, namely IeAT (whose infrastructure had been already used in the previous version of this deliverable), and EMC.
The IeAT infrastructure hosts a HP Eucalyptus cloud stack, which is available for all the partners within the SPECS Project. The EMC infrastructure, instead, is private and used only for the implementation of the EMC use cases, discussed in detail in D5.3 and D5.4. However, the adoption of a second infrastructure demonstrated the independence of the services from the specific underlying technologies.

For what concerns the other advances with respect to the previous version of this deliverable, as illustrated in Section 3.1, in this final iteration we were able to complete the development of the missing/incomplete components (*Discover Sys. component*, *Components Controller*, *Component* and *Node Operational REST APIs*, *Cluster Manager* and *Artifact Repository*), which are now available. With these components, the coverage of requirements was completed.

This document is structed as follows. In Section 2 we describe describe the relationships with other deliverables. Section 3 is dedicated to the SPECS Enabling Platform description, were all the components are described in detail.

2. Relationship with other deliverables

The work presented in this document is related mainly to activities of all Tasks in WP1 and updates with respect to the D1.6.1 deliverable (the preliminary report). In particular, in deliverable D1.2 we discussed the requirements that the *Enabling Platform* should respect, and in deliverables D1.1.1 and D1.1.2 we motivated and designed the solution to provide an independent *Enabling Platform* to manage the whole life cycle of all SPECS Services.

Figure 1 shows the relationships described above. The platform described in this deliverable enables the execution of all SPECS services. However, for readability's sake we do not report here the complete list of all implementation tasks (and related deliverables) that use the services provided by the *Enabling Platform*.

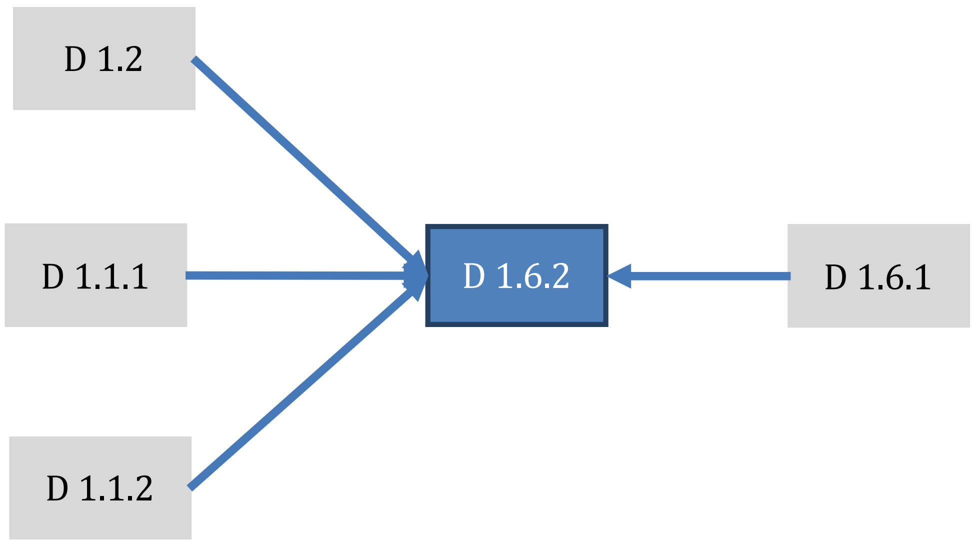


Figure 1.Relationships with other deliverables

3. SPECS Enabling Platform

The Enabling Platform implementation consists of two main parts: the *SPECS Enabling Platform* and the *SPECS Testbed supporting infrastructure*. SPECS Enabling Platform is described in sub-section 3.2. *SPECS Testebed supporting infrastructure* was described in detailed in D1.6.1 and was introduced in Section 1 of this document and

3.1. Status of development activities

In Table 1 we report the list of SPECS software components under development associated with the *SPECS Enabling Platform*, as discussed in D1.2 and D1.1.2, together with the requirements they respectively cover.

Enabling Platform Requirements	SPECS Software Components											
	Custom OS	***Components Logging***	***Node Bootstrapper***	***Node Logging***	***Node discovery***	***Node controller***	***Component Discover Sys.***	***Components Controller***	***Artifact Repository***	***Component Operational***	***Node Operational REST API***	***Cluster Manager***
ENPL_R1				X		X				X		
ENPL_R2			X							X		X
ENPL_R3			X		X					X		X
ENPL_R4					X	X				X		
ENPL_R5					X	X				X		
ENPL_R6	X				X					X		
ENPL_R7	X									X		
ENPL_R8		X										
ENPL_R9								X			X	
ENPL_R10								X			X	
ENPL_R11								X			X	
ENPL_R12							X	X			X	
ENPL_R13							X	X			X	
ENPL_R14							X	X			X	
ENPL_R15		X									X	
ENPL_R16								X			X	
ENPL_R17								X			X	
ENPL_R18							X		X		X	
ENPL_R19									X			

Table 1. SPECS Components related to the Enabling Platform and related requirements

All 19 requirements have been covered (100% requirements coverage) and all the components covering these requirements are already available.

In Table 2, we report the current development status of all SPECS artefacts associated with the *Enabling Platform*.

Module	Artefacts under development	Status
Enabling Platform	Components:Custom OS	Available
	Components:Components Logging	Available
	Components:NodeBootstrapper	Available
	Components:Node Logging	Available
	Components:Node discovery	Available
	Components:Node controller	Available
	Components:Component Discover System	Available
	Components:Components Controller	Available
	Components:Artifact Repository	Available
	Components:Component Operational REST API	Available
	Components:Node Operational REST API	Available
	Components:Cluster Manager	Available

Table 2. Enabling Platform Implementation Status

The second testbed is deployed at the EMC partner and it has been described in D5.3 and D5.4 as an integrated solution with EMC's SPECS Platform adapted for the ViPR technology. EMCs solution is closed source and it is described separately from the open source solutions.

3.2. SPECS Enabling Platform

The *SPECS Enabling Platform* is a software stack that enables the SPECS platform deployment and its management at infrastructure level. It has three main components: *Custom Operating System*, *Bootstrapper* and *Cluster Manager*; these three components are described in the next paragraphs.

Most of the components part of SPECS Enabling Platform were designed and developed based on SPECS requirements. However *Bootstrapper*'s *Resource registration and discovery* service and *Cluster Manager's Cloud Resource Allocator* were derived from a component that was developed under a different research project, called mOSAIC Cloud [5].

3.2.1. Custom Operating System

Compared to D1.6.1 the are no significant improvements over the operating system. CustomOS uses the openSUSE 13.1 linux distribution, as base operating system, with all the packages updated to the latest version. This was necessary in order to address the latest security issues published by the vendor. Moreover, some packages received newer versions that improve also the performance and stability of the *Enabling Platform* dependencies.
Usage information on how to build a custom image and to use it in a cloud environment are available publicly on the BitBucket repository that hosts the mOS sources:

- https://bitbucket.org/specs-team/specs-core-enabling_platform-custom-os

The Custom OS also relies on packages that are customized for SPECS. These packages descriptors are hosted on BitBucket together with a tool called *package factory*, that allows a developer to create and publish its own custom mOS packages. All the details regarding the *package factory* are described on its dedicated repository wiki page:

- https://bitbucket.org/specs-team/specs-core-enabling_platform-custom-os-packaging

3.2.2. Bootstrapper

The Bootstrapper (or *platform bootstrap service*) represents a collection of services that run on top of the *Custom OS (mOS)*. These services are used to customize a mOS instance in order to fulfill the requirements in terms of runtime environment, service deployment, configuration and management etc. It implements two components that were designed in D1.1.2, namely the *Node Bootstrapper* and the *Component Logging*. The architecture is depicted in Figure 2. In particular, these services are responsible for resource registration and discovery, resource configuration and component logging. These responsibilities are described in the subsequent paragraphs.

Resource registration and discovery
This is a service that handles the information about the deployed resources. It uses a distributed system and a custom communication protocol that detailed in [1]. When a resource is first deployed, the resource is registered into the DNS catalog. If a resource requests access to a different platform service hosted on a different node it has to query the DNS catalog to discover the remote resource endpoint. The DNS naming schema allows the registration of the services name (as DNS aliases or hostname entries) and services details using DNS text records. The DNS naming convention is:

[service_unique_name].[service_group].[cluster_instance_id].cloud.domain_name.

This service is flexible and can be adapted for any specific use case scenario.

Resource configuration
Resource configuration is in charge of the customization of the operating environment. This includes the installation and configuration of the software packages needed by the platform. This service uses Chef Client Deployment technology [2]. Chef client customizes the operating system based on a set of predefined templates or recipes. These recipes contain policies that are applied at the operating system level in order to deploy software package and configure the service it delivers. Resource configuration service can be used in two modes: (a) standalone mode, where all the recipes are downloaded locally and Chef client is called directly or (b) central mode, where Chef client connects to a Chef server from where it gets all the configuration templates (Figure 2).

Component logging
The logging service is used to verify different aspects of the platform components and also to debug potential issues that the platform may encounter, in real time. All the logging information is directly exposed using a standard HTTP interface. The default URL for accessing the logging service is: *http://VM_IP_ADDRESS:81/mos/*

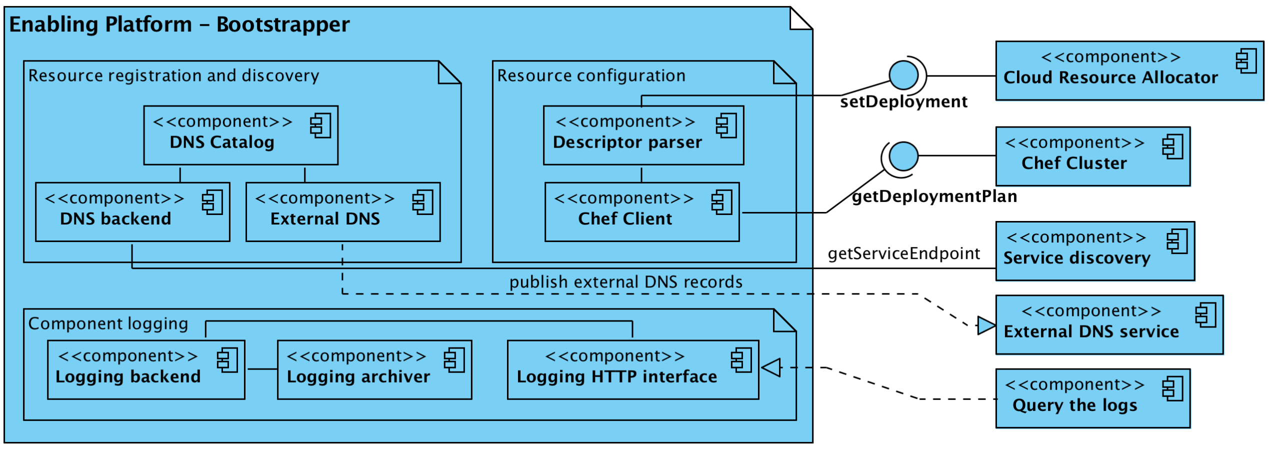

Enabling Platform – Bootstrapper
Resource registration and discovery
<<component>> DNS Catalog
<<component>> DNS backend
<<component>> External DNS
Resource configuration
<<component>> Descriptor parser
<<component>> Chef Client
Component logging
<<component>> Logging backend
<<component>> Logging archiver
<<component>> Logging HTTP interface
<<component>> Cloud Resource Allocator
setDeployment
<<component>> Chef Cluster
getDeploymentPlan
getServiceEndpoint
<<component>> Service discovery
publish external DNS records
<<component>> External DNS service
<<component>> Query the logs

Figure 2. Enabling Platform - Bootstrapper

The Bootstrapper is hosted as a Bitbucket repository available at:

- https://bitbucket.org/specs-team/specs-core-enabling_platform-bootstrapper

3.2.3. Cluster Manager

The Cluster Manager orchestrates the most important aspects of the deployment process, namely the remote resources acquisition process (from the cloud providers, resources used to host the core components) using the cloud resource allocator and the deployment of the platform components using a chef cluster, based on Chef Deployment technology.

Cloud Resource Allocator

Cloud Resource Allocator is the entry point of the *Enabling Platform*. It exposes a REST API and a web user interface for accessing the core functionalities of the *Enabling Platform*. In order to bootstrap a new *Enabling Platform* instance, the user needs to specify: (a) cloud specific information (cloud provider, authentication credentials and security information), (b) cluster information (deployment templates repository, cluster identification) and (c) the configuration of the cluster: number of virtual machines and the deployment templates to be applied. This interface is called *the Cluster Launcher*. *The Cluster Launcher* interface was described in details in D1.4.2. The hosted SPECS Cluster Launcher instance is available at:

- https://dashboard.cloud.specs-project.eu/

Based on the configuration descriptor received from the the user, the *Cluster Launcher* creates a deployment descriptor. This deployment descriptor is sent to the backend service that will acquire the resources from the targeted cloud provider and it will initialize the local deployment process using the *Resource configuration* service, that is described in *The Bootstrapper* in Section 3.2.2. The architecture of the component is described in Figure 3.

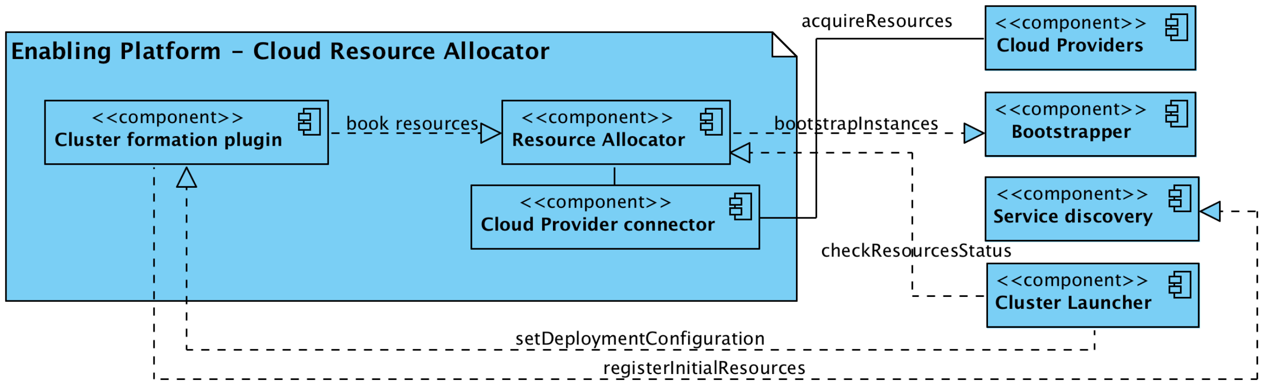

acquireResources
<<component>> Cloud Providers
Enabling Platform – Cloud Resource Allocator
<<component>> Cluster formation plugin
book resources
<<component>> Resource Allocator
bootstrapInstances
<<component>> Bootstrapper
<<component>> Cloud Provider connector
<<component>> Service discovery
checkResourcesStatus
<<component>> Cluster Launcher
setDeploymentConfiguration
registerInitialResources

Figure 3. Enabling Platform - Cloud Resource Allocator

The Cloud Resource Allocator component technical information is described in the Bitbucket repository dedicated wiki:

- https://bitbucket.org/specs-team/specs-core-enabling_platform-cloud-resource-allocator

The Cluster Launcher component is publicly available in the Bitbucket repository:

- https://bitbucket.org/specs-team/specs-core-enabling_platform-cluster-launcher

Chef Cluster

Chef Cluster: The *Enabling Platform* uses Chef technology to automatize the deployment of the supporting platform. Chef uses predefined configuration templates, called recipes, to customize a running environment. These recipes define a list of software requirements that need to be installed and what are the configuration customizations to be applied. For each platform component one or more recipes are used for its deployment. This list of recipes specific to a single component deployment are so-called cookbooks. Chef needs to be manually configured. There is no default method for automatic configuration of the Chef technology. For cluster mode we need one server instance of Chef and minimum one client instance. On deployment phase Chef server defines the credentials used by the Chef clients to communicate with the server. All these information (credentials, chef server endpoint etc.) must be manually provided by the Chef clients. The *Enabling Platform* automatizes the entire process and it does it in a secure way by using simple tokens authentication and authorization between the Chef Cluster nodes. The Chef Cluster consists of multiple components integrated into *The Bootstrapper* and *Custom OS* special packages customizations. The architecture is presented in Figure 4.

By default, the Chef Cluster will automatically upload and register the available cookbooks hosted on Bitbucket repositories available at:

- https://bitbucket.org/specs-team/specs-core-enabling_platform-repository
- https://bitbucket.org/specs-team/specs-core-enforcement-repository

Any new developed cookbooks that are registered under one of these two repositories will be automatically available and registered by the *Chef Cluster*.

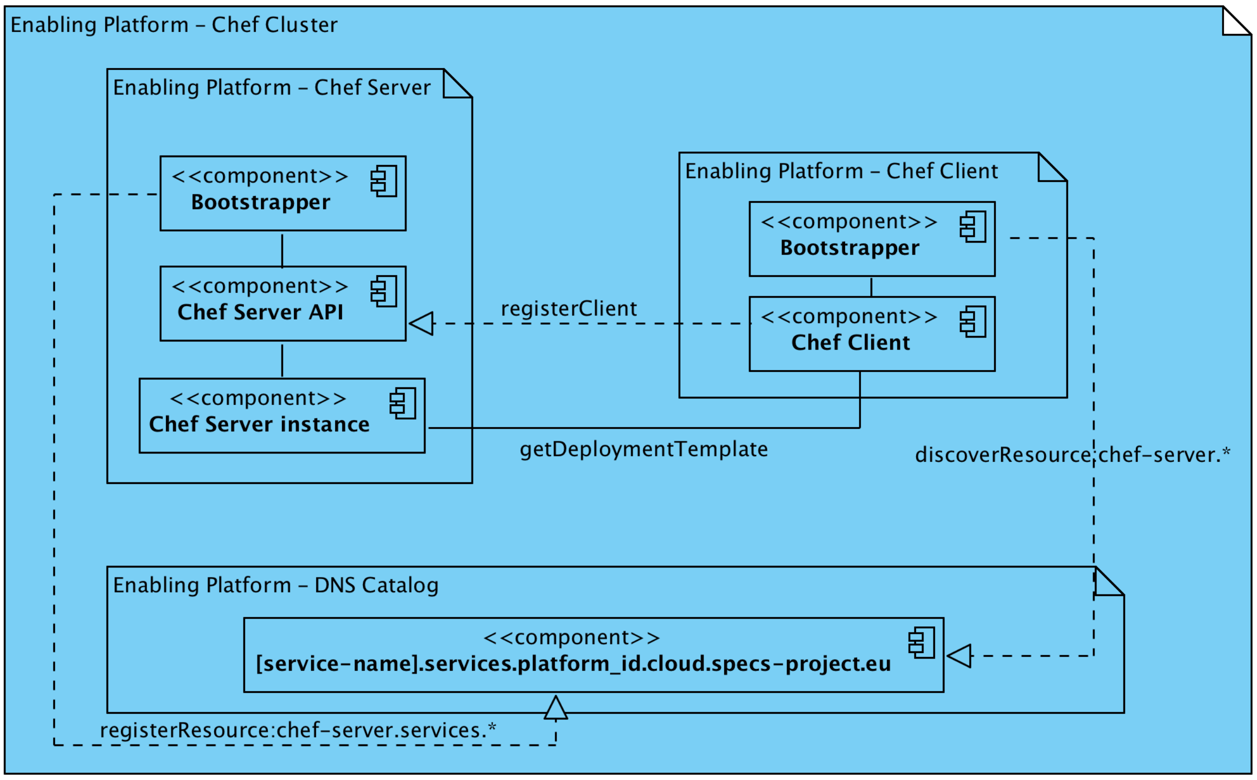

Enabling Platform – Chef Cluster
Enabling Platform – Chef Server
<<component>> Bootstrapper
<<component>> Chef Server API
<<component>> Chef Server instance
Enabling Platform – Chef Client
<<component>> Bootstrapper
<<component>> Chef Client
registerClient
getDeploymentTemplate
discoverResource:chef-server.*
Enabling Platform – DNS Catalog
<<component>> [service-name].services.platform_id.cloud.specs-project.eu
registerResource:chef-server.services.*

Figure 4. Enabling Platform - Chef Cluster

3.3. The testbed supporting infrastructure

3.3.1. The hosted solution

The hosted solution of the SPECS Testbed was entirely described in D1.6.1. The hosted solution uses HP Helion Eucalyptus as a cloud stack on IeAT's infrastructure. The procedures on obtaining access and how to use the solution where detailed in D1.6.1. There are no updates regarding the hosted solution so in the following paragraphs we will mainly focus on the *local solution* installation and usage details. With respect to the *hosted solution* in Section 3.4 we present a brief overview over the usage of the physical resources.

3.3.2. The local solution

The local solution represents a custom deployment of the *Enabling Platform* on a different cloud provider than the official supported testbed, hosted by IeAT. The local solution deployment has the following requirements:

- the use/registration of the CustomOS image with the official SPECS Custom OS packages repositories enabled (CustomOS image comes with the repositories enabled by default);
- the installation of the Cloud Resource Allocator;
- the deployment of the Cluster Launcher interface on top of a running HTTP server;

3.3.2.1. Installation

The installation assumes that the user has access to a private/public cloud provider or resources and is able to register new cloud images and to acquire resources from a supported cloud provider.

3.3.2.2. Custom OS image upload

In order to host the components required by the *Enabling Platform*, a Custom OS image has to be imported and made available within the targeted cloud provider. The official Custom OS image supported by SPECS is available at:

- ftp://ftp.specs-project.eu/public/custom-os/latest

The image needs to be bundled using the cloud stack specific tools. The image registration process is specific to the cloud stack instance used for deployment. The image registration process is usually described in documentation of the cloud stack software or cloud provider documentation.
After successfully importing the Custom OS image, the user must start a virtual machine (VM) using the Custom OS image. The VM should have at least 2GB of memory and 5GB of virtual hard drive.

3.3.2.3. Cloud Resource Allocator

On the newly created VM the user must install the Cloud Resource Allocator (RA) component. The installation steps of the RA component are available on the dedicated Bitbucket repository wiki space:

- https://bitbucket.org/specs-team/specs-core-enabling_platform-cloud-resource-allocator

The Cloud Resource Allocator was designed to work in a Unix/Linux environment that has support for the following requirements:

- Python Programming Language - version 2.7.x
- Python libraries: Pip, Apache Libcloud 0.12.4, CherryPy 3.2.4 and SOAPpy;
- Mercurial for repository download;

Assuming that the running environment meets the above requirements, the following command lines will download, install and configure Cloud Resource Allocator locally:

```
     mkdir -p /opt/specs-cloud-resource-allocator
     cd /opt/specs-cloud-resource-allocator
     hg clone https://bitbucket.org/specs-team/specs-core-
enabling_platform-cloud-resource-allocator .
     pip install -r requirements.txt
```

To control the service, use the following command:

```
     /opt/specs-cloud-resource-allocator/run.sh [start|stop]
```

The basic customizations that should be done are:

Add the cloud image details for each supported cloud providers (by default, AmazonEC2 and HP Eucalyptus are enabled); a cloud image is represented by the image id (a random string generated by the cloud provider). Edit the */opt/specs-cloud-resource-allocator/conf/images.conf* file by adding a JSON structure like:

```
     [
         {
             "description": "Custom OS 4.2.0 64bit",
             "name": "CustomOS-4.2.0-001",
             "platform": 64,
             "supported": [
                 {
                     "image": "emi-65c3e382",
                     "kernel": "",
                     "provider": "ieat-euca",
                     "ramdisk": ""
                 }
             ],
             "version": "4.2.0"
         }
     ]
```

Required fields are:

- *name*: custom string without spaces that represents the name displayed in the *Cluster Launcher* interface;
- *description*: image description;
- *platform*: 32 or 64 (bits);
- *supported*: a list of providers for which the cloud image with *name* is supported:
    - *image*: represents the image id provided by the cloud provider;
    - *provider*: the provider on which the cloud image is registered (accepted values: *ieat-euca* and *ec2-eu-west*);

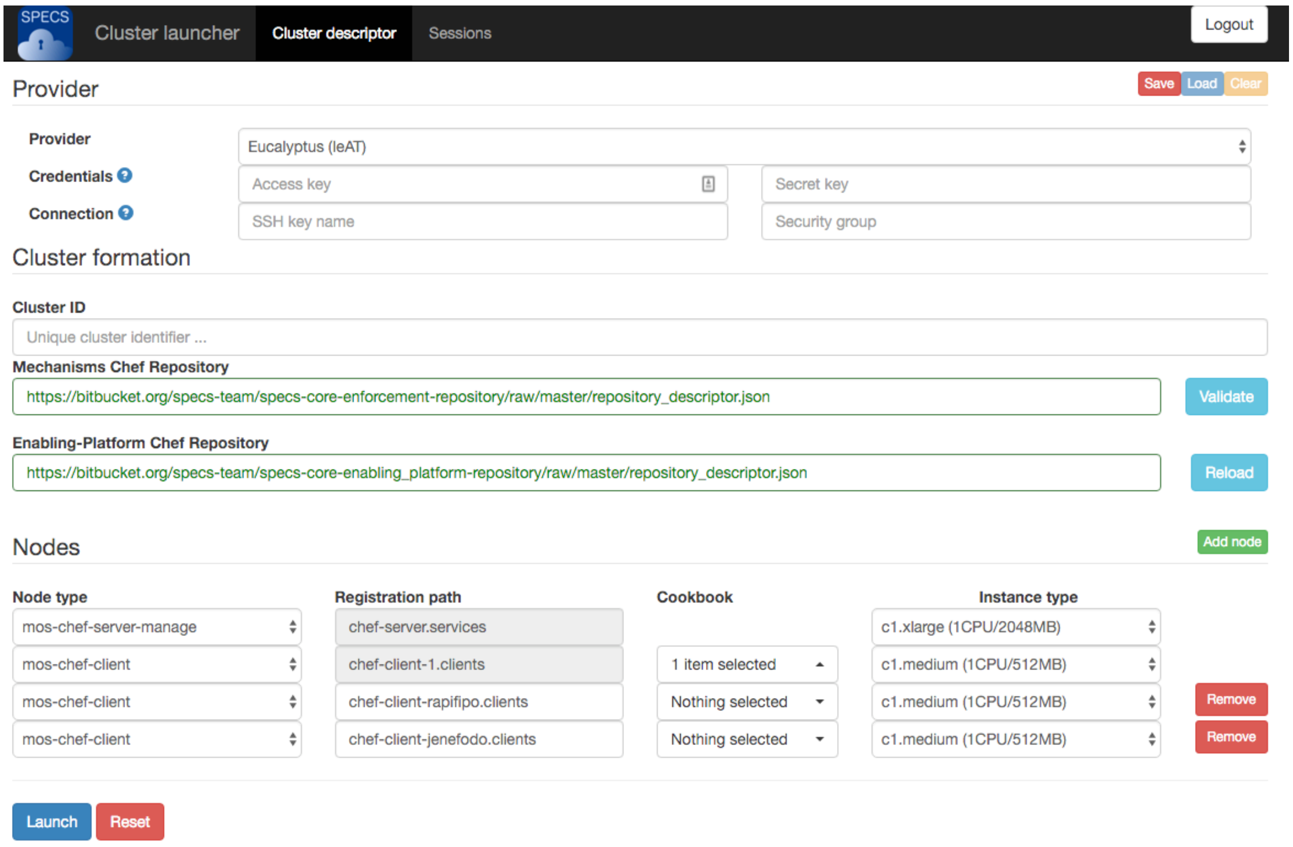
Configure the list of supported applications. An application represents a cloud image that is customized with a specific set of packages and transformed in a node with a certain role. The default roles are: Chef Server and Chef Client. The only customization that needs to be applied is to edit */opt/specs-cloud-resource-allocator/conf/applications.conf* and to add to field *supported* the corresponding image name defined in the *images.conf* file above.

Add a list of users authorized to use the cloud resource allocator. Cloud Resource Allocator uses a simple authentication using a token. The default token is registered in */opt/specs-cloud-resource-allocator/conf/users.conf*. After each modification over the configuration files the service needs to be restarted in order to reload the configuration parameters.

Next, in order to use the Cloud Resource Allocator the management interface has to be deployed on the same VM. Follow the next instructions to deploy the Cluster Launcher interface.

3.3.2.4. The Cluster Launcher interface

The last step, in deploying a custom *Enabling Platform* instance, is to have the *Cluster Launcher* interface installed. *Cluster Launcher* interface is written as a plain web-based application that requires only a running HTTP server, for serving its content, and a JavaScript compatible browser to use the interface. *Cluster Launcher* must be installed on the same VM where *Cloud Resource Allocator* is installed. The *Cluster Launcher* interface is shown in Figure 5 and all the information on how to use the interface where explained in detail in D1.4.2.



Provider

Provider: Eucalyptus (IeAT)

Credentials: Access key | Secret key

Connection: SSH key name | Security group

Cluster formation

Cluster ID: Unique cluster identifier ...

Mechanisms Chef Repository: https://bitbucket.org/specs-team/specs-core-enforcement-repository/raw/master/repository_descriptor.json

Enabling-Platform Chef Repository: https://bitbucket.org/specs-team/specs-core-enabling_platform-repository/raw/master/repository_descriptor.json

Nodes

Node type	Registration path	Cookbook	Instance type
mos-chef-server-manage	chef-server.services		c1.xlarge (1CPU/2048MB)
mos-chef-client	chef-client-1.clients	1 item selected	c1.medium (1CPU/512MB)
mos-chef-client	chef-client-rapifipo.clients	Nothing selected	c1.medium (1CPU/512MB)
mos-chef-client	chef-client-jenefodo.clients	Nothing selected	c1.medium (1CPU/512MB)

Figure 5. Cluster Launcher - Web Interface

To install a HTTP server, that will host the launcher interface, the user must run on the VM console, logged in as root:

```
        zypper install apache2
```

Also be sure you have the Mercurial versioning tool installed for cloning the source files from the repository.
Next, the *Cluster Launcher* has to be deployed and configured locally using the HTTP server installed above:

```
        mkdir -p /srv/www/dashboard.cloud.mydomain.eu/
        cd /srv/www/dashboard.cloud.mydomain.eu/
        hg clone https://bitbucket.org/specs-team/specs-core-
enabling_platform-cluster-launcher .
```

The HTTP server has to be configured to deliver the content of the Cluster Launcher interface by adding a new configuration file to the HTTP server. Add a new file to */etc/apache2/vhosts.d/* directory with name *dashboard.cloud.mydomain.eu* having the following content:

```
        <VirtualHost *:80>
                ServerAdmin email@address
                ServerName dashboard.cloud.mydomain.eu
                DocumentRoot /srv/www/dashboard.cloud.mydomain.eu
                <Directory />
                        Options FollowSymLinks
                        AllowOverride None
                        Require all granted
                </Directory>
                <LocationMatch "/(data|conf|bin|inc)/">
                    Order allow,deny
                    Deny from all
                    Satisfy All
                </LocationMatch>
                <Directory /srv/www/dashboard.cloud.mydomain.eu>
                        Options Indexes FollowSymLinks MultiViews
                        AllowOverride All
                        Order allow,deny
                        allow from all
                </Directory>

                ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
                <Directory "/usr/lib/cgi-bin">
                        AllowOverride None
                       Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
                        Order allow,deny
                        Allow from all
                </Directory>

                LogLevel warn
                CustomLog /var/log/apache2/dashboard.cloud.mydomain.eu-access.log
        combined
                ErrorLog /var/log/apache2/dashboard.cloud.mydomain.eu-error.log

                ProxyPass /cloud http://127.0.0.1:5003/cloud
                ProxyPassReverse /cloud http://127.0.0.1:5003/cloud

        </VirtualHost>
```

Hostname *dashboard.cloud.mydomain* must be replaced with a custom the subdomain set up to point out the the *Cluster Launcher interface* VM IP address. We recommend to keep the current naming convention prefix: *dashboard.cloud.* . Restart the HTTP server with *systemctl restart apache2* command.

3.3.2.5. The Chef Cluster

Chef cluster doesn't need any special installation requirements if the above steps were completed. The Chef Cluster is setup by the *Cloud Resource Allocator* by applying specific deployment plans on the newly acquired resources. The deployment plan is generated relying on the information provided in the *Cluster Launcher* interface.

3.3.2.6. Usage

If all the components described in the *Installation* section where successfully installed, the Cluster Launcher interface can be used to deploy a custom *Enabling Platform* and SPECS Platform running on your local cloud stack. The interface can be accessed directly using a JavaScript enabled browser pointing out the URL address of the VM used to deploy the components. Using the naming convention from the example described above, the URL is:

- http://dashboard.cloud.mydomain/

The complete tutorial on how to use the interface where presented in D1.4.2 (Section 3.3) together with a tutorial on how the *Cluster Launcher* interface can be used to bootstrap a new SPECS Platform.

3.4. *Usage of the physical resources*

In this sub-section we present the hardware resources used for testing and executing the SPECS Platform. In Table 3 we report the VM types and their maximum available number, available for hosting SPECS experiments related to development, testing and integration activities.

Instance type	CPU	RAM Memory (MB)	Disk storage (GB)	Maximum available
m1.small	1	256	5	36
m1.medium	1	512	5	36
c1.medium	1	512	9	18
m1.large	1	1024	15	18
c1.xlarge	1	2048	9	18
m2.2xlarge	2	2048	9	18
m1.xlarge	1	1024	20	12
m2.xlarge	1	2048	10	12
m3.xlarge	2	2048	25	12
m3.2xlarge	2	4096	15	8
cc1.4xlarge	4	4096	20	8

Table 3. IeAT HP Helion Eucalyptus available VM types.

Base on the usage pattern of the resources, correlated with the SPECS Modules hosted by these resources, we defined a list of resource templates. These templates contain the number of VMs (of different types) required to successfully host a particular SPECS module. The list of templates is outlined in Table 4.

Template	VM Type	Number of VMs	Description
SPECS Platform	m2.2xlarge	1	Chef Server - Node
	m1.xlarge	1	SPECS Components
WebContainers	m1.large	3	SPECS Web Containers components
Storage	m1.xlarge	3	Storage services

Table 4. SPECS Testbed resource requirements templates.

Based on these templates, we compute the physical resources required to conduct different experiments during the project life-cycle (Table 5).

Consumer	Template / VM Type	No.	CPU	RAM (MB)	HDD (GB)
apps.specs-project.eu	SPECS Platform	1	3	3072	29
Partners experiments	SPECS Platform	2	6	6144	58
	m1.large	6	6	6144	90
	c1.xlarge	3	3	6144	27
Integration	SPECS Platform	1	3	3072	29
	m1.large	3	3	3072	45
TOTAL			**24**	**27648**	**278**

Table 5. SPECS resource requirements for different experiment types.

The physical resources usage is reported by the HP Helio Eucalyptus. At the project level, from 01.01.2015 up to 27.04.2016, the consumed resources is outlined in Table 6. These numbers demonstrate that the supporting platform was intensively used to support SPECS experiments made on development, testing and integration activities.

Number of VMs started	VM hours consumption	Network IN traffic (GB)	Network OUT traffic (GB)	Disk Usage Read (GB)	Disk Usage Write (GB)
484	114816	2439	681	4273	11122

Table 6. SPECS physical resources usage report.

4. Conclusions

This document presents the final implementation of the SPECS Testbed, which provides the environment to configure and run the SPECS Enabling Platform, on top of which all the SPECS services are executed.

In this document, we presented the status of implementation activities and reported the final coverage of the requirements that were identified during the requirement analysis and design phases of the Enabling Platform. As illustrated, all the requirements are now covered and all artifacts are available. Finally, two testbeds are currently available, represented by the IeAT and EMC infrastructures.

The performance and scalability discussion was not tackled in this document because the *Enabling Platform* doesn't need to be scalable and its performance is not critical. The *Enabling Platform* is used to bootstrap the SPECS Platform by the resource provider. In this way the workload on the *Enabling Platform* is reduced and the response time is not critical as it is normally used only once per a SPECS Platform instance setup.

The *Enabling Platform* tackled the problem of autonomic deployment of distributed software systems. In [3] we published an analysis over the existent solutions from different point of views: open-source versus enterprise, hosted solutions versus deployable solutions, etc. The analysis conducted in [3] and [4] outlined that the existent technologies are inefficient when distribution, heterogeneity, scalability, dynamics and openness are primary concerns. With our approach we tried and partially managed to address the above requirements. As a proof of concept, our proposed solution, namely the *Enabling Platform*, manages to automatize the setup process of the popular Chef deployment solution, that SPECS is using to automatize the deployment of the SPECS Platform. The Chef deployment technology has a complex deployment procedure for its components setup. With the *Enabling Platform* (the *Chef Cluster* component) we managed to automatize the entire process and by this to hide the complexity from the End-user or owner perspective. This approach is new and it can be reused in other contexts with less effort.

In conclusion, Table 7 reports the main outcomes related to the activities of Task 1.6.

Outcome description
Enabling platform final implementation
SLA Platform implementation and testbed available
SLA platform interface (SPECS Dashboard)

Table 7. Task 1.6 outcomes

5. Bibliography

[1] Panica, S. & Petcu, D. (2013), Distributed Resource Identification Service for Cloud Environments. *15th International Symposium on Symbolic and Numeric Algorithm for Scientific Computing* (pp. 448-453). Timisoara: SYNASC 2013.

[2] Chef technology used for automatic deployment, http://chef.io/, last accessed 04.2016

[3] Silviu Panica, Dana Petcu, "Unattended deployment of enabling platforms for Cloud-based applications", in AINA 2016, 30th IEEE International Conference on Advanced Information Networking and Applications, CCPI Workshop, Switzerland, 23-25 March, 2016, in press.

[4] J.-P. Arcangeli, R. Boujbel, and S. Leriche, "Automatic deployment of distributed software systems: Definitions and state of the art," Journal of Systems and Software, vol. 103, pp. 198 – 218, 2015

[5] mOSAIC Cloud, FP7 EU funded project, http://www.mosaic-cloud.eu/