Automatic, multi-grained elasticity-provisioning for the Cloud

CELAR System Prototype

Deliverable no.: 6.4
30/09/2015

[Re-submission 29/01/2016]

CELAR is funded by the European Commission DG-INFSO
Seventh Framework Programmed, Contract no.: 317790

Table of Contents

1 Introduction 6

2 Integration between Platform components 6
2.1 CAMF (c-Eclipse) OSGi bundles via project's P2 repository 6
2.2 SlipStream extensions for horizontal and vertical scalability 6
2.3 VM scaling actions on Flexiant and Okeanos 8
2.4 Integration of SlipStream into CELAR flow 9
2.5 TOSCA translation into SlipStream modules 9
2.6 DMM bootstrap and integration with JCatascopia 9

3 Build and integration process 10
3.1 Continuous Integration server 10
3.2 Maven repository 10
3.3 RPM artifacts and YUM repo 10
3.4 Automated Platform deployment for integration and system level tests 11

4 Testing of the CELAR Platform 12
4.1 Chosen testing approaches and strategy 12
4.2 Test plan 14
4.3 Per-component functional and inter-component integration testing 15
4.3.1 DMM and JCatascopia 15
4.3.2 DMM and CELAR Orchestrator 16
4.3.3 CELAR Orchestrator and SlipStream 16
4.3.4 SlipStream horizontal and vertical scaling framework 17
4.3.5 Vertical scaling in SlipStream Okeanos and Flexiant connectors 17
4.3.6 CELAR Manager and SlipStream 18
4.3.7 CAMF and CELAR Manager 19
4.3.8 Information System and CELAR Manager 19
4.3.9 High availability of CELAR Manager 19
4.4 System level testing 20
4.4.1 System testing with DataPlay on Flexiant 20
4.4.2 System testing with SCAN on Okeanos 21
4.5 Results and Conclusions 21

5 The CELAR Platform readiness, delivery and support 21
5.1 Technology Readiness Levels 21
5.2 Delivery 22
5.3 Support 23

6 Conclusions 24

7 References 24

List of Figures

Figure 1 Platform deployment automation with SlipStream. 12

List of Tables

Table 1 The available user application hooks in SlipStream. 7
Table 2 Set of IaaS actions supported by SlipStream and implemented by Flexiant and Okeanos connectors. 8
Table 3 Chosen testing approaches and their scopes across WPs. 13
Table 4 TRL levels of the components of the CELAR Platform, cloud platforms the project was using and the project's pilot applications. 22

List of Abbreviations

DMM: Decision Making Module

MELA: Multi-Level Metric Evaluation Module

CAMF: Cloud Application Management Framework

Deliverable Title	CELAR System Prototype
Filename	celar_D6.4_resubmission.pdf
Author(s)	Konstantin Skaburskas
Date	29/01/2016

Start of the project:	October 2012
Duration:	3 years
Project coordinator organisation:	Athena Research and Innovation Center in ICKT
Deliverable title:	CELAR System Prototype
Deliverable no.:	6.4
Due date of deliverable:	30/09/2015
Actual submission date:	30/09/2015
Due date of resubmission:	01/02/2016
Actual resubmission date:	29/01/2016

Dissemination Level

X	PU	Public
	PP	Restricted to other programme participants (including the Commission Services)
	RE	Restricted to a group specified by the consortium (including the Commission Services)
	CO	Confidential, only for members of the consortium (including the Commission Services)

Deliverable status version control

Version	Date	Author
0.1	28/09/2015	Konstantin Skaburskas
1.0	29/09/2015	Konstantin Skaburskas
1.1 (resubmission)	27/01/201	Konstantin Skaburskas, Giannis Giannakopoulos, Dimitrios Tsoumakos

Abstract

Final Prototype of the CELAR System. This document describes the build, integration, testing and the delivery processes of the CELAR System. The document also acts as a complementary material to the code and binary artifacts of the Final Prototype of the CELAR Platform, which are subject to be released to the public domain.

Keywords

CELAR, integrated system, Elasticity provisioning, monitoring, application description, CELAR Server, CELAR Orchestrator, Decision-Making module

1 Introduction

This deliverable presents the final CELAR Platform integration and testing points during the last year of the project. It also briefly presents highlights of the automated integration process, as well as the release, delivery and support strategies.

We've identified and present the Technology Readiness Levels (TRL) [1] as a whole for the CELAR Platform and per component (where applicable).

For the detailed functional description of the Platform components please see the respective project deliverables D1.2 CELAR Architecture [2], D2.3 c-Eclipse Framework (CAMF) [3], D3.3 CELAR Manager and Orchestrator [4], D4.3 Multi-Level Metric Evaluation Module, JCatascopia and Info System [5] and D5.3 Decision Making Module (DMM) [6].

2 Integration between Platform components

2.1 CAMF (c-Eclipse) OSGi bundles via project's P2 repository

c-Eclipse is now an official Eclipse Technology project and is now called the Cloud Application Management Framework (CAMF) (see [7] and D2.3). CAMF is in the process of being officially integrated with the build and release process of the Eclipse community to be able to be distributed via Eclipse Marketplace.

For the project the original *git* repository hosted on GitHub is still being used:

https://github.com/CELAR/c-Eclipse

Jenkins and Nexus were configured for automating the build of all CAMF OSGi bundles. At the moment for the scope of the CELAR Project the following repo should be used for installation of the CAMF

http://downloads.celarcloud.eu/ceclipse/p2/

The P2 repository is refreshed every time a new build is invoked.

2.2 SlipStream extensions for horizontal and vertical scalability

In the second year of the project the main outcome of the collaboration with the project was the implementation of the horizontal scaling feature in SlipStream. This feature was validated and released with the production version of SlipStream as the first class citizen and used, for example, by CERN for running production computing activities in the scope of EU HelixNebula project [8].

During the final year of the project the vertical scaling feature was added to SlipStream. The SlipStream connector framework was extended to support the following scaling actions:

- resize CPU/RAM or instance type
- attach disk
- detach disk.

To ease the request of the scaling actions from the SlipStream clients the following set of CLIs was introduced (see [9]):

ss-node-add [options] <run> <node-name> [<number>]
ss-node-remove [options] <run> <node-name> <ids> [<ids> ...]
ss-scale-resize [options] [--cpu <num>, --ram <num>]|[--instance-type <type>] <run> <node-name> <ids> [<ids> ...]
ss-scale-disk [options] [--attach <GB> | --detach <device>] <run> <node-name> <ids> [<ids> ...]

In the user application space, to be able to prepare to and make use of the VM changes due to the vertical scale actions, SlipStream added pre- and post-scale scripts (hooks). Table 1 presents the full list of the available to the user hooks, corresponding scale action they are bound to, and when they are executed.

Table 1 The available user application hooks in SlipStream.

Script	Action	When Executed
"On VM Add"	*horizontal scale up*	after addition of new VMs on all the VMs of the deployment except the ones that were just added.
"On VM Remove"	*horizontal scale down*	after the removal of the requested VMs on all the VMs left in the deployment.
"Pre-Scale"	*horizontal scale down*	before VMs removal action, on the VMs targeted for the removal, and therefore, before the "On VM Remove" script
"Pre-Scale"	*vertical scale up/down*	before any vertical scaling action (VM resizing or attaching/detaching of extra disk) on the VMs that are subject to the scaling action.
"Post-Scale"	*vertical scale up/down*	after any vertical scaling action (VM resizing or attaching/detaching of extra disk) on the VMs that are subject to the scaling action.

The corresponding documentation about the usage of the scaling actions in SlipStream Cloud Application Management Platform can be found at [9].

In most of the cases, the application of the vertical scale actions requires reboot of the targeted VM. The SlipStream node and orchestrator executors (agents running on the VMs and responsible for provisioning and orchestration of the user applications) were updated to support the reboot of the VM.

Applying the lessons learned from the usage of SlipStream with mutable and scalable runs the following was added and improved

- updated provisioning framework and orchestration,

- increased reliability and fault tolerance of the provisioning and orchestration of the scalable deployments.

The details of the changes can be found on the SlipStream release notes page [10].

2.3 VM scaling actions on Flexiant and Okeanos

The results around the horizontal scaling achieved during the second year of the project were hardened by updating the Flexiant and Okeanos connectors. This was made possible thanks to starting running the deployments of the project's pilot applications from the WP7 and WP8 (DataPlay and SCAN respectively).

In the third year, following the updates of the SlipStream connector framework and addition of the ability to run the vertical scaling actions, the Flexiant and Okeanos connectors were extended to support: resize CPU/RAM or instance type, attach/detach extra disk.

The following Table 2 summarizes the IaaS actions supported by SlipStream and developed in the connectors. It's clear, that by the end of the project the collaboration was able to achieve promised results regarding the implementation of the generic and scale IaaS actions.

Table 2 Set of IaaS actions supported by SlipStream and implemented by Flexiant and Okeanos connectors.

Action	Flexiant	Okeanos
Authenticate to IaaS	Yes	Yes
List VMs	Yes	Yes
Start VM (used in horizontal scale up)	Yes	Yes
Terminate VM (used in horizontal scale down)	Yes	Yes
Resize CPU/RAM/Instance Type	Yes	Yes
Attach disk	Yes	Yes
Detach disk	Yes	Yes
Start VM with volatile extra disk	Yes	Yes
Start VM with persistent extra disk	No	No
Build new image	Yes	Yes
Request public or private network for VM	Yes	Yes

All of the implemented actions from Table 2 were tested and validated against the actual pilot applications from WP7 and WP8 both on Flexiant and Okeanos Clouds.

The source code for both Flexiant and Okeanos connectors was open sourced and is publicly available on GitHub.

Flexiant:
https://github.com/CELAR/SlipStreamConnector-Fco

Okeanos:
https://github.com/CELAR/SlipStreamConnector-Okeanos

2.4 Integration of SlipStream into CELAR flow

CELAR Manager and CELAR Orchestrator that are responsible for (i) starting the user deployments, (ii) tracking their overall progress and (iii) issuing elastic actions against cloud provisioner were updated and now support request of all the types of the application or VM actions exposed via SlipStream API [11]. Both components share the same codebase. The following functionality was updated and implemented

CELAR Manager (to SlipStream)

- define project (first level grouping for the modules)
- define image module (holds cloud ID and app. deployment scripts)
- define deployment (binds together the images for deployment)
- run a deployment (representation of the running VMs and application)
- monitor state of the run (query the run for the state of the deployment)

CELAR Orchestrator

- accept elastic actions from DMM (decisions made to scale the deployment)
- issue elastic actions against SlipStream
  - VM add / remove (horizontal scale)
  - VM resize CPU/RAM/instance type (vertical scale)
  - VM disk attach/detach

More details can be found in D3.3 [4].

2.5 TOSCA translation into SlipStream modules

CAMF interfacing with CELAR Manager produces user deployment projects description in TOSCA format. This format is not understood by SlipStream. A special translation module from TOSCA to SlipStream's project/image/deployment modules was considerably enhanced and updated to provide integration of the enhanced SlipStream into the CELAR flow. The major third year extensions include the translation of the user application elasticity hooks.

2.6 DMM bootstrap and integration with JCatascopia

To be able to start making scalability related decisions DMM should be bootstrapped with the topology of the deployed application and elasticity constraints, as well as during the application runtime supplied with the different monitoring metrics from VMs and the application itself. The bootstrap of the DMM is achieved by providing it with the endpoint of CELAR Manager and the ID of the started application. DMM then, fetches the required application related info from the CELAR Manager and prepares to

pull post-processed metrics from the MELA coupled with JCatascopia cloud and application monitoring service.

The installation of the components is done in advance as part of the release via CELAR Orchestrator VM image appliance (see section 4.), and the configuration for the components is supplied at runtime.

For more details please see deliverable D5.3 [6].

3 Build and integration process

There were no considerable changes in the code hosting, development, build and integration process (see D6.2 [12]). In the third year, the consortium continued using the same *git* repository hosting service (GitHub) under the organization CELAR https://github.com/CELAR

After a feature or fix is locally developed and unit tested, developer pushed the code to the corresponding *git* repo on GitHub.

3.1 Continuous Integration server

The project's Jenkins Continuous Integration service http://snf-153388.vm.okeanos.grnet.gr was configured with a number of jobs to poll the corresponding GitHub repositories and if changes were detected the code was pulled from GitHub and automatically built.

3.2 Maven repository

As the result of the build process, the produced binary artifacts were pushed to the project's maven repository http://snf-175960.vm.okeanos.grnet.gr/nexus (Sonatype Nexus is used).

Below is the configuration of the CELAR Platform's maven Snapshots and Releases repositories

```
<distributionManagement>
 <snapshotRepository>
  <id>snapshots</id>
  <url>http://snf-175960.vm.okeanos.grnet.gr/nexus/content/repositories/snapshots</url>
 </snapshotRepository>
</distributionManagement>

<distributionManagement>
 <repository>
  <id>releases</id>
  <url>http://snf-175960.vm.okeanos.grnet.gr/nexus/content/repositories/releases</url>
 </repository>
</distributionManagement>
```

3.3 RPM artifacts and YUM repo

The project chose to use RedHat based systems as the target distribution. Therefore, OS level binary artifacts generated by the corresponding maven plugins are RPMs. Nexus has a great plugin that is capable of detecting RPM artifacts that get uploaded to the repository and builds or updates YUM repo with all the RPMs in the corresponding Nexus repository. Below is the configuration of the CELAR YUM snapshots and releases repositories

```
[CELAR-snapshots]
name=CELAR-snapshots
baseurl= http://snf-175960.vm.okeanos.grnet.gr/yum/snapshots
enabled=1
protect=0
gpgcheck=0
metadata_expire=30s
autorefresh=1
type=rpm-md

[CELAR-releases]
name=CELAR- releases
baseurl= http://snf-175960.vm.okeanos.grnet.gr/yum/releases
enabled=0
protect=0
gpgcheck=0
metadata_expire=30s
autorefresh=1
type=rpm-md
```

3.4 Automated Platform deployment for integration and system level tests

On average, the above process takes around 10 minutes and its clear to see that at this point the Platform developers can already deploy on the test machines the respective Platform components and perform integration or system level QA process.

During the third year of the project we updated and used the previously created set of the per component deployment scripts to automate the testing and validation process of the Platform being developed. The scripts for all the components that constitute the CELAR Platform are publicly available on GitHub

https://github.com/CELAR/celar-deployment

To facilitate and streamline development, integration, testing and validation of the CELAR Orchestrator components constituting the core of the CELAR Elasticity framework (see D3.3), the project was actively using the automated build of the new CELAR Orchestrator image. The build image feature is provided by SlipStream. Building new image on Flexiant or Okeanos takes approximately 15 min, after which the new image with the latest installed CELAR Orchestrator components is ready to be used for validation or in testing of the elasticity actions for WP7 and WP8 applications (DataPlay and SCAN).

More details on the Platform testing is available in the section 4 of the deliverable.

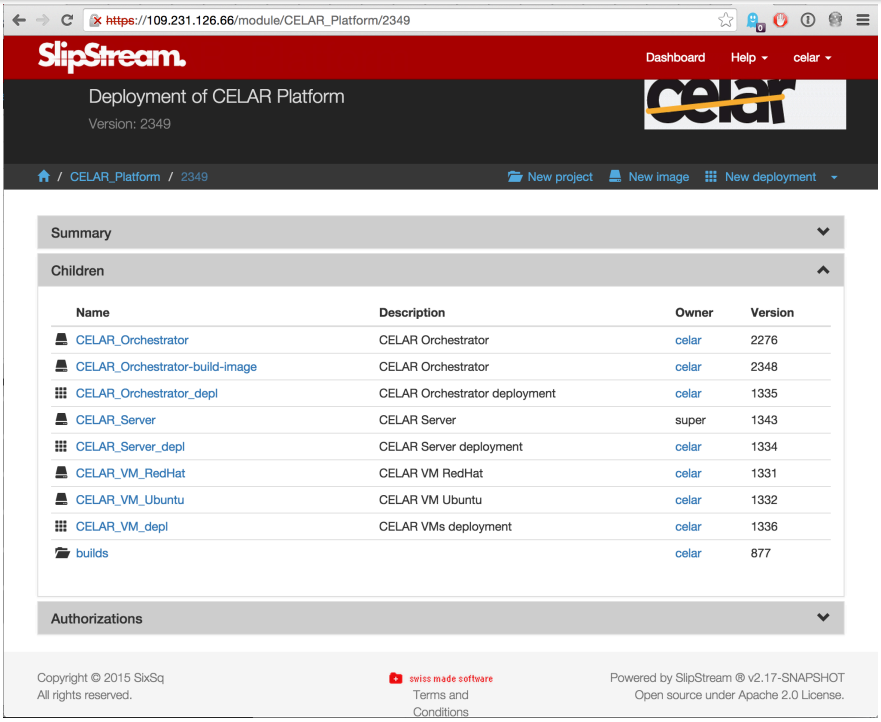

Figure 1 Platform deployment automation with SlipStream.

4 Testing of the CELAR Platform

A good definition and overview of the testing methodologies and strategies can be found in [13]. The approaches and strategies to testing chosen and implemented in a software development project depend on the purpose, architecture, complexity and other attributes of the product, as well as they should be related to the adopted development methodology.

4.1 Chosen testing approaches and strategy

The requirements to the CELAR Platform were gathered at the beginning of the project [14]. Later on, certain quality attributes were updated and minor features added/removed [15], but in overall, the requirements and the feature set remained relatively constant through the lifetime of the project.

The fact that architecturally the CELAR Platform is divided into a number of sufficiently self-contained multi-module components [2] and the project has chosen an iterative approach to the development process, the incremental integration testing in the integration process looked as the most suitable for the project.

In this case, the development WPs can work iteratively on the internal set of the modules gradually building their respective components as a whole. At the same time, when required, the WPs participate in the integration process by running the multi-

component functional tests or providing required mocking level information to the developers of the component's dependant services. This way we are trying to ensure an immediate availability of the developed features and/or propagation of the changes in the components to the integrated Platform. They key point here is a constant coordinated interaction between the groups of developers from WPs, which was achieved by running regular weekly WP6 integration meetings.

The result of each iteration is either a new module/component, or enhancements of the existing modules/components. This module is integrated into the software architecture and the entire system is tested all together.

Table 3 represents the testing approaches that were adopted on the different granularity levels of the project.

Table 3 Chosen testing approaches and their scopes across WPs.

Approach	WP scope	Details
Development testing	Developers in WPs	Unit testing.
Acceptance testing	Developers in WPs	"Smoke" tests.
Functional and non-functional testing	Developers in WP and within WP6 team	
Integration testing	Within WP6 team	Mocking some parts of the platform (if required) allowing testing of a subset of it.
System testing	Within WP6 team and WP7 and WP8	Following the Platform level functional requirements [14, 15].
Usability testing	WP9	Usability of CAMF user interface.

As part of the overall integration strategy we used weekly meetings to plan for the next required set of functional/integration/system tests depending on the readiness of the different per-component features and advances in the integration of the Platform. The planning for the concrete set of tests and their execution was dynamic to suit the development pace (utilizing mocking of the dependencies when required). However, a care was taken to not jeopardize the overall deadlines set for the delivery of the major features of the components by tracking the features delivery progress against the project's high level technical plan.

Those meetings were roughly split into three parts

- provide the latest information on the readiness of the modules/components,
- plan for the next set of functional, integration and/or system tests,
- retrospect the previous integration iterations to provide inter-WPs feedback and improve coordination.

The meetings allowed to define different (usually short term) testing strategies and tactics:

- set testing objective for the next iteration,
- define methods of testing new functionality,
- define required testing environment,
- define resources required for testing,
- estimate total time required for testing.

We also want to highlight here that the project decided not to address the performance and security testing, labeling them as non-functional in the context of the project goals. To address the generic security concerns it was agreed to follow the common sense approach by

- using communication over SSL,
- ensuring no credentials provided to CLI as parameters (so that they don't become visible in the listing of the system processes),
- strict ownership and permissions to the configuration files containing different credentials.

4.2 Test plan

Development testing. Developers of the Platform components that are hosted in the project's organization on GitHub (https://github.com/CELAR) were responsible for writing the unit tests and making them automatically runnable by the project's build framework (Jenkins - http://snf-153388.vm.okeanos.grnet.gr). If tests don't pass, the new build is not produced and the code committer gets notified.

Acceptance testing. A set of per-component deployment and usage "smoke" tests were defined by the component developers in WPs and was ran after the component deployment by the corresponding automated deployment scripts (https://github.com/CELAR/celar-deployment).

Functional and non-functional testing. The tests were planned in advance and included validation of the features that each Platform component provides to the system. Most of the tests were created and ran by the component developers. Creation, planning and executions of some of the tests to assert certain features required inputs from the developers of the pilot applications.
At the same time, the project introduced a non-functional feature: high-availability solution for CELAR Manager component (as part of WP3). This feature was thoroughly tested in WP3.

Integration testing. The integration test plan was dynamic and was subject to changes and adjustments whenever new features were becoming available in the Platform. The integration process started early in the project and span its whole lifetime.

System testing. Because the idea of the tests is to exercise the functional behavior of the Platform as an integrated product by the pilot applications, the tests were planned according the gathered high-level requirements [15].

Usability testing. Testing of the usability of the CAMF was planned and scheduled within WP9 (see [21] - D9.5 section 5).

Each major feature of the Platform was passing through all of the above-mentioned test types and was a subject of assertions on different levels – code correctness, ability to be properly deployed/configured/started, properly working with the other services on the interface level as well as a part of the whole Platform. Due to the iterative nature of the development process, the project was re-running the previous tests for the purpose of regression testing anytime updates to the code were done or new features introduced. The short/middle-term planning of the tests was usually done during the weekly WP6 integration meetings and implemented within the following weeks. Integration and system level testing of some of the major features (e.g., application vertical scalability) were planned before and executed during the project's face-to-face technical meetings two/three times a year.

Our functional and system level KPIs were binary in essence, like

- scaling action decided,
- action completed/not failed,
- metric monitored,
- application described,
- application description stored

and in our tests we were asserting if they were *'true'*. The functional and integration tests specific KPIs are described in each of the per-components sub-section 4.3.x.

4.3 Per-component functional and inter-component integration testing

This section highlights functional testing on the per-component level and integration testing between the communicating components. The tests were performed within WPs during the component development process and inter-component interface level integration processes.

4.3.1 DMM and JCatascopia

The DMM needs to obtain real time metrics from JCatascopia, associate them with the application modules and decide on the action(s) that should be applied to the application. The monitoring metrics are sent from the JCatascopia Server to MELA, which acts as a part of the monitoring system and associates the monitoring metrics to the related application modules/components and forwards them to the DMM, that will take the decisions.

For the validation of the accuracy of this pipeline, a toy application (a Cassandra cluster) was deployed and simple monitoring agents were also installed in the same hosts. In parallel, a JCatascopia server instance was deployed along with MELA and the DMM into a separate host. The test's objective was to verify that the monitoring metrics were successfully interpreted by the DMM; to this end, a YCSB client was utilized to generate load to the cluster. The load followed a sinusoidal pattern: at first the requested throughput was increasing up to a specific point, after which the load was

decreasing until zero. Through this procedure, we ensured that this sinusoidal pattern was identified by JCatascopia, MELA and the DMM. As an outcome, the DMM issued the appropriate "scale out" request at the increasing front of the load and, respectively, "scale in" requests were issued during the decreasing front. In the following section we will describe the testing procedure followed for the integration of the DMM with the CELAR Orchestrator.

4.3.2 DMM and CELAR Orchestrator

When the DMM makes a new elastic decision, it issues a new resizing request to the CELAR Orchestrator module. The CELAR Orchestrator consumes these requests and translates them into SlipStream commands that, in turn, create (or destroy) resources and orchestrate the application to utilize (or stop utilizing) them.

Integration testing was necessary for those two modules in order to verify that: (a) DMM's requests where successfully recognized by the CELAR Orchestrator and (b) the communication protocol between the two modules does not lead to deadlocks or any other pathogenic situations. To this end, a Cassandra cluster was deployed through CELAR and instances of the DMM and the CELAR Orchestrator were deployed to manage the orchestration of the application. According to the resizing actions that were under investigation, the test load and the user policies were tweaked accordingly: e.g. if a scale out action was tested then the resizing policy was similar to: "if request throughput is over a certain value, then scale out" and the load was set to a specific value. This procedure was repeated for each supported resizing action for the two supported cloud providers. This way, the testing verified that (a) the resizing requests are translated appropriately, (b) the possible resizing actions become available to the DMM and (c) the policy changes drastically affect the behavior of the DMM.

4.3.3 CELAR Orchestrator and SlipStream

When the CELAR Orchestrator receives a resizing request it communicates with SlipStream that enforces the action into the deployed application. The CELAR Orchestrator should be able to translate the parameters of the resizing action requests (made by the DMM) into commands/parameters that SlipStream was able to understand. For example, all the resizing requests issued by the DMM that entailed VM flavor changing (e.g. add/remove cores or memory) where accompanied by parameters dictating how many cores and/or memory should be added/removed. On the other hand, SlipStream only understood information regarding the flavor of each VM. However, the term "flavor" would greatly complicate the design of the DMM, thus CELAR Orchestrator became responsible for the translation between the two "dialects", which became one point of validation for the integration testing between the CELAR Orchestrator and SlipStream. Apart from that, timing issues should also be validated: when an issued resizing action is completed, the Orchestrator should be notified immediately.

These two points were tested during the integration testing of the CELAR Orchestrator and SlipStream. The testing was also driven by a toy application: at the first

stages of the implementation, a simple web application was used to verify that deployment and termination of an old deployment was possible. Later on, the horizontal resizing actions (scale out/in) was tested with a Cassandra cluster and the newly added resizing actions related to disk attach/detach, vertical and diagonal scalability were tested using the two applications (Dataplay and SCAN) or parts of the applications for the two different IaaS providers.

4.3.4 SlipStream horizontal and vertical scaling framework

The developed framework in SlipStream for horizontal and vertical scaling was functionally tested as part of the development and integration process in SixSq. For that we used the example multi-component distributed applications that come bundled with SlipStream.

The validation and testing of the horizontal scalability was performed on the web application (apache2) and a set of test clients. SlipStream was requested to deploy the web application layer and the test client with initial number of instances (VMs). After the successful initial deployment of the components of the test application, we were manually requesting the addition and removal of the instances on the two layers (the web application and the test clients). Then, we were asserting that the requested actions were fulfilled on both the cloud layer (VMs were properly added/removed) and on the application layers (all the application level deployment/startup/etc. actions and synchronizations were executed correctly).

The vertical scalability feature introduced IaaS actions that required reboot of the VMs under the actions. In the case of the vertical scaling part of the framework the most important was to ensure that the SlipStream node executors (agents that orchestrate the application deployment) were properly acting as OS daemons on most of the latest version of the Linux distributions (Debian and RedHat based). This assumes they were properly registered within the system startup process, starting and connecting to SlipStream without any issues and resuming from the point they were stopped. But because the framework was developed prior to the cloud connectors implementing the respective IaaS actions, we had to isolate the SlipStream executors and run the VMs reboot action manually. We've tested this in a semi-automated manner, using Vagrant [16] with VirtuaBox [17] as the virtualization provider. We've created a set of Vagrant files targeting Ubuntu (versions 10 to 14) and CentOS (version 5 and 6) for the deployment of the SlipStream executor only. When the provisioning process is done and the executor is running correctly, we were rebooting the VMs and asserting that after the reboot the executor is running and its state is consistent.

The above (and a number of other tests) were ran during the SixSq's sprint demo meetings to validate and assert that the planned features are working as expected and ready for the public release.

4.3.5 Vertical scaling in SlipStream Okeanos and Flexiant connectors

The implementation of the cloud platform specific IaaS actions for Okeanos and Flexiant cloud connectors was done following the guidelines coming with the corresponding SlipStream cloud connector framework.

The functional testing of the VM resizing and extra disk attach/detach features was done by writing the tests that utilize Python's unit testing framework. An example for such tests can be found in [18]. It uses the unit testing framework to create the required fixtures and mocks and launch the series of the cloud level calls to run the workflow that starts VM, resizes, attaches/detaches extra disks and finally releases the allocated cloud resources. The assertions in the tests allowed to ensure the correctness of the code and its proper execution within the predefined workflows. The tests were developed and ran by the ~okeanos and Flexiant connector developers, ensuring that the main functionality of the features work as expected.

For the integration testing, separate instances of SlipStream were deployed on each cloud provider (~okeanos and Flexiant). Initially, a simple single node deployment was used to test and assert the correctness of the added vertical scalability features when running entirely from SlipStream. The next step was to test the added functionality on the scalable parts of the pilot applications. We used DataPlay's pooled PostgreSQL persistence layer on Flexiant and SCAN's worker VMs on ~okeanos. The tests were run manually using SlipStream API and scalability CLI (ss-node-resize, ss-disk-attach/detach commands). The goal was to assert that

- the scaling actions are correctly applied on the IaaS level,
- the application components running on the scaled VMs were able to detect and utilize the change,
- the SlipStream executors on the nodes are running in the consistent state,
- the SlipStream deployment after orchestrating the scalability actions is in final and consistent state.

The above tests were conducted manually. The results of the assertions showed that the features are working correctly and can be integrated further with the automated elasticity decision making stack of the Platform.

4.3.6 CELAR Manager and SlipStream

As presented in D3.3, CELAR Manager is the endpoint of the platform, and one of its responsibilities is to interact with SlipStream and trigger new application deployments. When the deployment is completed, the orchestration control is passed to the CELAR Orchestrator instance which is responsible for the application scaling and management.

The integration between those components, needed to be tested for the following cases:

- CELAR Manager can describe any application into SlipStream,
- CELAR Manager can deploy the application, providing deployment-specific info to SlipStream (e.g. node multiplicity, etc.),
- CELAR Manager can terminate a deployment.

The aforementioned actions were tested for a handful of different applications: at first a Cassandra cluster was used, as mentioned before, while at the later stages of the development DataPlay and SCAN were used as pilot applications.

4.3.7 CAMF and CELAR Manager

CAMF provides a user-friendly UI through which the user can describe their application, in terms of architecture (modules and components), deployment artifacts (e.g. number of VMs per module, flavors, etc.) and elasticity constraints (policies and constraints). All this information is packed into a CSAR file, as presented in D2.3. This CSAR file is sent to the CELAR Manager, where it is translated and deployed.

The testing process was needed to verify that CAMF produces CSAR files understandable to the CELAR Manager; to this end, numerous CSAR files were produced through CAMF describing applications with different architectures and deployment information. Apart from the CELAR Manager, the CSAR file is also forwarded to the CELAR Orchestrator as it contains information needed by the DMM in order to function properly. The same testing process was followed for the CELAR Orchestrator case as well.

4.3.8 Information System and CELAR Manager

The Information System provides details to the users regarding the running and past deployments of their applications. This information is stored into the CELAR DataBase and it is exported to the Information System through the CELAR Manager: the later retains a rest API serving all deployment specific details a user may require. The Information System provides a friendly UI, enabling the user to authenticate with the platform and query it about their applications and deployments.

The testing of the integration of those two components is, essentially, downgraded to testing the REST API (from the CELAR Manager side), the REST client (from the Information System side) and their cooperation. Specifically, every provided call was tested for numerous parameters (if the call supported them) and, in case that the queries returned no results or the parameters were illogical, descriptive error messages were returned and presented to the user.

4.3.9 High availability of CELAR Manager

As presented in D3.3, CELAR supports the deployment in an HA manner. Regarding the CELAR Manager instances, we in parallel deploy multiple CELAR Manager instances and redirect the traffic caused by the clients of the platform towards them through a single load balancer (acting as the endpoint of the platform). To avoid failures at this node, we utilize a cluster of load balancers and elect one node to be the master (through *keepalived)*; the remainder nodes keep checking on the master's status and, in case of failure, elect a new master and execute a script that detaches the public IP address from the failed node and attach it into the newly elected master node. This new node will act as the endpoint of the platform.

The testing procedure of this deployment was the following: we deployed CELAR with multiple CELAR Manager and load balancer instances and caused failures to specific VMs. In our case, the VM failures lead to connection error between the nodes. Since many different errors may occur during the platform's deployment and in many levels (OS Level, VM Level, Application Level), we model those error by their impact on the application's connectivity: if the platform is able to respond to particular queries, we assume that it is running, else the specific node is considered as failed and recovery actions are triggered. During the evaluation, we tried to disable the network interfaces of specific VMs, stop the platform's daemons and reboot the VMs. When those test happened on the CELAR Manager instances we noticed zero downtime: the load balancer realized the absence of a CELAR Manager immediately and new traffic was not redirected to the instance that erred. On the other hand, when those tests were executed on the master load balancer, we noticed a downtime of 1-5 seconds: requests made at this time period failed. This downtime can be attributed to the time needed to detach and re-attach an IP address into a running VM, a procedure that occurs when a new load balancer instance is elected as the new master and obtains the IP address of the failed master. Obviously, this time is heavily affected by the cloud provider but it can be considered marginal as long as VM failures (from the cloud provider side) are seldom. Finally, as also described in D3.3, we replicated the CELAR Database into a PostgreSQL cluster, both for performance and HA reasons. The load balancer that redirects the traffic to the database nodes was placed into the same host as the load balancer mentioned above (that handles traffic for the CELAR Manager instances), and we utilized the same architecture and evaluation mechanisms. In D3.3 Figure 8, we also provide an evaluation of the performance degradation occurring due to master traversal. It is visible that a failure does not result in downtime and the performance drop can be considered as marginal.

4.4 System level testing

During the system level testing, the platform was deployed in both cloud providers (~okeanos and Flexiant) and tested through deploying and scaling the two pilot applications (Dataplay and SCAN). Since the nature of the two applications differs, the respective application descriptions contain different elasticity constraints, resizing actions and policies. During the testing, it was verified that the platform is capable of:

- Receiving applications descriptions with elasticity constraints
- Deploying applications
- Scaling applications according to their elasticity constraints
- Abstracting the cloud's internal properties and providing a unified way of describe/deploying applications to the user

We will not enlist the tests conducted for the two pilot applications for the two different infrastructures.

4.4.1 System testing with DataPlay on Flexiant

DataPlay was deployed and tested over the Flexiant cloud. The following resizing actions were tested:

- Horizontal scalability
- Vertical scalability

Both actions were applied into the "Master" cluster, which issues queries to Cassandra and Redis, as presented in D7.3. Horizontal scalability refers to changing the multiplicity of the cluster (adding and removing nodes) and vertical scalability refers to changing the amount of CPU and memory for each node. A video demonstration of DataPlay's scaling can be found at [19].

4.4.2 System testing with SCAN on Okeanos

SCAN was deployed and tested over ~okeanos. The following resizing actions were tested:

- diagonal scalability,
- disk attach/detach.

Both actions were applied to SCAN's workers, as presented in D8.3. Diagonal scalability is the composition of horizontal and vertical scalability: new VMs are allocated (increasing the cluster's multiplicity) each of which contains an increased amount of resources in comparison to the existing ones. Disk attach/detach refer to adding/removing block devices into the existing VMs. A video demonstration of SCAN's scaling can be found at [20].

4.5 Results and Conclusions

During the development and integration of the CELAR Platform the multimodal and multilevel testing was performed. The testing was a collaborative process between the developers in the research and development WPs (WP2-5), the WP6 team, and the early platform users from WP7 and WP8. All the intended per-component and integrated system level features were tested, asserted and proved to be working by the projects pilot applications (DataPlay in WP7 and SCAN in WP8). As the result, the consortium agreed that the CELAR System meets the project's functional requirements and is acceptable for the final public release.

5 The CELAR Platform readiness, delivery and support

5.1 Technology Readiness Levels

We estimate the TRL level of the **CELAR Platform** as **TRL 6** - technology demonstrated in relevant environment (industrially relevant environment in the case of key enabling technologies). The classification is given according to [1].

The following Table 4 lists the TRL levels of the components of the CELAR Platform, cloud platforms the project was using and the project's pilot applications developed as part of the project.

Table 4 TRL levels of the components of the CELAR Platform, cloud platforms the project was using and the project's pilot applications.

Component/Application/Platform	TRL level	Comments
CELAR Platform (as a whole)	TRL 6	pre-alpha
CAMF (c-Eclipse)	TRL 6	pre-alpha
DMM	TRL 6	pre-alpha
CELAR Manager and Orchestrator	TRL 6	pre-alpha
JCatascopia	TRL 6	pre-alpha
InfoSystem	TRL 6	pre-alpha
SlipStream cloud provisioner	TRL 9	production
FCO cloud platform	TRL 9	production
Okeanos cloud platform	TRL 9	production
SlipStream Okeanos connector	TRL 8	beta
SlipStream Flexiant connector	TRL 8	beta
DataPlay	TRL 7	alpha
SCAN	TRL 7	alpha

5.2 Delivery

We are releasing the Platform as a set of four VM image appliances with the released components preinstalled and preconfigured with sensible defaults. The release notes contain the information about downloading of the appliances, bringing them up as VMs, configuring and using them. The following appliances as CELAR Platform v0.2 will be released

CELAR Server
- Manager
- CELAR DB
- Information System
- SlipStream

CELAR Orchestrator
- DMM
- JCatascopia Server and Web
- CELAR Orchestrator
- SlipStream Orchestrator dependencies

CELAR user VM CentOS
- JCatascopia Agent and OS level probes
- SlipStream node executor dependencies

CELAR user VM Ubuntu
- JCatascopia Agent and OS level probes
- SlipStream node executor dependencies

The Platform appliances (as VMs, and accompanied by the documentation) are available at the following address

http://downloads.celarcloud.eu/appls

An alternative way to deploy the CELAR Platform could be the deployment directly from the binaries repository. As the project chose CentOS 6.x as the target OS the following is the link to the YUM repo with RPMs for CentOS 6.x

http://downloads.celarcloud.eu/yum

In the second case, the set of SlipStream images and deployments utilizing the deployment scripts of the CELAR Platform itself that the project used in its integration processes is going to be handy. That is why they were made publicly available for download and use from GitHub
https://github.com/CELAR/celar-deployment

Using SlipStream CLI *ss-module-upload,* the SlipStream modules available under https://github.com/CELAR/celar-deployment/tree/master/deployment-slipstream can be simply uploaded to any SlipStream instance and used to deploy the CELAR Platform to any cloud the SlipStream instance is pointing to.

After the release of the final version of the CELAR Platform, we are also planning to publish the deployments of the CELAR Platform to SlipStream AppStore on https://nuv.la This will make the Platform available for deployment on virtually any cloud in one click.

The following link points to the list with the CELAR Platform per component documentation
https://docs.celarcloud.eu

5.3 Support

After the end of the project, partners will be providing a certain level of support to the components of the Platform they've developed. Thus, the support for the Platform as a whole should be viewed as based on the options provided by each partner responsible for the development of the respective components.

Support for the components
- SlipStream (SixSq)
- SlipStream connector for FCO (Flexiant)
- Flexiant Cloud Orchestrator (Flexiant)

developed and open sourced or publicly licensed by the commercial partners Flexiant and SixSq is on the best effort basis.

Support to the components
- CELAR Manager, Orchestrator, DB (ATHENA)
- CELAR Information System (UCY)

- CELAR DMM (TUW)

will be provided by the respective academic partners on the voluntary basis.

Since CAMF is now an official Eclipse technology project, support will be provided according to the principles that guide the Eclipse Development Process (see https://www.eclipse.org/projects/handbook/). Members of partner UCY who currently leading the CAMF project will make sure that the necessary policies and guidelines of the above process are adequately maintained.

Support for SlipStream connector for Okeanos by GRNet will be provided until there is an interest and usage of the later as part of the SlipStream installation on Okeanos by GRNet for the purposes of the National Greece scientific community.

6 Conclusions

The work package was able to successfully fulfill the defined goals regarding the extensions of the cloud provisioner and its connectors to support horizontal and vertical scaling actions on both cloud platforms used in the project, and the integration of the coherent set of the components into the CELAR Platform.

At the moment the Platform is undergoing the last steps of the integration and the system level testing process. The final release of the CELAR Platform v0.2 is scheduled for the end of October 2015.

7 References

[1] TRL levels. https://en.wikipedia.org/wiki/Technology_readiness_level

[2] D1.2 CELAR Architecture
https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Deliverables:Deliverables_1.X:D1.2

[3] D2.3 c-Eclipse Framework (CAMF)
https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Deliverables:Deliverables_2.X:D2.3

[4] D3.3 CELAR Manager and Orchestrator
https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Deliverables:Deliverables_3.X:D3.3

[5] D4.3 Multi-Level Metric Evaluation Module, JCatascopia and Info System
https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Deliverables:Deliverables_4.X:D4.3

[6] D5.3 Decision Making Module (DMM)
https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Deliverables:Deliverables_5.X:D5.3

[7] CAMF. https://projects.eclipse.org/proposals/cloud-application-management-framework

[8] HelixNebula project. http://www.helix-nebula.eu

[9] Scalable deployments in SlipStream. Documentation.
http://ssdocs.sixsq.com/documentation/advanced_tutorial/scalable_deployments.html

[10] SlipStream release notes.
http://ssdocs.sixsq.com/documentation/release_notes/candidate_releases.html

[11] SlipStream API. http://ssapi.sixsq.com

[12] D6.2 Integration Prototype "CELAR System Prototype" V2.
https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Deliverables:Deliverables_6.X:D6.2

[13] Ali Mili, Fairouz Tchier, “Software Testing: Concepts and Operations”, Wiley-Blackwell, 2015. ISBN-13: 978-1118662878.

[14] D1.1 User Requirements and System Architecture V1, Month 6
https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Deliverables:Deliverables_1.X:D1.1

[15] D1.2 Updated User Requirements and System Architecture, Month 20
https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Deliverables:Deliverables_1.X:D1.2

[16] https://www.vagrantup.com/

[17] https://www.virtualbox.org/

[18] Example of vertical scalability tests https://github.com/CELAR/SlipStreamConnector-Okeanos/blob/master/python/tar/test/TestOkeanosClientCloudLive.py

[19] https://www.youtube.com/watch?v=xDHt61N4wbM

[20] https://www.youtube.com/watch?v=YWdSy-l3QOA

[21] D9.5 Dissemination and Exploitation Activities Report V3
https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Deliverables:Deliverables_9.X:D9.5