



PaaSage

Model Based Cloud Platform Upperware

Deliverable D9.3.2

Final Training Material and Workshop Product Launch

Version: 1

D9.3.2

Name, title and organisation of the scientific representative of the project's coordinator¹:

Mr Philippe Rohou **Tel: +33 (0)4 97 15 53 06**

Fax: +33 (0)4 92 38 78 22

E-mail: phillipe.rohou@ercim.eu

Project website² address: <http://paasage.eu/>

Project	
Grant Agreement number	317715
Project acronym:	PaaSage
Project title:	Model Based Cloud Platform Upperware
Funding Scheme:	Integrated Project
Date of latest version of Annex I against which the assessment will be made:	20 April 2016
Document	
Period covered:	M36-M42
Deliverable number:	D9.3.2
Deliverable title	Final Training Material and Workshop Product Launch
Contractual Date of Delivery:	30-09-2016
Actual Date of Delivery:	30-09-2016
Editor (s):	Manos Papoutsakis (FORTH)
Author (s):	Manos Papoutsakis, Kyriakos Kritikos, Kostas Magoutis, Pierre Guisset, Daniel Baur, Etienne Charlier
Reviewer (s):	Ping Wang, Tom Kirkham
Participant(s):	All
Work package no.:	9
Work package title:	Training and Dissemination
Work package leader:	Pierre Guisset
Distribution:	PU
Version/Revision:	1.0
Draft/Final:	Final
Total number of pages (including cover):	94

¹ Usually the contact person of the coordinator as specified in Art. 8.1. of the grant agreement

² The home page of the website should contain the generic European flag and the FP7 logo which are available in electronic format at the Europa website (logo of the European flag: http://europa.eu/abc/symbols/emblem/index_en.htm ; logo of the 7th FP: http://ec.europa.eu/research/fp7/index_en.cfm?pg=logos). The area of activity of the project should also be mentioned.

DISCLAIMER

This document contains description of the PaaSage project work and findings.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any responsibility for actions that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of the PaaSage consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 27 Member States of the Union. It is based on the European Communities and the member states cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors. (<http://europa.eu>)



PaaSage is a project funded in part by the European Union.

Executive Summary

Success of any platform, either commercial or research-based, is critically dependent on the availability of sufficient documentation and training materials. These materials guide prospective users and maintainers through to an understanding of the platform, and demonstrate how the platform's main functionality can be exploited to achieve user goals and requirements. The purpose of this deliverable is to complement and extend previous deliverables, providing insight into the planning and interim implementations of training materials (D9.3.1 Initial Training Materials, M24) as well as deliverables describing design and implementation of workshops (D9.4.2 Industrial Workshop Planning, M36; D9.4.1 Workshops Prototype Design, M18).

This deliverable describes training material that is made available by PaaSage and that are presented online at <http://www.paasage.eu/training-materials>. The training materials include: instructions on how to deploy and configure the PaaSage platform or its modules and instructions on the management of the PaaSage Executionware. Detailed processes on how users can specify their requirements in CAMEL is also covered to along with ways in which the PaaSage's Social Network (SN) can be utilized in order to achieve best utilise the platform. Such utilisation includes the discovery and sharing of critical knowledge which can be exploited in the management of multi-cloud applications. The deliverable additionally focuses on the finalized design of how to conduct a training workshop about the PaaSage project, which draws from experience from several such events and capitalizing on the training material made available by the project. This design can be used as guidance for training workshops ranging from one-hour sessions to half- or full-day events. Finally, this deliverable complements D9.4.2 (M36), reporting on industrial workshops that were organized by the project in the period beyond M36 through M42.

Intended Audience

This is a public document intended for different types of users, including business users, application designers, IT architects, system administrators and software engineers, depending on their intended usage of the PaaSage platform. In particular:

- *Business users* can exploit the presented material in order to be guided in expressing their business requirements as well as their organization's information via the CAMEL meta-model
- *System administrators* can be assisted in deploying and configuring the PaaSage platform as well as learn how to consider technical requirements and regulations.
- *Software engineers* can utilize the presented information in order to: (a) express application component and quality requirements in CAMEL, (b) extend the PaaSage platform components, and (c) produce new components with added-value functionality.
- *Application developers / IT architects* can be assisted in expressing application requirements in CAMEL as well as exploiting previous application design knowledge.
- *Any user type*: any user can utilize the presented material in order to learn how to use the PaaSage's Social Network (SN) for sharing application design and technical knowledge, browsing existing applications, deploying them, and exploiting previous application execution history and SN recommendations to establish better application deployments.

For each type of user, different knowledge and skills are required for a better comprehension of this document or its parts and the material it presents. Each presented material may also interest and cater for different types of users. For even better comprehension of the document, the prospective reader is also referred to the description of the overall PaaSage architecture presented in Deliverable [D1.6.1](#) [D1.6.1]. D1.6.1 provides background on the overall PaaSage architecture and the way different modules fit in it, as well as the internal architecture of particular PaaSage components. The reader can also refer to the separate deliverables for each PaaSage module, i.e., [D3.1.1](#) [D3.1.1], [D4.1.1](#) [D4.1.1] and [D5.1.1](#) [D5.1.1] in order to have a complete view about the exposed functionality of each module and its internal components. Finally, the reader can refer to the deliverable [D2.1.2](#) [D2.1.2] for a complete documentation of the CAMEL meta-model and how it can be used to express different types of models, including those pertaining to end-user requirements.

Contents

Executive Summary	4
Intended Audience	5
Contents	6
1 Introduction.....	9
2 PaaSage Platform Configuration & Deployment.....	13
2.1 Supported platforms & Resource Requirements	13
2.2 Deployment of the PaaSage platform	13
2.2.1 STEP #1: VM CREATION	13
2.2.2 STEP #2: Bootstrap code download.....	14
2.2.3 STEP #3 Bootstrap code configuration	14
2.2.4 STEP #4 Bootstrap code execution	14
2.2.5 STEP #5: Connect Social Network to the new PaaSage Platform	15
2.2.6 STEP #6 Clean up	15
3 Executionware Deployment & Usage.....	16
3.1 Installation	16
3.2 Configuration	16
3.3 Example: Deploying an application.....	16
3.3.1 Application Model.....	17
3.3.1.1 Writing the application scripts	17
i. An utility script for common operations.....	18
ii. A script installing the haproxy server.....	18
iii. A script installing apache2 and mediawiki.....	20
iv. A script installing MariaDB.	22
3.3.1.2 Selecting the desired cloud resources	23
3.3.1.3 Defining the LifecycleComponents, the ApplicationComponent and the Application.....	23
i. LifecycleComponents:.....	23
ii. Application:	23
iii. ApplicationComponents	23
3.3.1.4 Defining the Communication	24
3.3.1.5 Linking the scripts to Cloudiator	24
i. HaProxy Bridge Script.....	24
ii. Mediawiki Bridge Script	25
iii. MariaDB Bridge Script.....	26
3.3.1.6 API Interaction.....	26
i. Creating the Application.....	26
ii. Creating the LifecycleComponents	27
iii. Creating the VirtualMachineTemplate	29
iv. Creating the ApplicationComponents	29

v.	Creating the RequiredPorts, and ProvidedPorts	30
vi.	Creating the Communication.....	32
3.3.2	Cloud Model.....	32
3.3.2.1	Openstack Example	33
3.3.2.2	API Interaction	33
i.	Create API	33
ii.	Create Cloud	34
iii.	Create CloudCredential	35
3.3.2.3	Discovery	36
3.3.3	Starting the application.....	36
3.3.3.1	Starting virtual machines	36
3.3.3.2	Starting instances	36
3.3.3.3	Waiting until the deployment is finished	36
3.3.3.4	API Interaction	37
i.	Starting virtual machines	37
ii.	Creating the application instance.....	39
iii.	Creating the application component instances	40
3.3.4	Java Example.....	42
3.3.4.1	Introduction	42
3.3.4.2	Installation	42
3.3.4.3	Configuration	42
3.3.4.4	Running the example	43
4	Camel Model Creation	45
4.1	Overview.....	45
4.2	CAMEL creation	46
4.2.1	Deployment Aspect – DeploymentModel.....	47
4.2.1.1	Components	47
4.2.1.2	Communications	49
4.2.1.3	Hostings	49
4.2.2	Requirement Aspect – RequirementModel	50
4.2.2.1	Hard requirements	50
4.2.2.2	Soft requirements	52
4.2.3	Location Aspect – Location Model.....	54
4.2.4	Measurement/Metric Aspect – MetricModel	55
4.2.4.1	Metrics	55
4.2.4.2	Metric Formulas	55
4.2.4.3	Properties	55
4.2.4.4	Metric Conditions	56
4.2.4.5	Property Conditions	56
4.2.4.6	Condition Contexts	56
4.2.4.7	Metric Context	56

4.2.5	Scalability Aspect – ScalabilityModel	57
4.2.5.1	Scalability Rules	57
4.2.5.2	Actions	57
4.2.5.3	Events	57
4.2.6	Security Aspect – SecurityModel	62
4.2.7	Type Aspect - TypeModel	64
4.2.8	Unit Aspect – UnitModel	66
4.3	Summary	66
5	Social Network User Guide	68
5.1	Site Sections	68
5.2	User Login / Register	69
5.3	User Profile	70
5.4	Social Network Community	72
5.5	Models	73
5.6	Deployment of an application	75
5.7	Summary	77
6	Training Workshops	78
6.1	Design and implementation	78
6.1.1	General overview of the PaaSage project	78
6.1.2	Success stories	78
6.1.3	Introduction to CAMEL	79
6.1.4	Application deployment using the ruby client of PaaSage platform	79
6.1.5	Introduction of PaaSage social network and integration with PaaSage platform	80
6.1.6	Questionnaire	82
6.1.7	Additional sections	82
7	Industrial workshops	83
7.1	Product launch strategy	83
7.2	Structure of the industrial workshops	83
7.2.1	ASC(S Industrial Workshop (Stuttgart, Germany)	84
7.2.2	EVERY Industrial Workshop (Oslo, Norway)	86
7.2.3	LSY Industrial Workshop (Budapest, Hungary)	88
7.3	New PaaSage leaflet	90
7.4	PaaSage roll-up	92
8	Conclusion	93
9	Bibliography	94

1 Introduction

The PaaSage platform may be exploited by different organisations with various goals to achieve. Some organisations might just desire to exploit the main functionality of the platform. While others might need to adopt and possibly extend it in order to develop a cloud-based platform in the form of a business product, which can make a differentiation in the market and thus increase its market share. However, before exploiting the platform, sufficient documentation and training material should be in place which will indicate the various uses of the platform and detail of all necessary steps and required knowledge that is needed to perform the appropriate steps towards its utilisation. This is exactly the main purpose of this deliverable: to provide training material which can be used for the proper exploitation of the PaaSage platform.

The set of material presented in this deliverable is separated into the following different sections:

- Section 2 presents material which indicates how to configure, build and deploy the PaaSage platform. This material will certainly interest organisations who would like to create and configure their own platform, through which their applications can be deployed across different clouds. This material is mainly intended for *system admins* involved in such organisations as it contains low-level technical details which might not be understandable e.g. by business users. However, this does not mean that other types of users cannot exploit it to fulfil the respective task as the description of the platform configuration and deployment process is quite straightforward.
- Section 3 focuses on a particular module of the PaaSage platform, namely the Executionware, with the intention to illustrate how this module can be deployed and utilized to enable an organisation to deploy its applications in the cloud. The deployment process described unveils various steps that need to be performed manually in contrast to the respective automated executionware deployment process described in Section 2. The material is directly suited for *system admins*, similarly to the case of Section 2, as it involves technical details that are more understandable and manageable by this type of users. The content is also suited for *application developers* who wish to gather a deeper understanding of the interplay of software components in the PaaSage Executionware. Finally, the example subsection contained in this section can be used by any user of the platform (e.g., an application owner or a user desiring to deploy an application belonging to a specific organization) who wish to perform application deployments in the cloud, as it shows how to register existing cloud accounts in the PaaSage platform.
- Sections 4 and 5 play complementary roles towards enabling organisations to provide appropriate information as input to the platform in order to fulfil particular organisation-required tasks, such as the deployment of applications. In particular, Section 4 analyses how the high-level business requirements of an organisation can be transformed into model-based information described via the Camel meta-model (see Deliverables D2.1.3). These requirements can then be used as input to the platform for appropriately deploying applications in the clouds as well as adapting them according to particular scalability rules. On the other hand, Section 5 analyses the social network perspective of the PaaSage platform by indicating how users can specify user profiles, manage

and search for application models, and exploit knowledge which has been produced from the execution history of the same or similar applications. This use of history encourages users to continually refine and pose in a better and more precise ways to express requirements as models. To this end, by combining the information from these two sections, a user will be able to perform various tasks which will enable him/her to appropriately manage his/her applications that are deployed in the cloud without really getting into low-level technical details at the platform and infrastructure level. The material in Section 4 is intended toward a variety of users as the combination of their skills and knowledge can lead to the transformation of high-level business requirements to requirements at the application and infrastructure level. The material in Section 5 does not impose particular requirements on the type of users, especially as we consider that the social network (SN) can cater for many types of users, apart from some basic knowledge in social networking which is not mandatory as the SN has been designed with a user-intuitive and simple to use UI.

- Section 6 is a step by step tutorial of how to conduct a training workshop about the PaaSage platform. Experience acquired from similar events lead to the composition of this tutorial, which can be used as guidance for training workshops which could last from half to a whole day. Such events may play important role in lowering the level of difficulty of using the PaaSage platform as a whole (model editors, platform and social network). Moreover, the conduct of a training session may be an important source of feedback for the training process itself and the PaaSage platform. Although the description of how a training workshop should be conducted is quite straightforward, the presenter of some parts of this tutorial must have a deep understanding of specific parts of the PaaSage platform. Such a part is the creation of a CAMEL model for an application, which requires the presenter to be at least familiar with the CAMEL modelling language.
- Section 7 describes the structure of industrial workshops that took place during the PaaSage project as part of the PaaSage product launch strategy. Those workshops were organized in several countries by the PaaSage industrial partners and aimed potential business partners who could embrace innovation provided by PaaSage.

The types of user that can benefit from the material presented in this document are the following:

- *system admins*: They can inspect the configuration and building guidelines provided in order to build and deploy the PaaSage platform or its parts/modules, like the Executionware. Their specialized knowledge enables them not only to comprehend such guidelines but also to implement them by also respecting their organisation's technical requirements, peculiarities and regulations and bypass any technical obstacles.
- *business users*: Such users, which might not have a technical background in IT, can benefit from the material provided in order to obtain and share knowledge as well as publish their application models. Supporting expression of business requirements for their applications which, in collaboration with other types of users from the same organisation, can be materialized into concrete Camel models specifying deployment, scalability, quality and

security requirements. These models can then be issued into the PaaSage platform for the proper management of respective applications in the cloud.

- *application developers / IT Architects*: Can exploit the guidelines to obtain and share application design knowledge as well as publish publication models for their organisations. They are also guided in the provision of requirements for their applications in terms of concrete Camel models.
- *software engineers*: They can exploit the presented material in order to: (a) extend particular PaaSage platform components, (b) produce new components that can be fitted into the platform providing added-value functionality, (c) develop the missing components for particular applications when the available library of components stored in the platform cannot be used for realizing completely the desired functionality, and (d) express application component and quality requirements in terms of Camel models.
- *simple users*: This user type can include the user types mentioned above, thus actually catering for any type of user (thus could be renamed as just *user*). They can exploit the material specifically related to the SN in order to find out applications that interest them and deploy them into the cloud by just specifying some quality requirements, if needed, and to let the system derive additional requirements and information needed for the proper deployment and management of the respective application (e.g., from the previous execution history of the application or of application similar to the desired one). Apart from just having a basic knowledge of how a SN functions, no other requirement is imposed on this type of user.

Table 1: Mapping of presented material to targeted user types

Material	System Admins	Business users	Application developers	Software Engineers	Simple Users
PaaSage platform configuration & deployment	√			~	
Execution-ware deployment & usage	√		√	~	
Camel model creation	√	√	√	~	
Social Network User Guide	√	√	√	√	√

For each type of user indicated above, we demonstrate in the following Table 1 the respective material that may be of interest to him/her. The rows of the table correspond to the presented material while the columns to the respective user types.

The symbols used in the table's cell content have the following meaning: "√" means that the material is especially targeted at the particular user type while "~" means that the presented material could interest the respective user type.

It should also be highlighted that apart from the documentation and user guides provided in this deliverable, there also exists video material from which some of the screenshots shown in the various guides/material in this document have been extracted. This material provides a significant and complementary role with respect to support training of potential users in the exploitation of the PaaSage platform and is available at <http://www.paasage.eu/training-materials>.

2 PaaSage Platform Configuration & Deployment

A lot of effort has been put into the automation of configuration and deployment of the PaaSage platform in order to make it as easy as possible for a newcomer to get it running.

2.1 *Supported platforms & Resource Requirements*

PaaSage is only supported on the Linux platform. Although any modern Linux distribution can be exploited, the PaaSage platform deployment has only been tested on Ubuntu 14.04 64bits.

PaaSage needs to be deployed on a virtual machine with at least:

- 2 CPU cores
- 8 GB ram
- 20 GB Hard disk

During application deployment by PaaSage, the destinations virtual machines need to communicate on specific ports specifically to contact the PaaSage platform (EG: to report deployment progress or for metrics collection).

The PaaSage platform thus needs to be reachable on the following ports:

- 22/TCP: SSH
- 80/TCP: execution ware UI
- 4001/TCP: colosseum etcd daemon
- 8080/TCP: time series database
- 9000/TCP: colosseum UI
- 9999/TCP: REST API
- 33034/TCP: RMI registry

2.2 *Deployment of the PaaSage platform*

As already stated, a lot of effort has been put to hide the complexity of the PaaSage platform deployment.

A few simple steps to execute on a fresh virtual machine gets a user started. In the following procedures, we assume the PaaSage platform is deployed on a Remote Cloud and not on a local workstation.

2.2.1 STEP #1: VM CREATION

- Actual actions depend on the Cloud provider (EC2, AZURE, OPENSTACK instance)
- Please make sure the requirements are fulfilled
 - VM size

- PUBLIC_IP MAPPING
- PORTS accessible
- SSH access

2.2.2 STEP #2: Bootstrap code download

- Log into the PaaSage VM
- Run the following commands

```
$ sudo apt-get update
$ sudo apt-get install git
$ git clone https://tuleap.ow2.org/plugins/git/paasage/paasage_one_click_install.git
```

- Bootstrap code is now downloaded on the to-be PaaSage VM.

2.2.3 STEP #3 Bootstrap code configuration

The bootstrap code is tailored by a few shell script variables. These variables need to be adapted for each deployment of the PaaSage platform.

The configuration file must be named \$HOME/override_vars.sh. An example is provided together with the bootstrap script (paasage_one_click_install/bootstrap/bootstrap.sh).

Please follow these steps to configure the bootstrap script.

- Log into PaaSage VM
- Run the following commands

```
$ cp paasage_one_click_install/override_vars.sh_SAMPLE
  ./override_vars.sh
$ vi override_vars.sh
```

- Adapt the file with your values
 - **NODE_GROUP**: identify the vm's created by this PaaSage platform (use only lowercase letters and limit to 10 characters)
 - **ELASTIC_IP**: public IP address assigned to this PaaSage Platform (by default, detected using checkip.amazonaws.com)
 - **SEED**: some random string (lower case letters) used as seed for password generation
 - Destination clouds credentials

2.2.4 STEP #4 Bootstrap code execution

Once the override_vars.sh is adapted, you can run the PaaSage platform deployment.

- Log into the PaaSage platform, then execute

```
$ paasage_one_click_install/bootstrap/bootstrap.sh
```

- Watch the deployment taking place. It can take from 10 to 20 min

2.2.5 STEP #5: Connect Social Network to the new PaaSage Platform

Connecting the PaaSage platform with the social network (SN) allows the latter to be used for CAMEL model deployment.

- Connect to the Social Network web site
- Navigate to My Area → Credentials
- Fill in the endpoint, email, password and tenant
- Click on the Save changes button

2.2.6 STEP #6 Clean up

If you want to reuse this PaaSage platform to deploy another application, you need to clean the databases up

- Log into the PaaSage Platform
- Execute the following command
- `$ /etc/paasage/reset-platform.sh` (the platform will reboot automatically)

3 Executionware Deployment & Usage

As described in Deliverable D5.1.2, the Executionware has the primary purpose to deploy applications into the target cloud infrastructure and to monitor the current runtime context of the deployed applications.

Besides being integrated into the PaaSage workflow, the Executionware is also available as a stand-alone project named Cludiator (<https://cludiator.github.io>). To increase the visibility and sustainability of the documentation, most content is featured on the webpage dedicated to Cludiator. This document thus only provides pointers to the webpage, where Cludiator's usage is depicted in more detail.

3.1 Installation

To ease the installation, Cludiator features an easy to use installation script that can be retrieved under <https://cludiator.github.io/docs/installation.html>. The script will install all dependencies of Cludiator and start the required services. Afterwards, it's UI can be access by calling http://{ip-of-server}/executionware_ui and its REST API is available under <http://{ip-of-server}:9000>.

In addition, the installation of Cludiator is also included into the scripts installing PaaSage.

3.2 Configuration

The installation scripts configure Cludiator with a sufficient default configuration, so that most use cases can be run without changing any configuration options. Especially in the PaaSage environment, the configuration of Cludiator does not need be changed. For more advanced use cases, all configuration options are available under <https://cludiator.github.io/docs/configuration.html>.

3.3 Example: Deploying an application

In this section we provide a tutorial that contains a detailed step-by-step guide, helping the user to deploy their first application using the Cludiator toolset, which must be installed as it is described in the Installation section above.

This tutorial will cover the following steps:

- i. A short introduction on the sample application (Mediawiki).
- ii. We will describe an Openstack Cloud so that it can be used with Cludiator.
- iii. We will describe Mediawiki using bash scripts so it can be deployed with Cludiator.
- iv. We will deploy Mediawiki using Cludiator.

Each step will be two-fold. First it will describe and explain the steps needed in detail by providing knowledge and background information. At the end, each section will describe the actions required to execute the steps with the Cludiator toolset, using a) the [REST-API](#), b) our [java client](#) and c) our [user interface](#).

3.3.1 Application Model

For this first step of modeling the application the following information is needed.

1. For each component, i.e. a *LifecycleComponent*, of the application you need:
 - a script used for *installing* and *starting* the component on the virtual machine.
 - the *Image* used for booting the virtual machine.
 - the *Hardware* used for booting the virtual machine.
 - the *Location* used for booting the virtual machine.
2. The *Communication* dependencies.

A more detailed description for the application model is given in the corresponding [Documentation Section](#).

3.3.1.1 Writing the application scripts

The first part of the scripts can be written independently from Cloudiator. In general we need three scripts:

- one installing the database (MariaDB)
- one installing the application server (apache2) and the wiki
- one installing the load balancer (HaProxy)

For each script we define two start actions:

- one blocking the start as required by [Lance's](#) Docker deployment
- one non-blocking start action as required by [Lance's](#) plain deployment.

In addition we define the following arguments for the scripts:

- the application server installation scripts takes the database IP address as argument.
- the load balancer scripts takes multiple application server IP addresses as argument.

This leads to the scripts presented below, also available at [Github](#). These scripts rely on apt-get to install packages, and were only tested on Ubuntu 14.04 LTS.

Each of these scripts provides the following functions when used with the depicted arguments:

Argument	Description
install	Installs the application on the server.
Start	Starts the application (non-blocking).
startBlocking	Starts the application (blocking).
configure	Configures the application, e.g. by downloading or writing configuration files.
Stop	Stops the application.

i. An utility script for common operations

This script simply provides the logic to run apt-get update and dist-upgrade while trying its best to avoid any interaction with the user.

```
#!/bin/bash

apt_update() {
    unset UCF_FORCE_CONFFOLD
    export UCF_FORCE_CONFFNEW=YES
    ucf --purge /boot/grub/menu.lst
    export DEBIAN_FRONTEND=noninteractive
    sudo -E apt-get update
    sudo -E apt-get -o Dpkg::Options::="--force-confold" --force-yes -fuy dist-upgrade
}
```

ii. A script installing the haproxy server

This script installs and configures the haproxy server.

```
#!/bin/bash

MY_DIR="$(dirname "$0")"
source "$MY_DIR/util.sh"
TMP_DIR="/tmp"
HA_PROXY_CONFIG_URL="https://raw.githubusercontent.com/dbaur/mediawiki-tutorial/master/config/haproxy.cfg"
RSYSLOG_CONFIG_URL="https://raw.githubusercontent.com/dbaur/mediawiki-tutorial/master/config/haProxyRsyslog.cfg"

IPS=${@:2}

install() {
    apt_update

    #install haproxy
    sudo apt-get -y install haproxy wget

    #enable haproxy
    sudo sed -i "s/ENABLED=0/ENABLED=1/g" /etc/default/haproxy

    #configure rsyslog
    wget ${RSYSLOG_CONFIG_URL} -O ${TMP_DIR}/haProxyRsyslog.tmp
    sudo cp ${TMP_DIR}/haProxyRsyslog.tmp /etc/rsyslog.d/haproxy.conf

    sudo /etc/init.d/rsyslog restart
    IPS="127.0.0.1"
    configure

    sudo /etc/init.d/haproxy stop
}

configure() {
```

```

#validate ips
if ! [[ -n "$IPS" ]]; then
    echo "Expected list of ips as parameter but got none."
    exit 1
fi

# remove existing tmp file
rm -rf ${TMP_DIR}/haproxy.tmp
# download config template
wget ${HA_PROXY_CONFIG_URL} -O ${TMP_DIR}/haproxy.tmp

# write servers into template
i=1
SERVERS=""
for var in ${IPS}
do
    SERVERS+="server wiki$i $var:80 check\\n"
    ((i++))
done
sudo sed -i -e "s/${servers}/${SERVERS}/" ${TMP_DIR}/haproxy.tmp

# mv temp file to real location
sudo mv /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.bak
sudo mv ${TMP_DIR}/haproxy.tmp /etc/haproxy/haproxy.cfg

# reload haproxy
sudo /etc/init.d/haproxy reload

}

start() {
    # start haproxy
    sudo /etc/init.d/haproxy start
}

startBlocking() {
    # start haproxy and sleep for infinity
    sudo /etc/init.d/haproxy start && sleep infinity
}

stop() {
    # stop haproxy
    sudo /etc/init.d/haproxy stop
}

### main logic ###
case "$1" in
    install)
        install
        ;;
    start)
        start
        ;;
    startBlocking)
        startBlocking

```

```

;;
configure)
    configure
;;
stop)
    stop
;;
*)
    echo $"Usage: $0 {install|start|startBlocking|configure|stop}"
    exit 1
esac

```

iii. ***A script installing apache2 and mediawiki.***

```

#!/bin/bash

MY_DIR="$(dirname "$0")"
source "$MY_DIR/util.sh"

TMP_DIR="/tmp"

# Download URL for mediawiki
MW_DOWNLOAD_URL="https://releases.wikimedia.org/mediawiki/1.26/mediawiki-1.26.2.tar.gz"

# Database
DB="wiki"
DB_USER="wiki"
DB_PASS="password"
DB_HOST=$2

# Wiki
NAME="dbaur"
PASS="admin1345"

install() {
    apt_update
    # Install dependencies (apache2, php5, php5-mysql)
    sudo apt-get --yes install apache2 php5 php5-mysql wget
    # remove existing mediawiki archive
    rm -f ${TMP_DIR}/mediawiki.tar.gz
    # download mediawiki tarball
    wget ${MW_DOWNLOAD_URL} -O ${TMP_DIR}/mediawiki.tar.gz
    # remove existing mediawiki folder
    sudo rm -rf /opt/mediawiki
    sudo mkdir -p /opt/mediawiki
    # extract mediawiki tarball
    sudo tar -xvzf ${TMP_DIR}/mediawiki.tar.gz -C /opt/mediawiki --strip-components=1
    # remove existing mediawiki symbolic link
    sudo rm -rf /var/www/html/wiki
    # create symbolic link
    sudo ln -s /opt/mediawiki /var/www/html/wiki
    # enable mod status
    sudo a2enmod status
}

```

```

# allow server status from everywhere
sudo sed -i "s/Require local/#Require local/g" /etc/apache2/mods-enabled/status.conf
# stop apache
sudo service apache2 stop
}

configure() {
    if ! [[ -n "$DB_HOST" ]]; then
        echo "you need to supply a db host"
        exit 1
    fi
    sudo service apache2 start
    # run mediawiki installation skript
    sudo php /opt/mediawiki/maintenance/install.php --dbuser ${DB_USER} --dbpass
${DB_PASS} --dbname ${DB} --dbserver ${DB_HOST} --pass ${PASS} $NAME
"admin"
    sudo service apache2 stop
}

start() {
    sudo service apache2 start
}

startBlocking() {
    sudo service apache2 start && sleep infinity
}

stop() {
    sudo service apache stop
}

### main logic ###
case "$1" in
    install)
        install
        ;;
    start)
        start
        ;;
    startBlocking)
        startBlocking
        ;;
    configure)
        configure
        ;;
    stop)
        stop
        ;;
    *)
        echo $"Usage: $0 {install|start|startBlocking|configure|stop}"
        exit 1
esac

```

iv. A script installing MariaDB.

```
#!/bin/bash

MY_DIR="$(dirname "$0")"
source "$MY_DIR/util.sh"

ROOT_PW="topsecret"
DB="wiki"
DB_USER="wiki"
DB_PASS="password"

install() {
    apt_update
    #set default root password for automated installation
    sudo debconf-set-selections <<< 'mariadb-server mysql-server/root_password
password ${ROOT_PW}
    sudo debconf-set-selections <<< 'mariadb-server mysql-server/root_password_again
password ${ROOT_PW}
    sudo apt-get --yes install mariadb-server
    sudo service mysql stop
}

start() {
    sudo service mysql start
}

startBlocking() {
    sudo service mysql start && sleep infinity
}

configure() {
    sudo service mysql start

    #create database
    mysql -u root -p${ROOT_PW} -e "CREATE DATABASE $DB;"

    #create user and grant privileges
    mysql -u root -p${ROOT_PW} -e "GRANT ALL PRIVILEGES ON $DB.* TO
'$DB_USER'@ '%' IDENTIFIED BY '$DB_PASS';";
    mysql -u root -p${ROOT_PW} -e "FLUSH PRIVILEGES;"

    #configure bind address
    sudo sed -i "s/.*/bind-address = 0.0.0.0/" /etc/mysql/my.cnf

    sudo service mysql stop
}

stop() {
    sudo service mysql stop
}

### main logic ###
case "$1" in
```

```

install)
    install
    ;;
start)
    start
    ;;
startBlocking)
    startBlocking
    ;;
configure)
    configure
    ;;
stop)
    stop
    ;;
*)
    echo $"Usage: $0 {install|start|startBlocking|configure|stop}"
    exit 1
esac

```

3.3.1.2 Selecting the desired cloud resources

As the next step we have to select the desired cloud offerings that we want to use for the virtual machines which will host the different application components.

For simplicity, we will use the same combination of *Image*, *Hardware* and *Location* for all *ApplicationComponents*.

All cloud resources can be retrieved by using the respective list actions of [Colosseum's API](#).

Once we have selected the desired cloud resources, creating a *VirtualMachineTemplate* is straightforward. As all components are going to use the same template, we will create only one using the foreign keys of the respective resources.

3.3.1.3 Defining the LifecycleComponents, the ApplicationComponent and the Application

When creating the *Application* with Cloudiator the user has to define the following entities:

- i. LifecycleComponents:**
 - A component for the HaProxy Loadbalancer
 - A component for the Apache Webserver including MediaWiki
 - A component for the MariaDB database server.
- ii. Application:**
 - One application: MediaWiki
- iii. ApplicationComponents**
 - three application components, each linking the created components to the application.

3.3.1.4 Defining the Communication

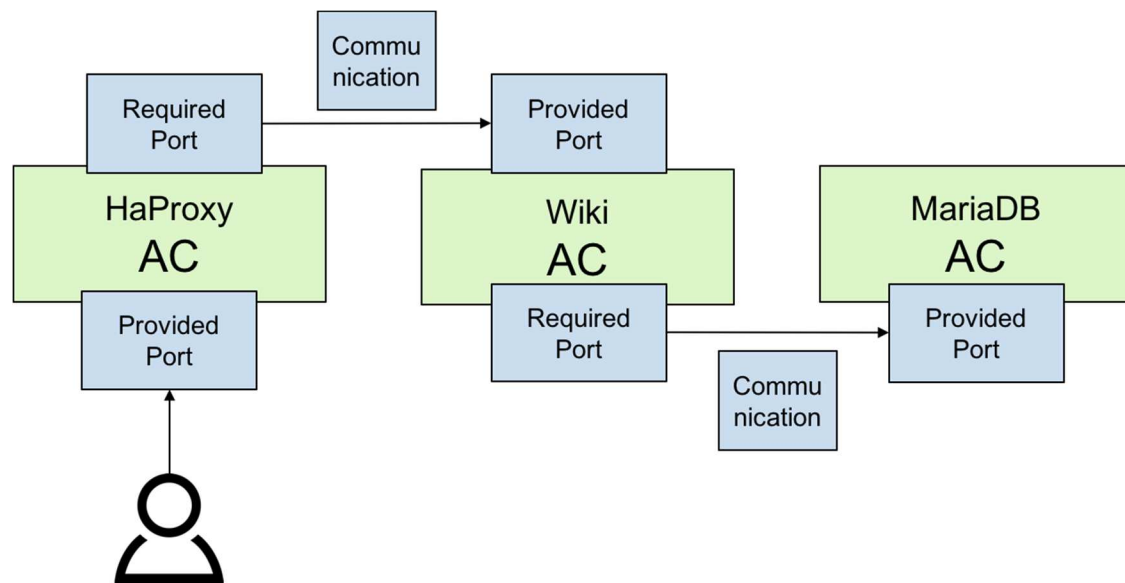


Figure 1: Communication within the Mediawiki Application

The picture above depicts the *Communication* within the Mediawiki Application.

ProvidedPorts: - MariaDB provides the database on port 3306. - Wiki (and Apache) provide the web server on port 80. - The HaProxy provides the load-balanced website on port 80. **RequiredPorts:** - HaProxy requires the webserver. - The wiki requires the database. **Communication:** - LOADBALANCERREQWIKI: link between the HaProxy and the webserver. - WIKIREQMARIADB: link between the web-server and the database.

It is important to remember the name of the communication entities, as the environment variables used in the script rely on them.

3.3.1.5 Linking the scripts to Cloudiator

As explained in the communication section of the [application model documentation](#) Cloudiator uses environment variables to provide IP addresses of downstream components. To account for this fact, we have to write a simple bridge script parsing this information and calling the corresponding scripts with the correct arguments. These scripts can also be found on [Github](#).

The corresponding bridge scripts just forward the main argument (see table above) to the original script, but in addition parses the environment variables if necessary and sends them as arguments to the above scripts.

i. **HaProxy Bridge Script**

```
#!/bin/bash

MY_DIR="$(dirname "$0")"

### main logic ###
case "$1" in
    configure)
        if [ -z ${PUBLIC_LOADBALANCERREQWIKI+123} ] ; then
```



```

        MESSAGE="Environment variable PUBLIC_LOADBALANCERREQWIKI
required, but not set."
        echo $MESSAGE
        exit 3
    elif [ -z ${PUBLIC_LOADBALANCERREQWIKI} ] ; then
        echo "Environment variable PUBLIC_LOADBALANCERREQWIKI
required, but not set to reasonable value."
        exit 3
    else
        arr=$(echo $PUBLIC_LOADBALANCERREQWIKI | tr " " "\n")
        for x in $arr
        ## take the last one (because there are only one)
        do
            echo "PUBLIC_LOADBALANCERREQWIKI > [$x]"
            WIKI_HOSTS+=$(echo "$x" | sed -e "s/.*$/")
            WIKI_HOSTS+=" "
        done
    fi
    ./${MY_DIR}/../shell/haproxy.sh configure $WIKI_HOSTS
;;
*)
    ./${MY_DIR}/../shell/haproxy.sh $@
esac

```

ii. **Mediawiki Bridge Script**

```

#!/bin/bash

MY_DIR="$(dirname "$0")"

### main logic ###
case "$1" in
    configure)
        DB_HOST="0.0.0.0"
        if [ -z ${PUBLIC_WIKIREQMARIADB+123} ] ; then
            MESSAGE="Environment variable PUBLIC_WIKIREQMARIADB
required, but not set."
            echo $MESSAGE
            exit 3
        elif [ -z ${PUBLIC_WIKIREQMARIADB} ] ; then
            echo "Environment variable PUBLIC_WIKIREQMARIADB required, but
not set to reasonable value."
            exit 3
        else
            arr=$(echo $PUBLIC_WIKIREQMARIADB | tr " " "\n")
            for x in $arr
            ## take the last one (because there are only one)
            do
                echo "PUBLIC_WIKIREQMARIADB > [$x]"
                DB_HOST=$(echo "$x" | sed -e "s/.*$/")
            done
        fi
        ./${MY_DIR}/../shell/mediawiki.sh configure $DB_HOST
    ;;
)

```

```
*)
./${MY_DIR}/../shell/mediawiki.sh $@
esac
```

iii. **MariaDB Bridge Script**

```
MY_DIR="$(dirname "$0")"

#!/bin/bash

MY_DIR="$(dirname "$0")"
./${MY_DIR}/../shell/mariaDB.sh $@
```

3.3.1.6 API Interaction

Finally, we can start creating the entities using the API of Cloudiator.

i. **Creating the Application**

REST

```
{
  "name": "MediawikiApplication"
}
```

colosseum-client

```
Application application = client.controller(Application.class).updateOrCreate(new
ApplicationBuilder().name("MediawikiApplication").build());
```

UI

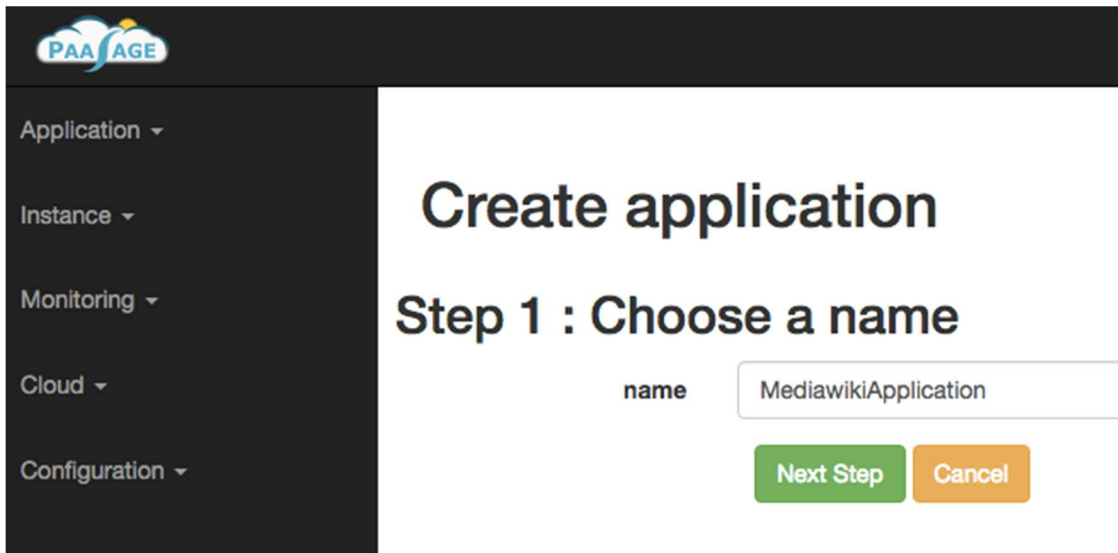


Figure 2: Creating application - Mediawiki

ii. **Creating the LifecycleComponents**

REST

```
{
  "name": "LoadBalancer",
  "preInstall": "sudo apt-get -y update && sudo apt-get -y install git && git clone
https://github.com/dbaur/mediawiki-tutorial.git",
  "install": "./mediawiki-tutorial/scripts/lance/haproxy.sh install",
  "start": "./mediawiki-tutorial/scripts/lance/haproxy.sh startBlocking"
}
{
  "name": "MediaWiki",
  "preInstall": "sudo apt-get -y update && sudo apt-get -y install git && git clone
https://github.com/dbaur/mediawiki-tutorial.git",
  "install": "./mediawiki-tutorial/scripts/lance/mediawiki.sh install",
  "postInstall": "./mediawiki-tutorial/scripts/lance/mediawiki.sh configure",
  "start": "./mediawiki-tutorial/scripts/lance/mediawiki.sh startBlocking"
}
{
  "name": "MariaDB",
  "preInstall": "sudo apt-get -y update && sudo apt-get -y install git && git clone
https://github.com/dbaur/mediawiki-tutorial.git",
  "install": "./mediawiki-tutorial/scripts/lance/mariaDB.sh install",
  "postInstall": "./mediawiki-tutorial/scripts/lance/mariaDB.sh configure",
  "start": "./mediawiki-tutorial/scripts/lance/mariaDB.sh startBlocking"
}
```

colosseum-client

```
String downloadCommand = "sudo apt-get -y update && sudo apt-get -y install git
&& git clone https://github.com/dbaur/mediawiki-tutorial.git";
```

```
LifecycleComponent loadBalancer =
client.controller(LifecycleComponent.class).updateOrCreate( new
LifecycleComponentBuilder().name("LoadBalancer").preInstall(downloadCommand)
.install("./mediawiki-tutorial/scripts/lance/haproxy.sh install")
.start("./mediawiki-tutorial/scripts/lance/haproxy.sh startBlocking")
.build());
```

```
LifecycleComponent wiki =
client.controller(LifecycleComponent.class).updateOrCreate( new
LifecycleComponentBuilder().name("MediaWiki").preInstall(downloadCommand)
.install("./mediawiki-tutorial/scripts/lance/mediawiki.sh install")
.postInstall("./mediawiki-tutorial/scripts/lance/mediawiki.sh configure")
.start("./mediawiki-tutorial/scripts/lance/mediawiki.sh startBlocking").build());
```

```
LifecycleComponent mariaDB =
client.controller(LifecycleComponent.class).updateOrCreate(new
LifecycleComponentBuilder().name("MariaDB").preInstall(downloadCommand)
.install("./mediawiki-tutorial/scripts/lance/mariaDB.sh install")
.postInstall("./mediawiki-tutorial/scripts/lance/mariaDB.sh configure")
.start("./mediawiki-tutorial/scripts/lance/mariaDB.sh startBlocking").build());
```

UI

The screenshot shows the 'Create component' form in the PAA AGE interface. On the left is a dark sidebar with a menu containing: Application (with a dropdown arrow), application, component (highlighted), virtualMachineTemplate, applicationComponent, communication, requiredPort, providedPort, Instance (with a dropdown arrow), and applicationInstance. The main area has a title 'Create component' and a form with the following fields: 'name' (LoadBalancer), 'init' (empty), 'preInstall' (sudo apt-get -y update && sudo apt-get -y install git && git clone https://github.com/dbaur/mediawiki-tutorial.git), 'install' (/mediawiki-tutorial/scripts/lance/haproxy.sh install), 'postInstall' (empty), 'preStart' (empty), and 'start' (/mediawiki-tutorial/scripts/lance/haproxy.sh startBlocking). The 'start' field is highlighted with a blue border.

Figure 3: Creating internalComponent - LoadBalancer

The screenshot shows the 'Create component' form in the PAA AGE interface for 'Mediawiki'. The sidebar is identical to Figure 3. The form fields are: 'name' (Mediawiki), 'init' (empty), 'preInstall' (sudo apt-get -y update && sudo apt-get -y install git && git clone https://github.com/dbaur/mediawiki-tutorial.git), 'install' (/mediawiki-tutorial/scripts/lance/mediawiki.sh install), 'postInstall' (/mediawiki-tutorial/scripts/lance/mediawiki.sh configure), 'preStart' (empty), 'start' (/mediawiki-tutorial/scripts/lance/mediawiki.sh startBlocking), and 'startDetection' (empty).

Figure 4: Creating internalComponent - Mediawiki

The screenshot shows the 'Create component' form in the PAA AGE interface for 'MariaDB'. The sidebar is identical to Figure 3. The form fields are: 'name' (MariaDB), 'init' (empty), 'preInstall' (sudo apt-get -y update && sudo apt-get -y install git && git clone https://github.com/dbaur/mediawiki-tutorial.git), 'install' (/mediawiki-tutorial/scripts/lance/mariaDB.sh install), 'postInstall' (/mediawiki-tutorial/scripts/lance/mariaDB.sh configure), 'preStart' (empty), 'start' (/mediawiki-tutorial/scripts/lance/mariaDB.sh startBlocking), and 'startDetection' (empty).

Figure 5: Creating internalComponent - MariaDB

iii. **Creating the VirtualMachineTemplate**

REST

```
{
  "cloud":1,
  "image":1,
  "location":1,
  "hardware":1
}
```

colosseum-client

```
VirtualMachineTemplate virtualMachineTemplate =
client.controller(VirtualMachineTemplate.class).create( new
VirtualMachineTemplateBuilder().cloud(cloud.getId()).location(location.getId())
.image(image).hardware(hardware.getId()).build());
```

UI

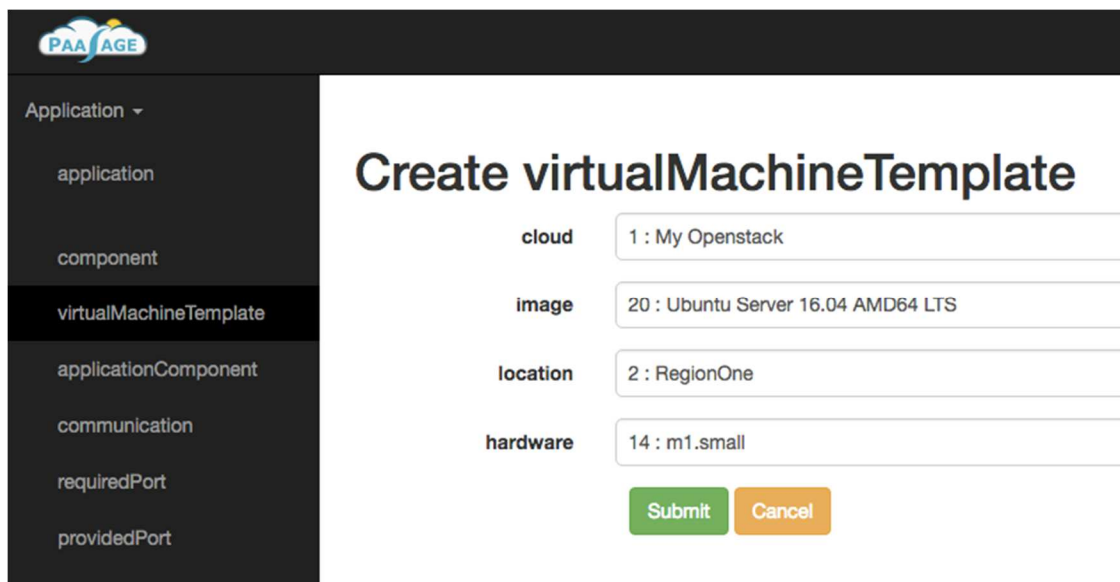


Figure 6: Creating virtualMachineTemplate

iv. **Creating the ApplicationComponents**

REST

```
{
  "application":1,
  "component":1,
  "virtualMachineTemplate":1
}
```

```
{
  "application":1,
  "component":2,
  "virtualMachineTemplate":1
}
```

```
{
```

```

"application":1,
"component":3,
"virtualMachineTemplate":1
}

```

colosseum-client

```

ApplicationComponent loadBalancerApplicationComponent =
    client.controller(ApplicationComponent.class).create(
        new ApplicationComponentBuilder().application(application.getId())
            .component(loadBalancer.getId())
            .virtualMachineTemplate(virtualMachineTemplate.getId()).build());

```

```

ApplicationComponent wikiApplicationComponent =
    client.controller(ApplicationComponent.class).create(
        new ApplicationComponentBuilder().application(application.getId())
            .component(wiki.getId())
            .virtualMachineTemplate(virtualMachineTemplate.getId()).build());

```

```

ApplicationComponent mariaDBApplicationComponent =
    client.controller(ApplicationComponent.class).create(
        new ApplicationComponentBuilder().application(application.getId())
            .component(mariaDB.getId())
            .virtualMachineTemplate(virtualMachineTemplate.getId()).build());

```

UI

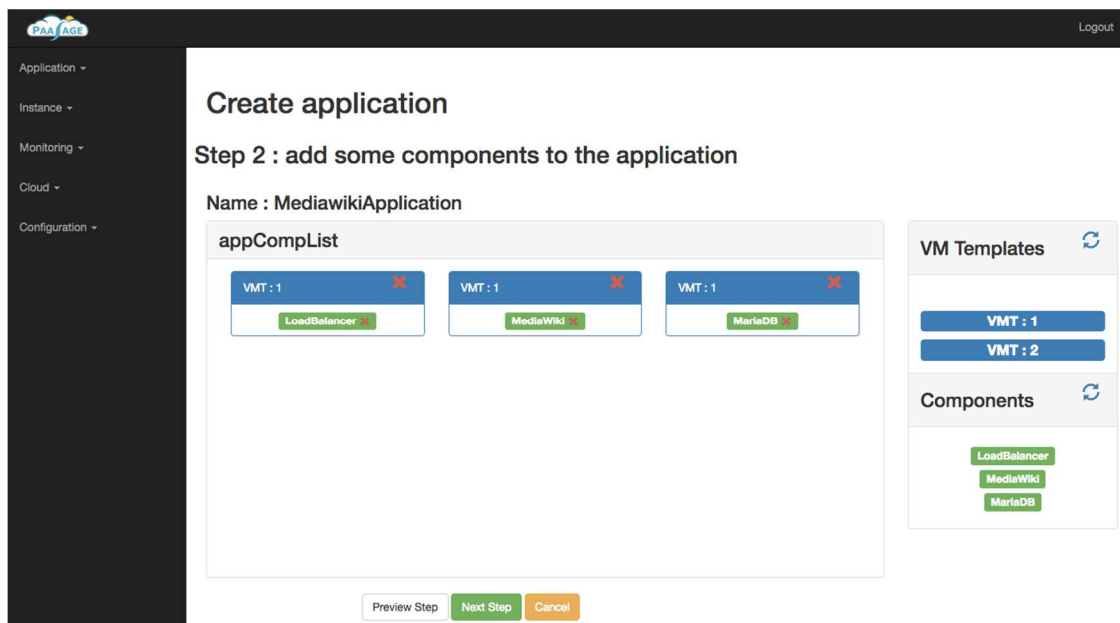


Figure 7: Adding components to the application

v. **Creating the RequiredPorts, and ProvidedPorts**

UI

See communication.

REST

```
{
  "name":"MARIADBPROV",
  "applicationComponent":1,
  "port":3306
}
```

```
{
  "name":"WIKIPROV",
  "applicationComponent":2,
  "port":80
}
{
  "name":"WIKIREQMARIADB",
  "isMandatory":"true",
  "applicationComponent":2
}
```

```
{
  "name":"LBPROV",
  "applicationComponent":3,
  "port":80
}
{
  "name":"LOADBALANCERREQWIKI",
  "updateAction":"./mediawiki-tutorial/scripts/lance/haproxy.sh configure",
  "isMandatory":"true",
  "applicationComponent":3
}
```

colosseum-client

```
//database
final PortProvided mariadbprov = client.controller(PortProvided.class).create(
    new PortProvidedBuilder().name("MARIADBPROV")

.applicationComponent(mariaDBApplicationComponent.getId()).port(3306).build());
// wiki
final PortProvided wikiprov = client.controller(PortProvided.class).create(
    new PortProvidedBuilder().name("WIKIPROV")
    .applicationComponent(wikiApplicationComponent.getId()).port(80).build());
final PortRequired wikireqmariadb = client.controller(PortRequired.class).create(
    new PortRequiredBuilder().name("WIKIREQMARIADB")

.applicationComponent(wikiApplicationComponent.getId()).isMandatory(true).build())
;
// lb
final PortProvided lbprov = client.controller(PortProvided.class).create(
    new PortProvidedBuilder().name("LBPROV")

.applicationComponent(loadBalancerApplicationComponent.getId()).port(80).build());
final PortRequired loadbalancerreqwiki =
client.controller(PortRequired.class).create(
    new PortRequiredBuilder().name("LOADBALANCERREQWIKI")
    .applicationComponent(loadBalancerApplicationComponent.getId())
```

```

.isMandatory(false)
.updateAction("./mediawiki-tutorial/scripts/lance/haproxy.sh configure")
.build();

```

vi. **Creating the Communication**

UI

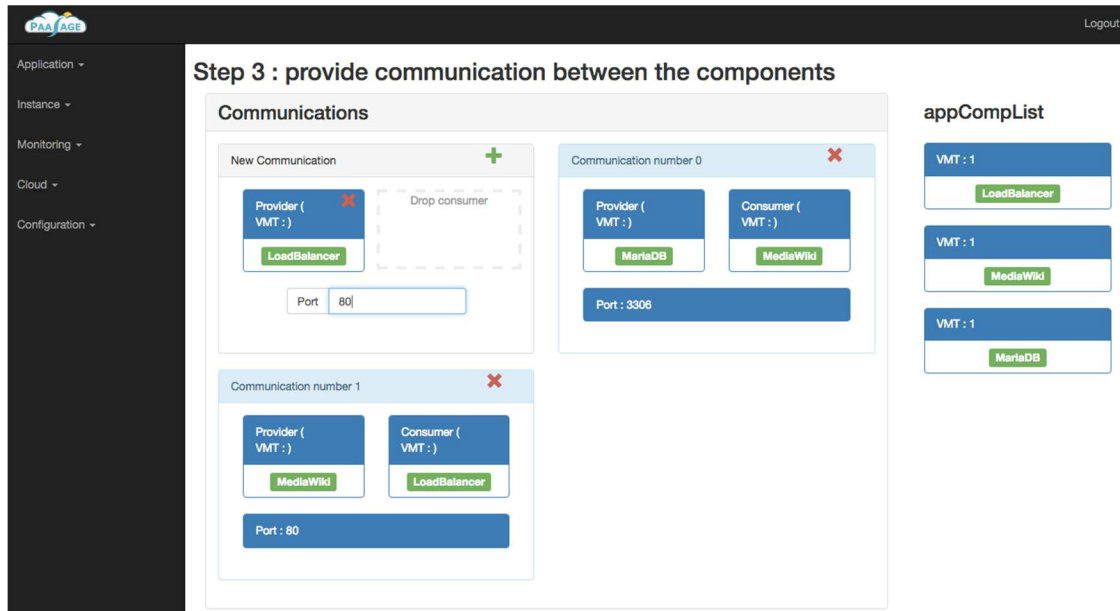


Figure 8: Creating communications

colosseum-client

```

// wiki communicates with database
final Communication wikiWithDB =
client.controller(Communication.class).create(
    new CommunicationBuilder().providedPort(mariadbprov.getId())
    .requiredPort(wikireqmariadb.getId()).build());
//lb communicates with wiki
final Communication lbWithWiki = client.controller(Communication.class).create(
    new CommunicationBuilder().providedPort(wikiprov.getId())
    .requiredPort(loadbalancerreqwiki.getId()).build());

```

3.3.2 Cloud Model

Before deploying an application with the Cloudiator framework the first step is to define the cloud target.

In order to define a cloud target three entities have to be created:

1. An Api, depicting the programming interface the cloud uses, e.g. Nova in case of Openstack.
2. A Cloud, depicting the endpoint where the API of the cloud is reachable.
3. A CloudCredential, depicting the user credentials for the given cloud endpoint.

3.3.2.1 Openstack Example

In this example we will create the required entities using an Openstack Cloud.

For this purpose, we will need the following information of your Openstack Cloud.

1. The *endpoint* of the Openstack Cloud.
2. Your *tenant* name.
3. Your *username*.
4. Your API *password*.

For other cloud providers, you can have a look at the [examples section](#) of Sword. The code examples list the information required for those providers.

You can easily retrieve this information from the Openstack dashboard:

5. Login
6. Select the correct *tenant* from the tenant dropdown menu near the Openstack logo in the top-left edge. (If it does not exist, you only have one tenant and you can skip this step)
7. Go to the “Compute” tab on the left navigation bar.
8. Click Access & Security.
9. Click the button “Download Openstack RC File”.
10. Open the file in the editor: *endpoint* maps to OS_AUTH_URL, *tenant* maps to OS_TENANT_NAME, *username* maps to OS_USERNAME and *password* is most likely the password you also used for dashboard authentication or OS_PASSWORD.

3.3.2.2 API Interaction

Finally, you can start creating the entities in Cloudiator. Throughout the example, we will list three possibilities: Using the [colosseum-client](#), direct REST (e.g. by using a client like [Insomnia](#)) or the [user interface](#).

Following the links above the guides for the components which present the APIs can be found including information about API authentication.

i. Create API

colosseum-client

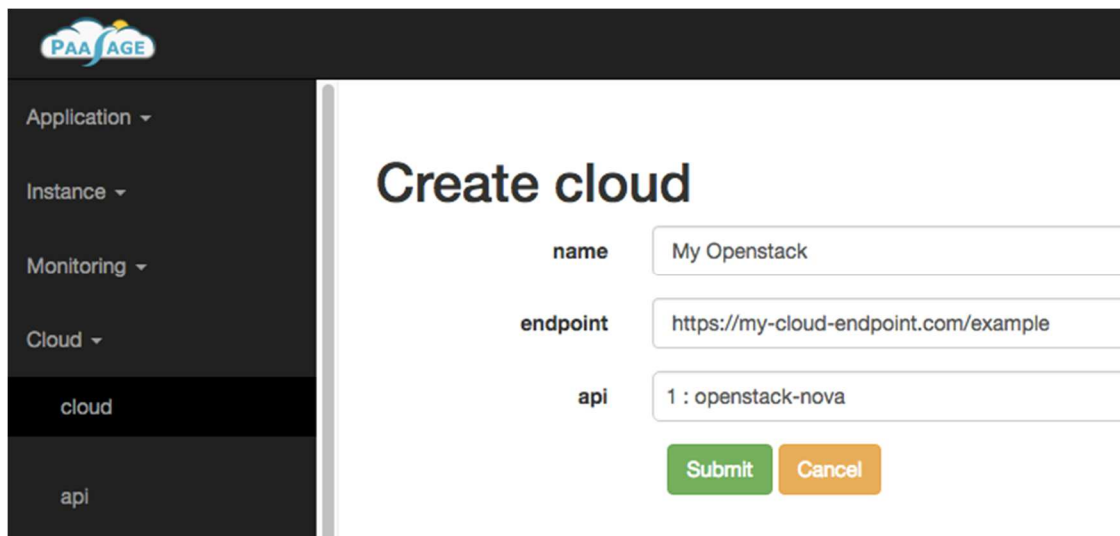
```
String apiName = "openstack-nova";
String internalProviderName = "openstack-nova";

client.controller(Api.class).updateOrCreate(
    new ApiBuilder().name(apiName)
        .internalProviderName(internalProviderName).build());
```

REST

```
{
  "internalProviderName": "openstack-nova",
  "name": "openstack-nova"
}
```

UI



The screenshot shows the PAA AGE web interface. On the left is a dark sidebar with a menu containing 'Application', 'Instance', 'Monitoring', 'Cloud', 'cloud', and 'api'. The 'cloud' item is highlighted. The main content area has a title 'Create cloud' and a form with three input fields: 'name' with the value 'My Openstack', 'endpoint' with the value 'https://my-cloud-endpoint.com/example', and 'api' with the value '1 : openstack-nova'. At the bottom of the form are two buttons: a green 'Submit' button and an orange 'Cancel' button.

Figure 9: Creating cloud

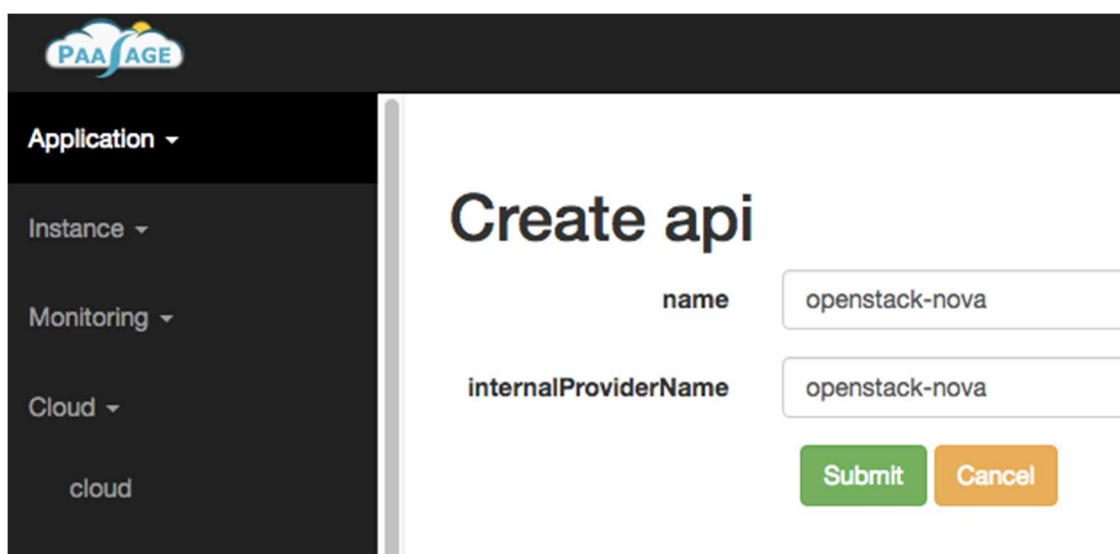
ii. **Create Cloud**

colosseum-client

```
Long apiId = 1;  
String endpoint = "https://my-cloud-endpoint.com/example";  
String cloudName = "My Openstack";
```

```
client.controller(Cloud.class).updateOrCreate(  
    new CloudBuilder().api(apiId).endpoint(endpoint)  
        .name(cloudName).build());
```

UI



The screenshot shows the PAA AGE web interface. On the left is a dark sidebar with a menu containing 'Application', 'Instance', 'Monitoring', 'Cloud', 'cloud', and 'api'. The 'cloud' item is highlighted. The main content area has a title 'Create api' and a form with two input fields: 'name' with the value 'openstack-nova' and 'internalProviderName' with the value 'openstack-nova'. At the bottom of the form are two buttons: a green 'Submit' button and an orange 'Cancel' button.

Figure 10: Creating API

REST

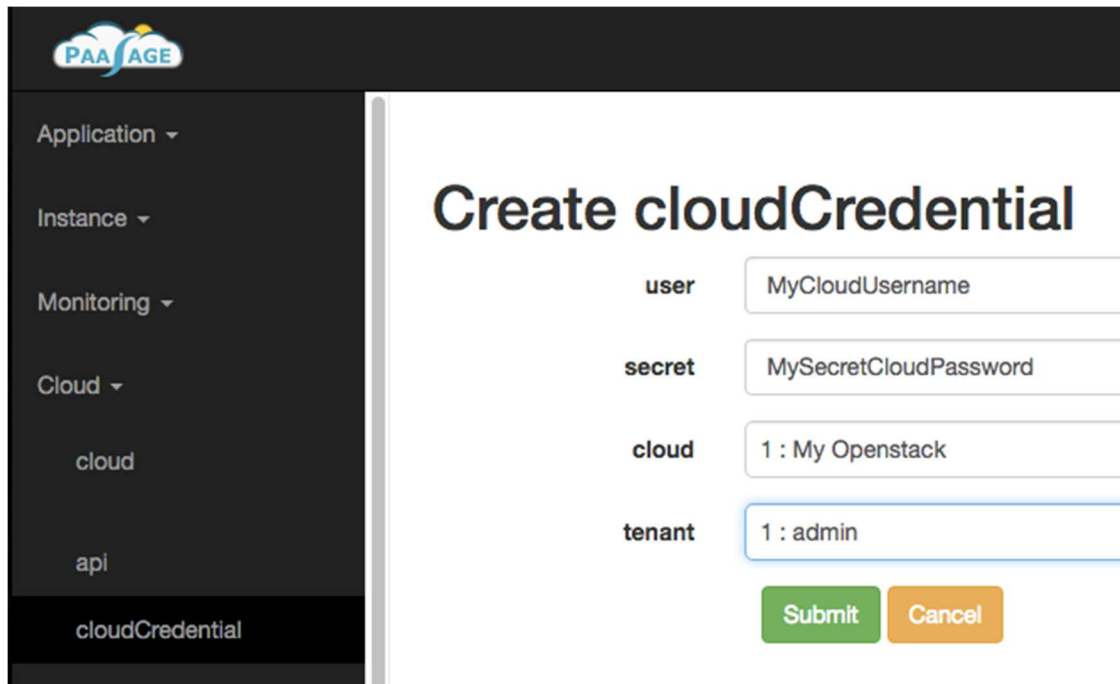
```
{  
  "name": "My Openstack",  
  "endpoint": "https://my-cloud-endpoint.com/example",  
  "api": 1  
}
```

iii. **Create CloudCredential**

colosseum-client

```
Long cloudId = 1;  
String username = "MyCloudUsername";  
String secret = "MySecretCloudPassword";  
  
client.controller(CloudCredential.class).updateOrCreate(  
    new CloudCredentialBuilder()  
        .cloud(cloudId)  
        .secret(secret)  
        .user(username)  
        .tenant(1)  
        .build());
```

UI



The screenshot shows the PAA AGE web interface. On the left is a dark sidebar with a menu containing 'Application', 'Instance', 'Monitoring', 'Cloud', 'cloud', 'api', and 'cloudCredential'. The 'cloudCredential' item is highlighted. The main content area has a title 'Create cloudCredential' and a form with four fields: 'user' (MyCloudUsername), 'secret' (MySecretCloudPassword), 'cloud' (1 : My Openstack), and 'tenant' (1 : admin). At the bottom of the form are 'Submit' and 'Cancel' buttons.

Figure 11: Creating cloudCredential

REST

```
{  
  "user": "MyCloudUsername",  
}
```

```

"secret": "MySecretCloudPassword",
"cloud": 1,
"tenant": 1
}

```

3.3.2.3 Discovery

After having created the above entities, the discovery of colosseum will start, importing the various offers (*Images*, *Hardware* and *Locations*) of the cloud provider. You can see the discovered entities, when accessing the corresponding API endpoints.

3.3.3 Starting the application

The next step is to start the instantiation of the application by starting the required virtual machines and installing the components by creating instances.

3.3.3.1 Starting virtual machines

Starting virtual machines is now easy. Simply select the correct offers with respect to Hardware, Image and Location and pass them to Colosseum.

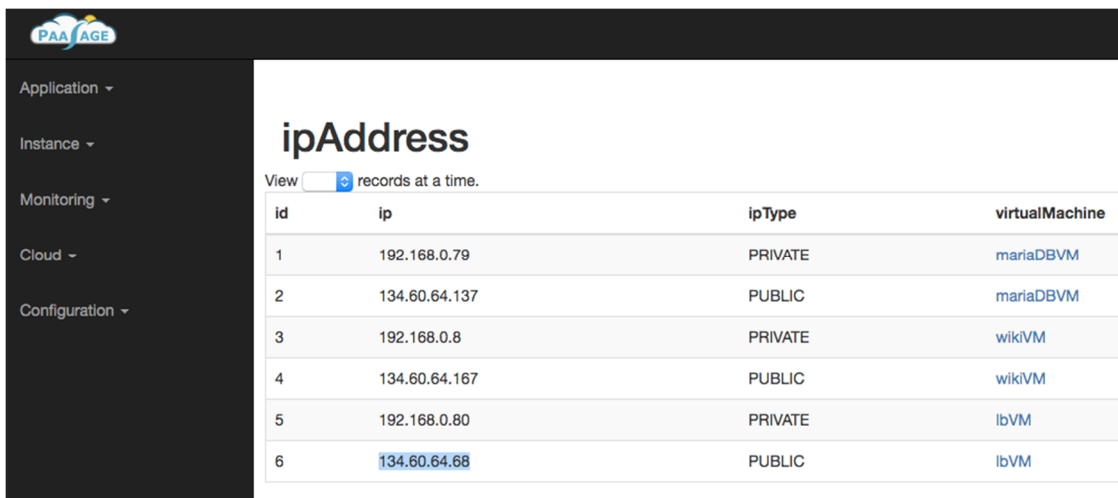
3.3.3.2 Starting instances

Before a VMinstance is started, we have to create a new application instance. An application instance is a logical group for instances, that belong together.

Afterwards, we can start the instances, by binding the already created application components to their virtual machines.

3.3.3.3 Waiting until the deployment is finished

Finally, we have to wait until the instances report a remote state of OK. Using the user interface we can see the IP Address where the load balancer is located, and access our wiki installation.



id	ip	ipType	virtualMachine
1	192.168.0.79	PRIVATE	mariaDBVM
2	134.60.64.137	PUBLIC	mariaDBVM
3	192.168.0.8	PRIVATE	wikiVM
4	134.60.64.167	PUBLIC	wikiVM
5	192.168.0.80	PRIVATE	lbVM
6	134.60.64.68	PUBLIC	lbVM

Figure 12: IP address of load balancer

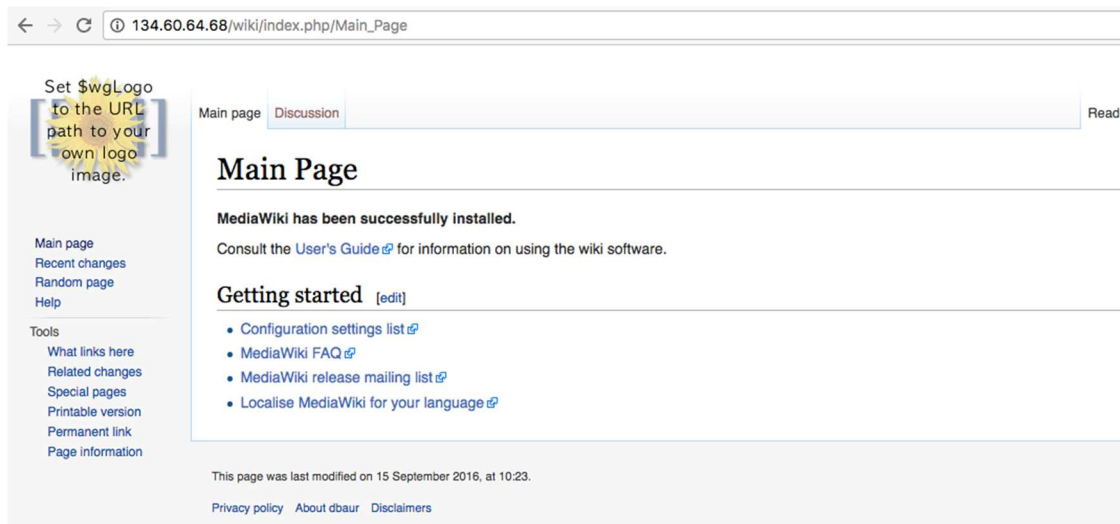


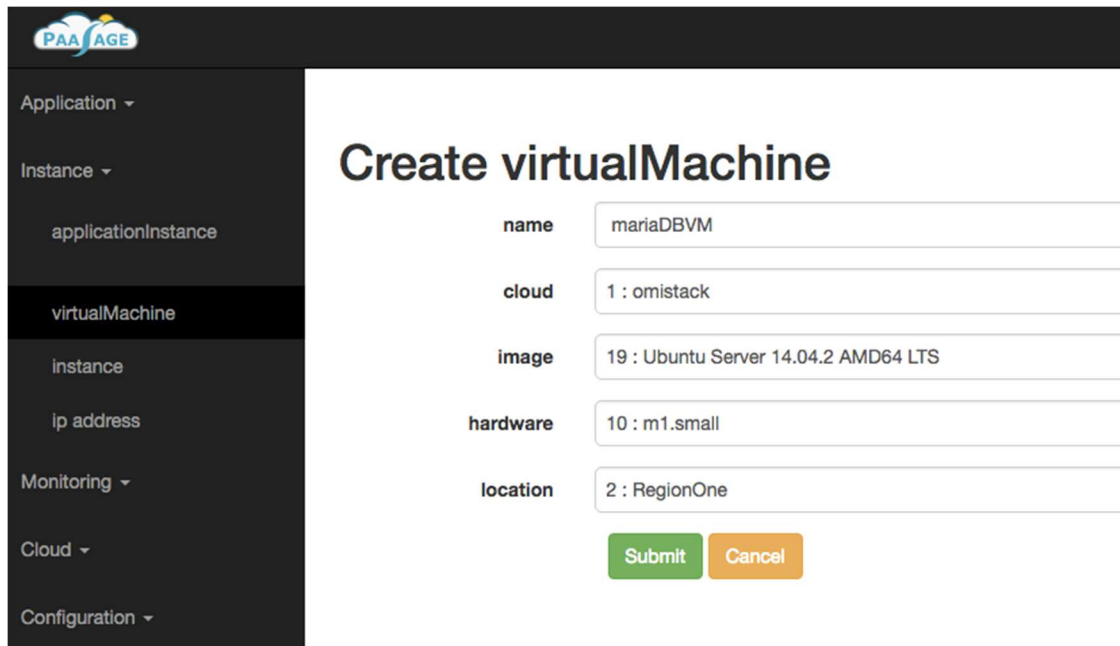
Figure 13: Wiki page

3.3.3.4 API Interaction

i. Starting virtual machines

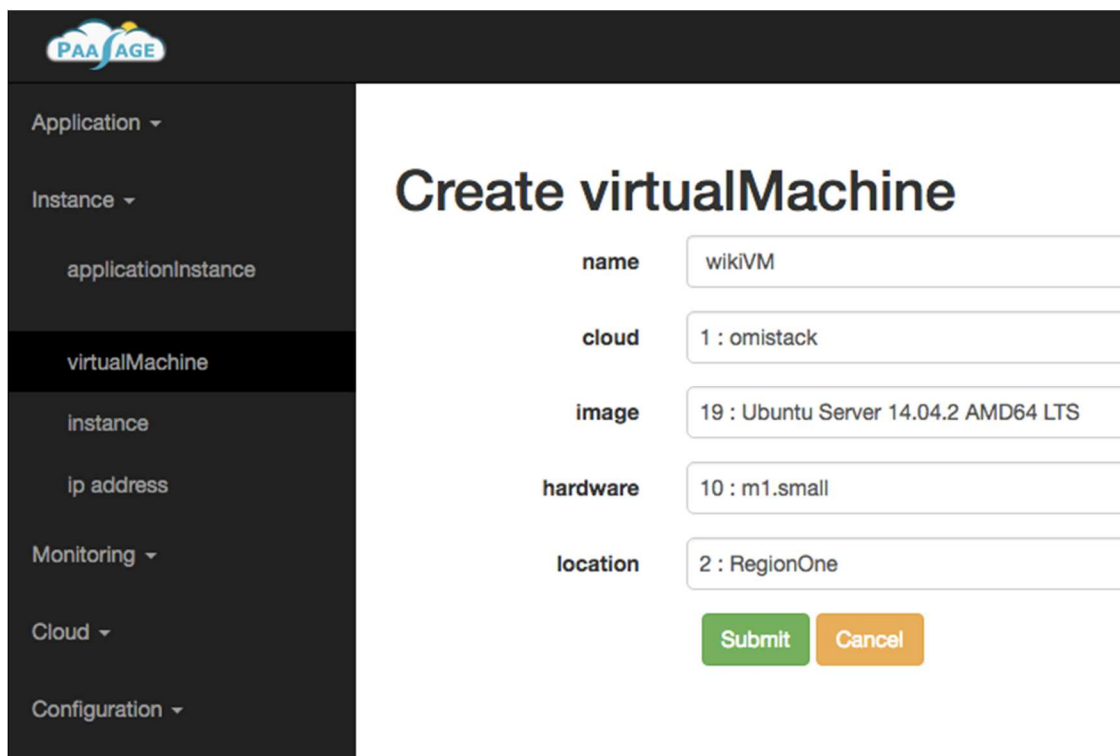
UI

Figure 14: Starting VM - lbVM



The screenshot shows the PAA AGE web interface. On the left is a dark sidebar with a menu containing 'Application', 'Instance', 'Monitoring', 'Cloud', and 'Configuration'. Under 'Instance', 'virtualMachine' is selected. The main content area is titled 'Create virtualMachine'. It contains a form with the following fields: 'name' (mariaDBVM), 'cloud' (1 : omistack), 'image' (19 : Ubuntu Server 14.04.2 AMD64 LTS), 'hardware' (10 : m1.small), and 'location' (2 : RegionOne). At the bottom of the form are two buttons: 'Submit' (green) and 'Cancel' (orange).

Figure 15: Starting VM - mariaDBVM



The screenshot shows the PAA AGE web interface. On the left is a dark sidebar with a menu containing 'Application', 'Instance', 'Monitoring', 'Cloud', and 'Configuration'. Under 'Instance', 'virtualMachine' is selected. The main content area is titled 'Create virtualMachine'. It contains a form with the following fields: 'name' (wikiVM), 'cloud' (1 : omistack), 'image' (19 : Ubuntu Server 14.04.2 AMD64 LTS), 'hardware' (10 : m1.small), and 'location' (2 : RegionOne). At the bottom of the form are two buttons: 'Submit' (green) and 'Cancel' (orange).

Figure 16: Starting VM – wikiVM

REST

```
{
  "name": "mariaDBVM",
  "cloud": 1,
  "image": 19,
  "hardware": 10,
  "location": 2
}
```

```

    }

    {
      "name": "wikiVM",
      "cloud": 1,
      "image": 19,
      "hardware": 10,
      "location": 2
    }

    {
      "name": "lbVM",
      "cloud": 1,
      "image": 19,
      "hardware": 10,
      "location": 2
    }
  }

```

colosseum-client

```

final VirtualMachine mariaDBVM = client.controller(VirtualMachine.class).create(
    VirtualMachineBuilder.of(mariaDBVirtualMachineTemplate)
        .name("mariaDBVM"))
    .build();

final VirtualMachine wikiVM = client.controller(VirtualMachine.class).create(
    VirtualMachineBuilder.of(wikiVirtualMachineTemplate)
        .name("wikiVM").build());

final VirtualMachine lbVM = client.controller(VirtualMachine.class).create(
    VirtualMachineBuilder.of(loadBalancerVirtualMachineTemplate)
        .name("lbVM").build());

```

ii. Creating the application instance

REST

```

{
  "application": 1
}

```

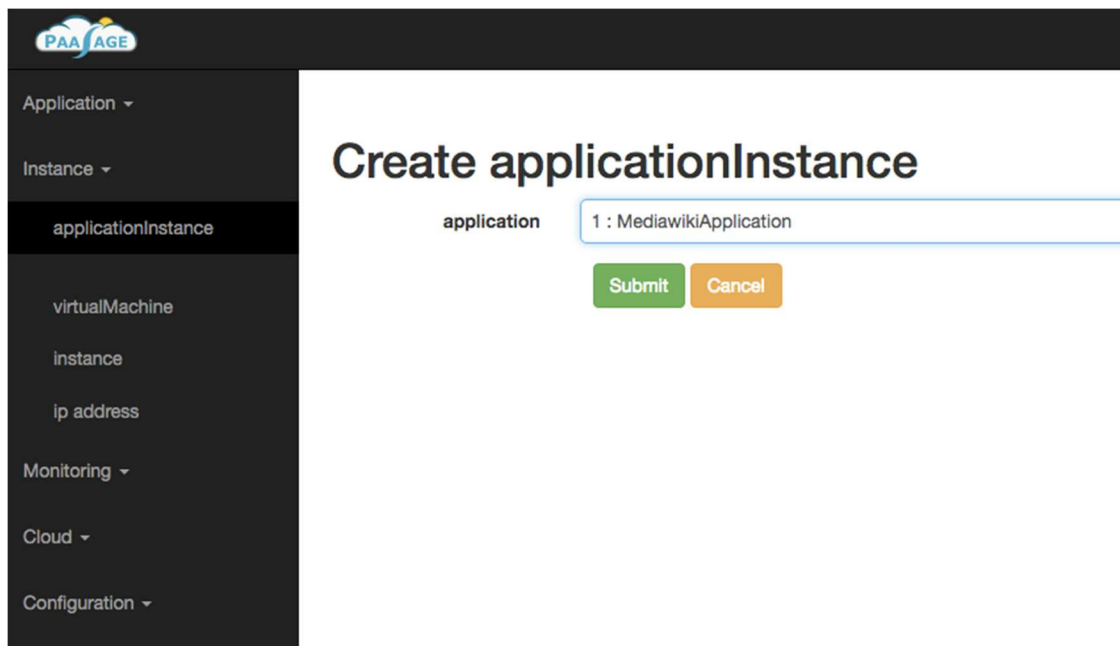
colosseum-client

```

final ApplicationInstance appInstance = client.controller(ApplicationInstance.class)
    .create(new ApplicationInstanceBuilder().application(application.getId()).build());

```

UI

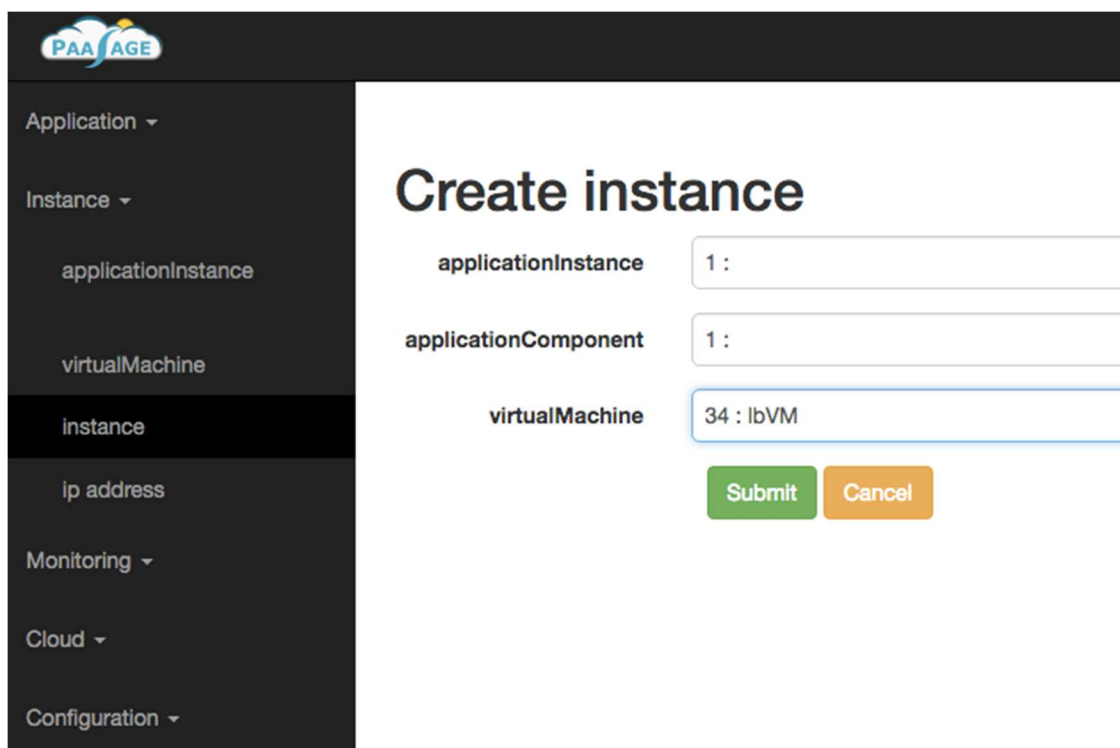


The screenshot shows the PAA AGE web interface. On the left is a dark sidebar with a menu. The 'Instance' dropdown is expanded, and 'applicationInstance' is selected. The main content area has a title 'Create applicationInstance'. Below the title, there is a form with a label 'application' and a text input field containing '1 : MediawikiApplication'. At the bottom of the form are two buttons: 'Submit' (green) and 'Cancel' (orange).

Figure 17: Creating applicationInstance

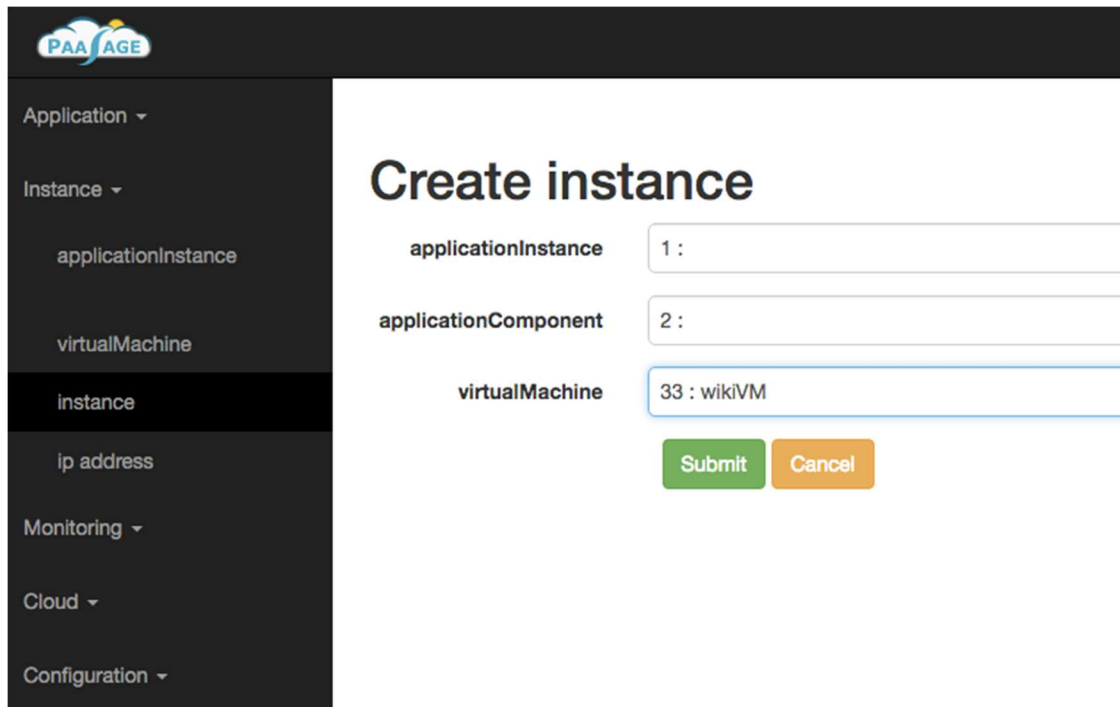
iii. Creating the application component instances

UI



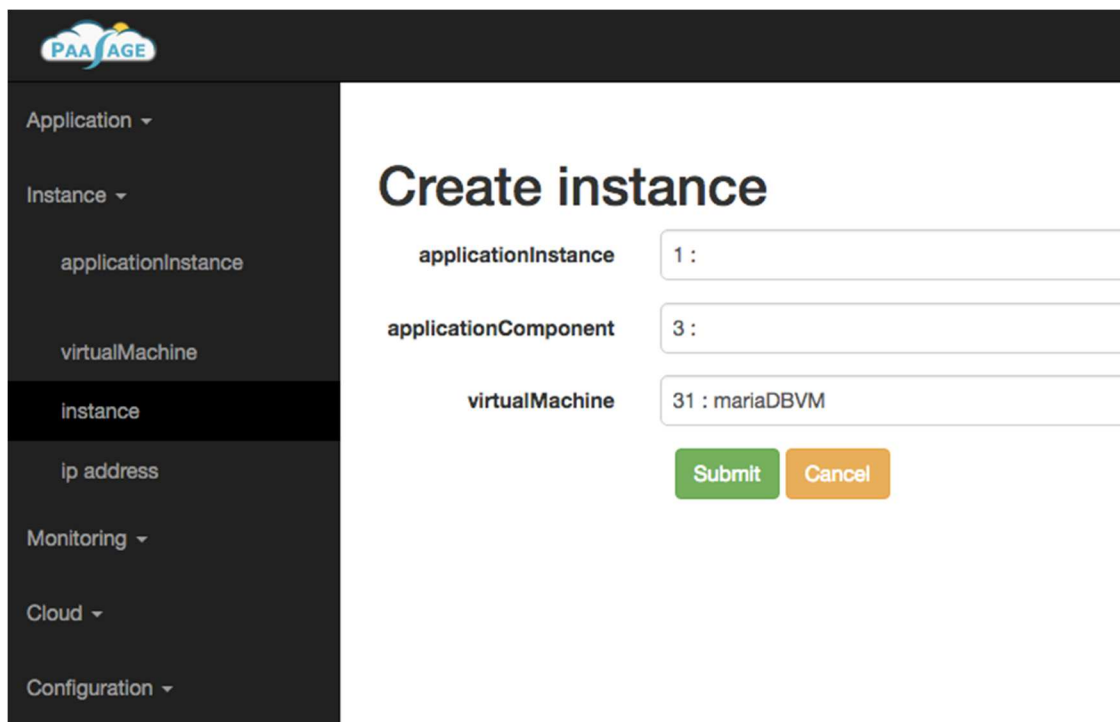
The screenshot shows the PAA AGE web interface. On the left is a dark sidebar with a menu. The 'Instance' dropdown is expanded, and 'instance' is selected. The main content area has a title 'Create instance'. Below the title, there are three form fields. The first is labeled 'applicationInstance' with a text input field containing '1 :'. The second is labeled 'applicationComponent' with a text input field containing '1 :'. The third is labeled 'virtualMachine' with a text input field containing '34 : lbVM'. At the bottom of the form are two buttons: 'Submit' (green) and 'Cancel' (orange).

Figure 18: Creating instance - lbVM



The screenshot shows the PAA AGE web interface. On the left is a dark sidebar with a navigation menu containing: Application, Instance, applicationInstance, virtualMachine, instance (highlighted), ip address, Monitoring, Cloud, and Configuration. The main content area is titled 'Create instance' and contains three input fields: 'applicationInstance' with the value '1', 'applicationComponent' with the value '2', and 'virtualMachine' with the value '33 : wikiVM'. At the bottom of the form are two buttons: 'Submit' (green) and 'Cancel' (orange).

Figure 19: Creating instance - wikiVM



The screenshot shows the PAA AGE web interface. On the left is a dark sidebar with a navigation menu containing: Application, Instance, applicationInstance, virtualMachine, instance (highlighted), ip address, Monitoring, Cloud, and Configuration. The main content area is titled 'Create instance' and contains three input fields: 'applicationInstance' with the value '1', 'applicationComponent' with the value '3', and 'virtualMachine' with the value '31 : mariaDBVM'. At the bottom of the form are two buttons: 'Submit' (green) and 'Cancel' (orange).

Figure 20: Creating instance – mariaDBVM

REST

```
{
  "applicationInstance": 1,
  "applicationComponent": 1,
  "virtualMachine": 34
}
```

```

{
  "applicationInstance": 1,
  "applicationComponent": 2,
  "virtualMachine": 33
}

{
  "applicationInstance": 1,
  "applicationComponent": 3,
  "virtualMachine": 31
}

```

colosseum-client

```

    final Instance lbInstance = client.controller(Instance.class).create( new
InstanceBuilder().applicationComponent(loadBalancerApplicationComponent.getId()
())

    .applicationInstance(appInstance.getId()).virtualMachine(lbVM.getId()).build());

    final Instance wikiInstance = client.controller(Instance.class).create( new
InstanceBuilder().applicationComponent(wikiApplicationComponent.getId())

    .applicationInstance(appInstance.getId()).virtualMachine(wikiVM.getId()).build());

    final Instance dbInstance = client.controller(Instance.class).create( new
InstanceBuilder().applicationComponent(mariaDBApplicationComponent.getId())

    .applicationInstance(appInstance.getId()).virtualMachine(mariaDBVM.getId())
    .build());

```

3.3.4 Java Example

3.3.4.1 Introduction

To make it easy to test cloudiator we have created a java code sample that will automatically execute the steps mentioned by this tutorial.

3.3.4.2 Installation

Simply download the latest build from [our Jenkins](#). You will need to have a Java JRE 8 installed to run it.

3.3.4.3 Configuration

You will need to create a configuration file. A template for the configuration file can be downloaded at [Github](#) or copied from below.

```

# Colosseum Configuration
colosseum.url = http://{ip-of-colosseum}:9000/api
colosseum.user = john.doe@example.com

```

```

colosseum.password = admin
colosseum.tenant = admin

# Cloud configuration

# Comma seperated list of all clouds
clouds = myCloud

## all configuration options should be present by using cloudName +
PropertyName ##

### myCloud

## The name for the cloud
myCloud.cloud.name = myCloud

## The endpoint of the cloud
myCloud.cloud.endpoint = http://endpoint.com

## The username for this cloud
myCloud.cloud.credential.username = myUser

## The credential for this cloud
myCloud.cloud.credential.password = topSecret

## The name for the api
myCloud.api.name = MyApi

## The driver for this cloud
myCloud.api.internalProviderName = openstack-nova

## ID of the image
myCloud.image.providerId = 9c154d9a-fab9-4507-a3d7-21b72d31de97

## ID of the location
myCloud.location.providerId = RegionOne

## ID of the hardware
myCloud.hardware.providerId = 3

## The login for the image
myCloud.image.loginName =

## A comma seperated list of properties for this cloud
myCloud.properties =

```

3.3.4.4 Running the example

To run the example simply execute:

```
java -Dconfig.file=path-to-your-config-file -jar colosseum-example-
jar-with-dependencies.jar
```

The program will automatically exit as soon as the mediawiki installation is running.

4 Camel Model Creation

4.1 Overview

The Cloud Application Modeling and Execution Language (CAMEL) [D2.1.3] super-DSL has been formed by aggregating pre-existing domain-specific languages (DSLs) (e.g., CloudML [CloudML] and Saloon feature model [SALOON]) as well as new ones developed in the context of the PaaSage project (e.g., Scalability Rule Language (SRL) [SRL]). This super-DSL provides significant support to the model-driven engineering approach adopted by the PaaSage project in order to facilitate the whole multi-cloud application management lifecycle. To this end, CAMEL is able to capture various aspects in the latter lifecycle, including deployment and non-functional requirements as well as scalability rules.

This document focuses mainly on the latter three aspects as they reflect the needs of application developers/owners, which guide the deployment plan derivation process, as well as the way the way this deployment plan can be evolved during application runtime. Apart from the aforementioned three main aspects, additional ones are also outlined by relying solely on the explication of the way respective aspect-specific modelling constructs can be specified such that are re-used to support the modelling of these three aspects.

While there are different ways via which CAMEL models can be specified, the CAMEL Textual Editor is used here. This relies on the textual syntax of CAMEL. This is in accordance to the latest documentation in CAMEL (see [D2.1.3]) as well as to the fact that, as the main target users for this language are devops users, such type of users will benefit the most from textual rather than graphical-based editors. Instructions about how to install and use the CAMEL Textual Editor can be found in the CAMEL documentation at <http://camel-dsl.org/documentation/>.

In the following, we adopt the Scalarm use case as a running example to exemplify how to specify CAMEL models in textual syntax. The complete Scalarm CAMEL model in textual syntax can be downloaded at:

<https://tuleap.ow2.org/plugins/git/paasage/camel?p=camel.git&a=blob&h=421be9e1caa955ee9d15725a6c26966cc5df9e9f&hb=f8905ee94cbef60fdf49a6aabe274e33b32ff022&f=examples/Scalarm.camel>.

Scalarm stands for Massively Scalable Platform for Data Farming and intends to fulfil the following requirements:

- support to all phases of a data farming experiment, starting from the experiment design phase, through simulation execution and progress monitoring, to statistical analysis of results,
- support of different size experiments from dozens to millions of simulations through massive scalability,
- support for heterogeneous computational infrastructure including private servers, computer clusters, grids and clouds.

Scalarm's architecture utilizes a service-oriented approach with additional modification, which addresses the scalability requirement.

Pre-Requisites

In order to follow this tutorial, it is recommended that the CAMEL Textual Editor is installed and launched in your system. This will enable you to copy the models illustrated in this tutorial and play with them such that you get accustomed and learn CAMEL. You should also be aware of some of the main features of this editor (see last section in this document as well as <https://eclipse.org/Xtext/#feature-overview>), such as auto-completion, which can assist you in the rapid specification and updating of your models. The prospective reader does not need to have any knowledge of CAMEL to understand the content of this tutorial but in some cases it is recommended that the user reverts to the [CAMEL documentation](#) in order to fully comprehend some relevant notions or inspect further details, if needed – this might happen in some cases as the goal of this tutorial is not to cover CAMEL in its entirety.

Audience

We consider that the specification of a CAMEL application model involves input from three types of users. To this end, this tutorial targets all of them. These user types are the following:

Application Designer: This type of user is expected to have knowledge of the main deployment and non-functional requirements of the application at hand, such as the application topology and requirements on application *response time*.

Business User: This type of user will set the higher level business requirements, such as the *cost* of application execution, and specific business policies/restrictions, such as data processing only using EU hosts.

Systems Admin: This type of user will know the wider technical context from an organizational perspective that the application should execute within. Requirements such as the wider security policies and technical details (e.g., OS-specific component configuration demands) can be set by this user type.

The end user monitors application execution in the Cloud against the requirements posed and could belong to either one of the user types above and as a result have needs for different levels of monitoring detail. The end user choice also depends on the organization's characteristics. It may also be possible that the setting of requirements is delegated to one of the above user types, although, as already stated, requirements from each type of user is needed for the PaaS platform to provide the optimum Cloud-based application management.

4.2 CAMEL creation

A *CamelModel* is a collection of sub-models mapping to the captured information aspects, including deployment details, end user requirements, monitoring measurements / metrics, scalability and organization details relevant for the multi-cloud application management lifecycle. Each aspect is mapped to a respective sub-model. All relevant aspects and corresponding sub-models, associated to the different types of user requirements, policies and rules that can be specified, are discussed in detail in the sequel in different sub-sections.

4.2.1 Deployment Aspect – DeploymentModel

A *DeploymentModel* is a collection of *DeploymentElements*. A deployment element can be a *Component*, a *Communication*, or a *Hosting*. A deployment element can refer to *Configurations*, which represent sets of commands to handle the deployment element's life cycle.

4.2.1.1 Components

A *Component* represents a reusable type of application component. A component can be an *InternalComponent* managed by the PaaS platform, or a requirement for a *VM* (short for virtual machine) offering maintained by the cloud provider. A virtual machine or a deployment model can be associated to a *VMRequirementSet*, which refers to a set of requirements for a single virtual machine or for all virtual machines, respectively, such as hardware, operating system and location requirements. These requirements are specified via a CAMEL requirement model (see also Listing 6).

Assume that we have to specify the Experiment Manager component of the example Scalarm use case. Listing 1 shows this specification in textual syntax where the corresponding component has been mapped to the definition of an internal component (i.e., an internal software component of the application) called *ExperimentManager*. provided communication *ExpManPort* represents that the Experiment Manager offers a communication port (443) via which its features can be exploited. required communication *StoManPortReq* and *InfSerPortReq* specify that the Experiment Manager requires features from the Information Service, which is another internal component, through port 11300 and from the Storage Manager through port 20001, respectively. The property mandatory of the latter signifies that the communication between the components should be obligatorily established, as the Execution Manager component needs to exploit the Storage Manager features from the very beginning of its initialization. As such, the Storage Manager will have to be started before the start of the Execution Manager.

Listing 1: Scalarm sample internal component

```

1 camel model ScalarmModel {
2
3 deployment model ScalarmDeployment {
4
5   internal component ExperimentManager {
6     provided communication ExpManPort {port: 443}
7     required communication StoManPortReq {port: 20001 mandatory}
8     required communication InfSerPortReq {port: 11300}
9     required host CoreIntensiveUbuntuGermanyHostReq
10
11    configuration ExperimentManagerConfiguration {
12      download: 'cd ~ && wget https://github.com/kliput/
scalarm_service_scripts/archive/paasage.tar.gz && sudo apt-get
update && sudo apt-get install -y groovy ant && tar -zxvf paasage.
tar.gz && cd ~/scalarm_service_scripts-paasage'
13      install: 'cd ~/scalarm_service_scripts-paasage && ./
experiment_manager_install.sh'
14      start: '~/scalarm_service_scripts-paasage/
experiment_manager_start.sh'
15    }
16  }
17  ...

```

Listing 1: Scalarm sample internal component

required host *CoreIntensiveUbuntuGermanyReq* indicates that the Experiment Manager needs to be hosted on a specific VM that satisfies certain requirements indicated in the description of this VM in the model. configuration *ExperimentManagerConfiguration* specifies the commands to handle the life cycle of the Experiment Manager. *download*, *install*, and *start* specify the Unix shell scripts for downloading, installing, and starting the Experiment Manager, respectively.

Then, assume that we have to specify the virtual machine on which the Experiment Manager needs to be deployed (which can be used for other VMs, if this is necessary). Listing 2 shows this specification in textual syntax. requirement set *CoreIntensiveUbuntuGermanyRS* specifies a reusable set of requirements for the VM being modelled. *quantitative hardware*, *os*, and *location* refer to the requirements *CoreIntensive*, *Ubuntu*, and *GermanyReq*, respectively, from the requirement model *ScalarmRequirement* (cf. Listing 6), mapping to the specification of the hardware requirements.

In vm *CoreIntensiveUbuntuGermany* the previous *requirementSet* is connected to the specification of the VM on which the Experiment Manager will be hosted.

provided host *CoreIntensiveUbuntuGermany* is the hosting port of the VM via which a respective component can be connected to indicate to the system that it should be hosted on that VM.

Listing 2: Scalarm sample vm

```
1 ...
2 requirement set CoreIntensiveUbuntuGermanyRS {
3   quantitative hardware: ScalarmRequirement.CoreIntensive
4   os: ScalarmRequirement.Ubuntu
5   location: ScalarmRequirement.GermanyReq
6 }
7
8 vm CoreIntensiveUbuntuGermany {
9   requirement set CoreIntensiveUbuntuGermanyRS
10  provided host CoreIntensiveUbuntuGermanyPort
11 }
12 ...
```

Listing 2: Scalarm sample vm

4.2.1.2 Communications

A *Communication* represents a reusable type of communication binding between a required and a provided communication port.

Assume that we have to specify the communication binding between the Experiment Manager and the Storage Manager. Listing 3 shows this specification in textual syntax. *Communication ExperimentManagerToStorageManager* specifies that reusable type of communication binding between the two internal components in question. *from .. to ..* block specifies that the communication binding is from the required communication port *StoManPortReq* of the component *ExperimentManager* to the provided communication port *StoManPort* of the component *StorageManager*. *type: REMOTE* specifies that the Experiment Manager and the Storage Manager is chosen to be deployed on separate virtual machine instances.

Listing 3: Scalarm sample communication

```
1 ...
2 communication ExperimentManagerToStorageManager {
3   from ExperimentManager.StoManPortReq to StorageManager.StoManPort
4   type: REMOTE
5 }
6 ...
```

Listing 3: Scalarm sample communication

4.2.1.3 Hostings

A *Hosting* represents a reusable type of containment binding between a required and a provided host port.

Assume that we have to specify the hosting binding between the Experiment Manager and the virtual machine *CoreIntensiveUbuntuGermany*. Listing 4 shows this specification in textual syntax.

hosting *ExperimentManagerToCoreIntensiveUbuntuGermany* in Listing 4 below, specifies such a hosting binding. from .. to .. block specifies that the hosting binding is from the required hosting port *CoreIntensiveUbuntuGermanyPortReq* of the component *ExperimentManager* to the provided hosting port *CoreIntensiveUbuntuGermanyPortReq* of the virtual machine *CoreIntensiveUbuntuGermany*.

Listing 4: Scalarm sample hosting

```

1  ...
2  hosting ExperimentManagerToCoreIntensiveUbuntuGermany {
3      from ExperimentManager.CoreIntensiveUbuntuGermanyPortReq to
        CoreIntensiveUbuntuGermany.CoreIntensiveUbuntuGermanyPort
4  }
5  ...

```

Listing 4: Scalarm sample hosting

4.2.2 Requirement Aspect – RequirementModel

A *RequirementModel* is a collection of *Requirements* which can be associated to an application and/or its main components. A requirement can be a *HardRequirement*, such as a service level objective (SLO) (e.g., response time < 100ms), which the PaaSage platform must satisfy at all costs, or a *SoftRequirement*, such as an optimization objective (e.g., minimize cost), which the platform will attempt to satisfy in the best possible way with no precise guarantees.

A *RequirementGroup* represents a tree-based requirement structure which can contain simple requirements as well as requirement sub-structures (i.e., complex requirements / requirement groups). The property *requirementOperator* of *RequirementGroup* represents the logical operator that is used to connect these requirements and it can be assigned two different alternative values mapping to known logical operators (AND (logical conjunction) or OR (logical disjunction)). A requirement group refers to an *Application* for which all the requirements must be satisfied.

Different kinds of requirements are supported by CAMEL, each analysed in respective subsections.

4.2.2.1 Hard requirements

A hard requirement can be attached to the specification of the requirements for a VM, or to a whole deployment model. In the former case, it specifies that instances of the VM must conform to the requirement in question. In the latter case, it specifies that all VM instances should be constrained according to that requirement.

Hardware, OS & Image and Provider Requirements

Two types of a *HardwareRequirement* exist. On the one hand, a *QualitativeHardwareRequirement* represents benchmarking constraints /

requirements with the intention to have a better classification and respective filtering of the VMs according to particular aspects like computation, memory, networking (e.g., computationally-large VMs vs memory-intensive VMs) or in an overall manner (by combining benchmark results over different aspects). As such, the respective properties *min-* and *maxBenchmark* of *QualitativeHardwareRequirement* of this class represent the range of benchmark results that a virtual machine instance must satisfy. On the other hand, a *QuantitativeHardwareRequirement* represents a set of constraints over the features of a VM (e.g., core number and RAM size) which can be used to perform typical filtering over the VM offerings across all cloud providers. For instance, in Listing 6, we can see that the user imposes for a respective hardware requirement that the number of cores provided should be from 8 to 32 while the size of main memory should range from 4096 to 8192 MB.

An *OsOrImageRequirement* can be specialized into an *OSRequirement* or an *ImageRequirement*. The former represents a requirement on the operating system run by a virtual machine, where the property *os* of *OSRequirement* represents the required operating system (e.g., “Ubuntu”, “Windows”, etc.), while the property *is64os* represents whether the operating system must conform to a 64bit architectures (e.g., x86-64). The latter represents a requirement on the image deployed on a virtual machine, where the property *imageId* of *ImageRequirement* represents the identifier of the required image.

A *ProviderRequirement* represents alternative cloud providers that could only be considered for the application deployment (e.g., Amazon and Rackspace only).

Location Requirements

A *LocationRequirement* refers to one or more *Locations*, which represent either geographical regions (e.g., a continent, a subcontinent, a country, or even a region) or cloud locations (i.e., regions and availability zones in Amazon cloud like us-east-1a).

Security Requirements

A *SecurityRequirement* refers to one or more *SecurityControls*, which represent the security controls that must be supported for a cloud provider in order to make it amenable for selection for use (see also Section 1.2.7 to comprehend the way security controls can be specified). Moreover, it can refer to an *Application* or *InternalComponent*, which represent the application or component on which the security controls must be enforced. If the security requirement refers to an application, then all cloud providers’ offerings and services, which are used by the application, must support the corresponding security controls. In case the security requirement refers to a single component, such as a virtual machine, then only offerings from cloud providers supporting the respective security controls can be selected for the particular component.

Scale Requirements

A *ScaleRequirement* can be referred to by a *ScalabilityRule* which dictates how corresponding scaling actions which can be performed are restrained. A *ScaleRequirement* can be a *HorizontalScaleRequirement*, which represents the minimum and maximum amount of instances allowed for a component, so that scale-out and scale-in actions will not exceed these bounds, respectively. Alternatively, it can be a *VerticalScaleRequirement*, which represents the minimum and maximum values allowed for virtual machine properties (e.g., number of CPU cores), so that scale-up and scale-down actions will not exceed these bounds, respectively.

Service Level Objectives

A *ServiceLevelObjective* represents an SLO. SLOs are used to specify measurable performance objectives (e.g., upper and/or lower thresholds regarding availability, response time, throughput, etc.) of a cloud service. In CAMEL, a *ServiceLevelObjective* refers to a *Condition*, such as a *MetricCondition*, which represents the metric condition that must be satisfied (i.e., the corresponding measurement values must not cross a particular threshold). Such a condition is specified via a metric model (see Section 1.2.4).

4.2.2.2 Soft requirements

Optimization Requirements

An *OptimisationRequirement* refers to a *Metric*, which represents how a metric should be optimized. Moreover, it refers to an *Application* or *InternalComponent*. The property *optimisationFunction* of *OptimisationRequirement* represents the optimization function applied to the metric and can be assigned the values of MINIMISE or MAXIMISE.

Listing 6: Scalarm requirement model

```

1 requirement model ScalarmRequirement {
2
3   quantitative hardware CoreIntensive {
4     core: 8..32
5     ram: 4096..8192
6   }
7
8   os Ubuntu {os: 'Ubuntu' 64os}
9
10  location requirement GermanyReq {
11    locations [ScalarmLocation.DE]
12  }
13
14  horizontal scale requirement HorizontalScaleSimulationManager {
15    component: ScalarmModel.ScalarmDeployment.SimulationManager
16    instances: 1 .. 5
17  }
18
19  slo CPUMetricSLO {
20    service level: ScalarmModel.ScalarmMetric.CPUMetricCondition
21  }
22
23  optimisation requirement
24    MinimisePerformanceDegradationOfExperimentManager {
25    function: MIN
26    metric: ScalarmModel.ScalarmMetric.
27    MeanValueOfResponseTimeOfAllExprimentManagersMetric
28    component: ScalarmModel.ScalarmDeployment.ExperimentManager
29    priority: 0.8
30  }
31
32  optimisation requirement MinimiseDataFarmingExperimentMakespan {
33    function: MIN
34    metric: ScalarmModel.ScalarmMetric.MakespanMetric
35    component: ScalarmModel.ScalarmDeployment.ExperimentManager
36    priority: 0.2
37  }
38
39  group ScalarmRequirementGroup {
40    operator: AND
41    requirements [ScalarmRequirement.CPUMetricSLO, ScalarmRequirement.
42    MinimisePerformanceDegradationOfExperimentManager,
43    ScalarmRequirement.MinimiseDataFarmingExperimentMakespan]
44  }
45 }

```

Listing 6: Scalarm provider model

Assume that we have to specify the requirements for the components of the Scalarm use case. Listing 6 show this specification in textual syntax. `quantitative hardware CoreIntensive` specifies that a *VM* must have from 8 to 32 CPU cores and from 4 to 8 GB of RAM. `os Ubuntu` specifies a quantitative hardware requirement prescribing that a *VM* must support the 64-bit edition of the Ubuntu operating system. `location requirement GermanyReq` specifies that a *VM* must be deployed in Germany. All three above requirements are referred to by the requirement set `CoreIntensiveUbuntuGermanyRS` in the deployment model `ScalarmDeployment` (cf. Listing 2). `locations` refers to the location `DE`, indicating the iso2 code for the country of Germany, in the location model `ScalarmLocation` (cf. Listing 7).

horizontal scale requirement
HorizontalScaleSimulationManager specifies that the component *SimulationManager* must scale horizontally between 1 and 5 instances. component refers to the internal component *SimulationManager* in the deployment model *ScalarmDeployment* (cf. Listing 2).

slo CPUMetricSLO is a specific SLO which is associated via the service level property to the metric condition *CPUMetricCondition* in the metric model *ScalarmModel* (cf. Listing 9). optimization requirement MinimisePerformanceDegradationOfExperimentManager specifies that the metric *MeanValueOfResponseTimeOfAllExperimentManagersMetric* of the component *ExperimentManager*, which is the average response time over all instances of the Experiment Manager application component, should be minimized and that this minimization has a priority of 0.8.

4.2.3 Location Aspect – Location Model

A *LocationModel* is a container for locations which is mainly used to represent location requirements. Two kinds of locations can be captured. On the one hand, physical locations are represented by *GeographicalRegions*. The property *name* of such a location represents its name in English, while the property *alternativeNames* represents alternative names of this location in other natural languages. A geographical region can refer to a parent region, which allows creating hierarchies of geographical regions. A *GeographicalLocation* can be a Country, which represents a distinct entity in the political geography.

On the other hand, a *CloudLocation* represents a virtual location that is specific to a particular cloud (e.g., the eu-west-1 availability zone in Amazon EC2). Similar to the geographical region, a cloud location can refer to a parent location, which allows creating hierarchies of cloud-specific locations (e.g., regions and encompassing availability zones in Amazon EC2).

Assuming that we have to specify the locations for the Scalarm use case, Listing 7 shows this specification in textual syntax. region EU specifies the region (continent) Europe. country DE specifies the country Germany. parent regions refers to the parent region of Europe for this country. Only the parents of a region need to be specified and not all possible ancestors. The ancestors of a country can be inferred in a recursive way by exploring the aforementioned parent-to-child relationship/property.

Listing 7: Scalarm location model

```

1 location model ScalarmLocation {
2
3   region EU {
4     name: 'Europe'
5   }
6
7   country DE {
8     name: 'Germany'
9     parent regions [ScalarmLocation.EU]
10  }
11
12  country UK {
13    name: 'United Kingdom'
14    parent regions [ScalarmLocation.EU]
15  }
16 }

```

Listing 7: Scalarm location model

4.2.4 Measurement/Metric Aspect – MetricModel

A metric model can be used to specific conditions over quality metrics or properties for applications and components (sw components and VMs), which can be associated to SLOs or (scalability rule) events, as well as all appropriate details to measure these metrics and properties. A condition can be specified by exploiting the following constructs analysed in the next sub-sections.

4.2.4.1 Metrics

A *Metric* is a standard of measurement which encapsulates all appropriate details for measuring non-functional properties. A *RawMetric* (e.g., raw response time) maps to the description of how raw measurements over a certain non-functional property (e.g., response time) can be produced. A *CompositeMetric*, in turn, represents an aggregated metric computed from other metrics. A metric refers to a *Unit* of measurement (e.g., the unit of SECONDS for the raw response time metric). In order to assist in checking the correctness of measurement values or their aggregations, a metric also refers to a *ValueType*, which represents the range of values the metric is allowed to take.

4.2.4.2 Metric Formulas

Each *CompositeMetric* refers to a *MetricFormula*, which explicates the computation formula used for deriving the composite metric measurements. For that purpose, a *MetricFormula* refers to one or more *MetricFormulaParameters*, which constitute its input, as well as to a pre-defined function to be applied on this input. There exist three kinds of parameters: *constants*, *Metrics*, or *MetricFormulas*. As such, a *MetricFormula* represents a measurement aggregation tree over particular metrics connecting different sub-formulas into a coherent whole.

4.2.4.3 Properties

Any *Metric* also refers to a measurable *Property*, i.e., the non-functional property of a component or an application that is measured by this metric. The attribute type represents the kind of property, where a value of MEASURABLE represents that the property can be measured, e.g., in the case of response time or CPU load, while a value of ABSTRACT represents that the property is not measurable. An abstract property that is not measurable can be realized by

more concrete and possibly measurable properties. In this way, the construction of property hierarchies is supported.

4.2.4.4 Metric Conditions

A *MetricCondition* represents a constraint imposed on a metric. A constraint is violated when the respective condition threshold is not met by the produced measurements of this metric. The violation of a metric condition may lead to the triggering of a simple, non-functional event, which might be part of the overall event pattern of a scalability rule, and/or to the violation of an SLO.

4.2.4.5 Property Conditions

A *PropertyCondition* represents a condition on a non-functional property. This way, it is possible to specify, e.g., constraints on the cost for the whole application or one or more of its components. Then, it is up to the PaaSage platform to interpret these constraints appropriately in order to derive the required property values (e.g., based on a particular internal to the platform metric used for producing the respective property value).

4.2.4.6 Condition Contexts

A condition, either pertaining to a metric or to a property, refers to a particular *ConditionContext*, which represents the context under which it should hold. The context explicates whether the condition must be enforced on the whole application or a particular component/VM. It also indicates for how many instances of the application or component/VM the condition must be checked. Two different types of quantification are distinguished: relative, in the form of percentages over the number of instances for an application or a component, and absolute, in the form of the actual number of instances for these applications or components.

4.2.4.7 Metric Context

A *MetricContext* is a condition context that also refers to the metric to be used for evaluating a respective condition as well as to information regarding the measurement schedule and window for this metric. For a composite metric, a *CompositeMetricContext* includes a reference to the contexts of the composing metrics of this metric. For a raw metric, a *RawMetricContext* represents a reference to the sensor that produces the measurements of this metric. The PaaSage runtime generates contextual information whenever possible so that it is not necessary to create all composing contexts by hand. This is possible as some information is inherited from the composite metric's context to its composing metrics' contexts (actually scheduling and window of measurement information).

Consequently, the definition of a context is only obligatory when information should not be inherited but differentiated for a specific composing context. For example, if we have specified the context of raw availability, the context of raw uptime (component of raw availability) does not need to include measurement scheduling and window information (e.g., measure the metric every 10 seconds) as this will be identical to the one encompassed in the availability's context.

4.2.5 Scalability Aspect – ScalabilityModel

A scalability model encompasses the specification of a set of scalability rules, regulating the adaptive runtime behaviour of a particular application, along with the events used to trigger them as well as the scaling actions executed upon this triggering. These three latter constructs are analysed in more detail below in separate subsections.

4.2.5.1 Scalability Rules

A *ScalabilityRule* associates an *Event* and a set of *Actions*. The *Event* represents either a single event or an event pattern/aggregation that triggers the execution of the actions. The *Actions* either specify which components and virtual machines should be scaled (i.e., case of scaling actions) and how global deployment decisions are made (i.e., for event creation actions). In the case where local adaptation fails or scalability limits based on given scaling requirements have been reached (that need to be associated to the respective *ScalabilityRule*). A scalability rule also refers to *Entities*, such as the user or the organization, which has specified it.

4.2.5.2 Actions

An *Action* can be specialized into a *ScalingAction* or an *EventCreationAction*. The *ScalingAction*, in turn, can be specialized into a *HorizontalScalingAction* or a *VerticalScalingAction*. The *HorizontalScalingAction* refers to a *VM* and an *InternalComponent* (both specified via the deployment package). In case such an action is executed, the specified component is scaled (out or in) along with the virtual machine hosting it. The property *count* defines the number of additional instances to create, or the number of existing instances to destroy. In contrast to horizontal scaling, the *VerticalScalingAction* refers to a concrete *VMInstance*. The properties named by the **Update* pattern define the amount of virtual resources (e.g., CPU cores, RAM, etc.) to be added to or removed from the virtual machine instance. An *EventCreationAction* signifies via the creation of an event that the scaling actions are not sufficient to maintain the target service level of a multi-cloud application. For instance, a multi-cloud application may still violate the target response time defined in an SLO despite the scale-out or scale-up actions performed.

4.2.5.3 Events

Events can be simple or composite (i.e., event patterns). A *SimpleEvent* can be specialized into a *FunctionalEvent* or a *NonFunctionalEvent*. The *FunctionalEvent* represents a functional error (e.g., a virtual machine or a component has failed). A *NonFunctionalEvent* refers to a metric or property condition that is triggered when this condition is violated. (e.g., the response time of a component exceeds the target response time in an SLO). The *NonFunctionalEvent* refers to a *MetricCondition*, which defines the threshold for the metric. On the other hand, an event pattern is an aggregation of events based on logical or time-based operators (e.g., a logical conjunction of two other events via the AND logical operator).

Listing 8 shows the Scalarm’s scalability model in textual syntax. This model encompasses one scalability rule that associates one binary event pattern with a scale-

out action, while it is restricted by the scaling policy specified in Listing 6. The semantics of this rule specifies that we need to scale-out the *SimulationManager* component of Scalarm when particular bounds/conditions of two metrics are violated, mapping to respective events aggregated via a logical conjunction into the corresponding binary event pattern, provided that the number of instances of this component is less than 5. The scale-out action specification indicates important information about the scaling, such as the scale action type, which is the component to be scaled and onto which the VM type/offering will be hosted.

Listing 8: Scalarm scalability model

```

1 scalability model ScalarmScalability {
2
3   horizontal scaling action HorizontalScalingSimulationManager {
4     type: SCALE_OUT
5     vm: ScalarmModel.ScalarmDeployment.CPUIntensiveUbuntuGermany
6     internal component: ScalarmModel.ScalarmDeployment.
      SimulationManager
7   }
8
9   non-functional event CPUAvgMetricNFEAll {
10    metric condition: ScalarmModel.ScalarmMetric.
      CPUAvgMetricConditionAll
11    violation
12  }
13
14  non-functional event CPUAvgMetricNFEAny {
15    metric condition: ScalarmModel.ScalarmMetric.
      CPUAvgMetricConditionAny
16    violation
17  }
18
19  binary event pattern CPUAvgMetricBEPAnd {
20    left event: ScalarmModel.ScalarmScalability.CPUAvgMetricNFEAll
21    right event: ScalarmModel.ScalarmScalability.CPUAvgMetricNFEAny
22    operator: AND
23  }
24
25  scalability rule CPUScalabilityRule {
26    event: ScalarmModel.ScalarmScalability.CPUAvgMetricBEPAnd
27    actions [ScalarmModel.ScalarmScalability.
      HorizontalScalingSimulationManager]
28    scale requirements [ScalarmRequirement.
      HorizontalScaleSimulationManager]
29  }
30 }
31
32 requirement model ScalarmRequirement {
33
34   horizontal scale requirement HorizontalScaleSimulationManager {
35     component: ScalarmModel.ScalarmDeployment.SimulationManager
36     instances: 1..5
37   }
38 }

```

Listing 8: Scalarm scalability model

Listing 9 shows the Scalarm's metric model in textual syntax, which encloses the specification of the event conditions involved in the previously analysed scalability rules, and the corresponding metrics encompassed in these conditions along with their scheduling information. The two metrics map to common information for two families of metrics: (a) a raw (sensor) metric measuring CPU load and (b) an average

CPU load metric; the latter metric will be instantiated with two different contexts, one with a window of five minutes, and another with a window of one minute. This is due to the semantics of the corresponding conditions mapping to these contexts which impose applying different bounds on the same composite metric with different measurement scheduling and window directives. In particular, one condition (*CPUAvgMetricConditionAll*) will be violated when the average CPU, computed every 1 minute with a sliding window of 5 minute, for all instances of the *SimulationManager* component is greater than 50%, while the other condition (*CPUAvgMetricConditionAny*) will be violated when the average CPU, computed every 1 minute with a sliding window of 1 minute, for any instance of *SimulationManager* is greater than 80%.

Listing 9: Scalarm metric model

```

1 metric model ScalarmMetric {
2
3   window Win5Min {
4     window type: SLIDING
5     size type: TIME_ONLY
6     time size: 5
7     unit: ScalarmModel.ScalarmUnit.minutes
8   }
9
10  window Win1Min {
11    window type: SLIDING
12    size type: TIME_ONLY
13    time size: 1
14    unit: ScalarmModel.ScalarmUnit.minutes
15  }
16
17  schedule Schedule1Min {
18    type: FIXED_RATE
19    interval: 1
20    unit: ScalarmModel.ScalarmUnit.minutes
21  }
22
23  schedule Schedule1Sec {
24    type: FIXED_RATE
25    interval: 1
26    unit: ScalarmModel.ScalarmUnit.seconds
27  }
28
29  property CPUProperty {
30    type: MEASURABLE
31    sensors [ScalarmMetric.CPUSensor]
32  }
33
34  sensor CPUSensor {
35    configuration: 'cpu_usage;de.uniulm.omi.cloudiator.visor.sensors.CpuUsageSensor'
36    push
37  }
38
39  raw metric CPUMetric {
40    value direction: 0
41    layer: IaaS
42    property: ScalarmModel.ScalarmMetric.CPUProperty
43    unit: ScalarmModel.ScalarmUnit.CPUUnit
44    value type: ScalarmModel.ScalarmType.Range_0_100
45  }
46
47  composite metric CPUAverage {
48    description: "Average of the CPU"
49    value direction: 1

```

```

50     layer: PaaS
51     property: ScalarmModel.ScalarmMetric.CPUProperty
52     unit: ScalarmModel.ScalarmUnit.CPUUnit
53
54     metric formula Formula_Average {
55         function arity: UNARY
56         function pattern: MAP
57         MEAN( ScalarmModel.ScalarmMetric.CPUMetric )
58     }
59 }
60
61 raw metric context CPUMetricConditionContext {
62     metric: ScalarmModel.ScalarmMetric.CPUMetric
63     sensor: ScalarmMetric.CPUSensor
64     component: ScalarmModel.ScalarmDeployment.SimulationManager
65     quantifier: ANY
66 }
67
68 raw metric context CPURawMetricContext {
69     metric: ScalarmModel.ScalarmMetric.CPUMetric
70     sensor: ScalarmMetric.CPUSensor
71     component: ScalarmModel.ScalarmDeployment.SimulationManager
72     schedule: ScalarmModel.ScalarmMetric.Schedule1Sec
73     quantifier: ALL
74 }
75
76 composite metric context CPUAvgMetricContextAll {
77     metric: ScalarmModel.ScalarmMetric.CPUAverage
78     component: ScalarmModel.ScalarmDeployment.SimulationManager
79     window: ScalarmModel.ScalarmMetric.Win5Min
80     schedule: ScalarmModel.ScalarmMetric.Schedule1Min
81     composing metric contexts [ScalarmModel.ScalarmMetric.
82     CPURawMetricContext]
83     quantifier: ALL
84 }
85
86 composite metric context CPUAvgMetricContextAny {
87     metric: ScalarmModel.ScalarmMetric.CPUAverage
88     component: ScalarmModel.ScalarmDeployment.SimulationManager
89     window: ScalarmModel.ScalarmMetric.Win1Min
90     schedule: ScalarmModel.ScalarmMetric.Schedule1Min
91     composing metric contexts [ScalarmModel.ScalarmMetric.
92     CPURawMetricContext]
93     quantifier: ANY
94 }
95
96 metric condition CPUMetricCondition {
97     context: ScalarmModel.ScalarmMetric.CPUMetricConditionContext
98     threshold: 80.0
99     comparison operator: >
100 }
101
102 metric condition CPUAvgMetricConditionAll {
103     context: ScalarmModel.ScalarmMetric.CPUAvgMetricContextAll
104     threshold: 50.0
105     comparison operator: >
106 }
107
108 metric condition CPUAvgMetricConditionAny {
109     context: ScalarmModel.ScalarmMetric.CPUAvgMetricContextAny

```



```

107     threshold: 80.0
108     comparison operator: >
109   }
110 }

```

Listing 9: Scalarm metric model

4.2.6 Security Aspect – SecurityModel

A *SecurityModel* is container of security-related constructs which can be exploited to specify security requirements and capabilities that can assist in the filtering of cloud providers during deployment plan reasoning. Such constructs are now analysed in detail.

A *SecurityControl* represents a technical or administrative countermeasure that aims to address security risks in a cloud-based application. Such a construct actually characterises high-level security requirements or capabilities that have to be satisfied or realised by the application owner or cloud provider, respectively. The *property* specification is used to specify textual descriptions of security controls in the CAMEL model. A security control can be linked to raw or composite security metrics which are specialisations of non-functional metrics (see Section 1.2.4). This kind of linkage enables the connection of high-level requirements or capabilities expressed via security controls to more concrete requirements or capabilities expressed via conditions on security metrics. As such, we can evaluate whether a particular security control is satisfied via assessing the respective conditions on metrics associated to this control. A security control is also associated to a security domain and sub-domain. The latter constructs can be exploited to perform a partitioning of security-related building blocks in terms of security controls, metrics and properties.

Listing 12: A set of security-related models which could be used to extend the Scalarm user case model

```

1 security model ScalarmSecurity {
2
3   domain IAM {
4     name: "Identity & Access Management"
5     sub-domains [ScalarmSecurity.IAM_CLCPM, ScalarmSecurity.IAM_CLCPM]
6   }
7
8   domain IAM_CLCPM {
9     name: "Credential Life Cycle/Provision Management"
10  }
11
12  domain IAM_UAR {
13    name: "User Access Revocation"
14  }
15 }

```

```

16  property IdentityAssurance {
17      description: "The ability of a relying party to determine, with
        some level of certainty, that a claim to a particular identity made
        by some entity can be trusted to actually be the claimant's true,
        accurate and correct identity."
18      type: ABSTRACT
19      domain: ScalarmSecurity.IAM
20  }
21
22  security control IAM_02 {
23      specification: "User access policies and procedures shall be
        established, and supporting business processes and technical
        measures implemented, for ensuring appropriate identity,
        entitlement, and access management for all internal corporate and
        customer (tenant) users with access to data and organisationally-
        owned or managed (physical and virtual) application interfaces and
        infrastructure network and systems components."
24      domain: ScalarmSecurity.IAM
25      sub-domain: ScalarmSecurity.IAM_CLCPM
26      security properties [ScalarmModel.ScalarmSecurity.
        IdentityAssurance]
27  }
28
29  security control IAM_11 {
30      specification: "Timely de-provisioning (revocation or modification
        ) of user access to data and organisationally-owned or managed (
        physical and virtual) applications, infrastructure systems, and
        network components, shall be implemented as per established
        policies and procedures and based on user's change in status (\eg,
        termination of employment or other business relationship, job
        change or transfer). Upon request, provider shall inform customer (
        tenant) of these changes, especially if customer (tenant) data is
        used as part the service and/or customer (tenant) has some shared
        responsibility over implementation of control."
31      domain: ScalarmSecurity.IAM
32      sub-domain: ScalarmSecurity.IAM_UAR
33      security properties [ScalarmModel.ScalarmSecurity.
        IdentityAssurance]
34  }
35
36  security capability SecCap {
37      controls [ScalarmSecurity.IAM_02, ScalarmSecurity.IAM_11]
38  }
39 }
40
41 requirement model ScalarmExtendedReqModel {
42
43     security requirement AllIAMsSupported {
44         controls [ScalarmSecurity.IAM_02, ScalarmSecurity.IAM_11]
45     }
46 }
47
48 organisation model AmazonExt {
49
50     provider Amazon {
51         www: "www.amazon.com"
52         email: "contact@amazon.com"
53         PaaS
54         IaaS
55
56         security capability [ScalarmModel.ScalarmSecurity.SecCap]
57     }
58
59 unit model ScalarmUnit {
60     time interval unit {sec: SECONDS}
61 }

```

Listing 12: A set of security related models

A security property is a kind of a non-functional property. *Certifiable* security properties can actually be measured/certified and are thus connected to respective security metrics. A *SecuritySLO* is a kind of SLO which involves security metrics and properties in its conditions.

Assume that we have to specify a security model for the Scalarm use case. Listing 12 above shows this specification model in textual syntax. domain IAM specifies the security domain of Identity & Access Management (IAM). domain IAM_CLCPM and IAM_UAR specify two sub-domains of IAM, namely Credential Life Cycle/Provision Management (CLCPM) and User Access Revocation (UAR), respectively. The property IdentityAssurance specifies an abstract security property associated with the security domain IAM.

Security control IAM_02 (related to the establishment of user-control access and policies at the cloud provider side) specifies a security control associated with the security sub-domain (CLCPM) and the property IdentityAssurance. Similarly, security control IAM_11 (related to the timely deprovisioning of user access) specifies a security control associated with the security sub-domain (UAR) and the property IdentityAssurance. Note that these security controls are part of the set of security controls of the Cloud Control Matrix³ identified by the Cloud Security Alliance (CSA)⁴. security capability SecCap specifies a security capability associated with the security controls IAM_02 and IAM_11. Finally, the organisation model AmazonExt refers to the security capability SecCap, which specifies that the Amazon provider supports this security capability.

4.2.7 Type Aspect - TypeModel

The type model includes the specification of values as well as of the types to which these values conform. Such types can be associated to metrics and feature attributes.

A *Value* represents a generic value. It can be specialised into a *NumericValue*, *StringValue*, *BooleanValue*, and *EnumerateValue*. A numeric value can be further split into the *IntValue*, *DoubleValue*, and *FloatValue*. A numeric value can also be split into *NegativeInf* and *PositiveInf*, which represent negative and positive infinity, respectively, and can be used for specifying one of the two bounds of range-based value types.

The *StringValue* and *BooleanValue* classes represent string and boolean values, respectively. On the other hand, the *EnumerateValue* represents an enumerated value. The property *name* represents the string associated with the value, while the property *value* represents an integer associated with the value (or position in the enumeration).

³ <https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3-0-1/>

⁴ <http://www.cloudsecurityalliance.org>

ValueType represents a generic value type. It can be specialised into a *StringValue*, *BooleanValue*, *Enumeration*, *List*, *Range* and *RangeUnion*. *StringValue* and *BooleanValue* represent string and boolean value types, respectively. *Enumeration* represents an enumeration type that can take *EnumerateValues*.

List represents a list type having members of which that can be of a basic (i.e., a numeric, string, or boolean value) or complex value type (e.g., an enumeration or a range). The property *primitiveType* represents the basic value type, and it has to be used in the first case. The referenced *type* represents the complex value type, and it has to be used in the second case.

A *Range* represents a range-based value type. It has two references *lowerLimit* and *upperLimit*. A *limit* represents an actual bound, either upper or lower, of a range. The property *included* indicates whether the limit's value is included or not in the range. The *RangeUnion* represents a union of range-based value types. It refers to the contained range-based value types as well as to the primitive type that is common across all the contained value types.

Assume we have to record the types of the Scalarm use case. Listing 14 shows this specification in textual syntax. The range statements specify two integer-based ranges and one double-based range. The first range is associated as a value type to the *CPU* metric (cf. Listing 9 to represent that CPU metric values should be between 0 and 100, both included). The second range is associated as a value type to the *ResponseTime* metric to signify that the values of this metric should be between 0, not included (i.e., between 1), and 10000, included. The third range is associated to the *Availability* metric and signifies that the respective metric values should be between 0.0 and 100.0, where both bound/limit values are included.

Listing 14: Scalarm type model

```
1 type model ScalarmType {
2
3   range Range_0_100 {
4     primitive type: IntType
5     lower limit {int value 0 included}
6     upper limit {int value 100}
7   }
8
9   range Range_0_10000 {
10    primitive type: IntType
11    lower limit {int value 0}
12    upper limit {int value 10000 included}
13  }
14
15  range DoubleRange_0_100 {
16    primitive type: DoubleType
17    lower limit {double value 0.0 included}
18    upper limit {double value 100.0 included}
19  }
20 }
```

Listing 14: Scalarm type model

4.2.8 Unit Aspect – UnitModel

A *UnitModel* is a collection of units that can be associated to metrics of a metric model or attributes of a provider model. A *Unit* represents an abstract unit. It can be specialised into the following classes:

- *CoreUnit*, which represents the unit of CPU cores
- *MonetaryUnit*, which represents a monetary unit (e.g., EUROS)
- *RequestUnit*, which represents the unit of number of requests
- *StorageUnit*, which represents the unit of storage (e.g., BYTES)
- *ThroughputUnit*, which represents the unit of throughput (e.g., REQUESTS_PER_SECOND)
- *TimeIntervalUnit*, which represents the unit of time interval (e.g., SECONDS)
- *TransactionUnit*, which represents the number of transactions
- *Dimensionless*, which represents a unit without dimension (e.g., a unit of PERCENTAGE is dimensionless).

Assume that we have to specify the units of the Scalarm use case. Listing 15 shows this specification in textual syntax. The unit model encompasses seven units that are used in the metric model. The specification of each unit follows the pattern: <unit_class> <unit_name>: <unit_type> (where the latter is an enumeration of all possible unit types). For instance, monetary unit {Euro: EUROS} specifies a monetary unit named “euro” and typed EUROS.

Listing 15: Scalarm unit model

```
1 unit model ScalarmUnit {
2
3   monetary unit {Euro: EUROS}
4
5   throughput unit {SimulationsPerSecondUnit: TRANSACTIONS_PER_SECOND}
6
7   time interval unit {ResponseTimeUnit: MILLISECONDS}
8
9   time interval unit {ExperimentMakespanInSecondsUnit: SECONDS}
10
11  transaction unit {NumberOfSimulationsLeftInExperimentUnit:
12    TRANSACTIONS}
13
14  dimensionless {AvailabilityUnit: PERCENTAGE}
15
16  dimensionless {CPUUnit: PERCENTAGE}
17 }
```

4.3 Summary

In this section, we have introduced CAMEL in the context of the specification of non-functional and deployment requirements as well as scalability rules. For each aspect, we have described the main modelling concepts, their properties and relations, while we have provided concrete examples of the respective aspect-specific part of the CAMEL syntax by relying on the Scalarm use case.

Through the use of the CAMEL textual editor, we believe that the prospective user does not only have access to many interesting editing services but also is presented with the capability to learn the CAMEL essentials without having to revert to any

extensive CAMEL documentation. The editor also enables rapid production of CAMEL models. The editor's services encompass capabilities for syntactic and semantic highlighting, domain validation reporting, auto-completion and clever suggestion (by also catering for user-intuitive cross-reference specification within or across CAMEL sub-models), while the automatic generation of the XMI CAMEL form is also supported. Such capabilities and the editing mode cater mainly *devops* and *admin* types of users as they are more close to the way these user types work.

The remaining user types, i.e., a business user, who is not comfortable with the editing mode can revert to the alternative ways to specify CAMEL models which are graphics-based. These latter ways include the default graphical tree-based CAMEL editor offered by the Eclipse Environment which can operate over the file system or the MDDB CDO Repository [D4.1.2], or a web-based editor developed via Eclipse's RAP⁵ technology which enables the on-line editing of CAMEL application (application + requirement) and organisation models over the MDDB CDO Repository.

⁵ Eclipse.org/rap

5 Social Network User Guide

The PaaSage Social Network (SN) is a social platform targeted at the creation of a community of users and developers. It enables users to exchange their experience and knowledge with respect to the PaaSage platform. Through the PaaSage SN, users can connect to other SN users and view their contributions, join groups, deploy applications, navigate through historical knowledge of previously executed application models, create application models by also utilizing useful modeling recommendations provided by the SN as well as comment and rate such applications models. SN is available for users (socialnetwork.paasage.eu) to create accounts and exploit the functionality exhibited by the SN. This section presents the basic functionality of the PaaSage Social Network and provides an overview of the pages and the actions that a user can perform. Furthermore, in the following subsections, all the required information about how to use and navigate through the PaaSage Social Network is supplied.

5.1 Site Sections

The fundamental view of the Social Network Web Site is shown in Table 2 below. In the *top-bar navigation menu* the user can navigate through the key elements of the SN, which are the home page, Models, Community. Home page is the entry page of the social network. Information about the activity of the user models is depicted here. Moreover, the *live feed* part, which is an infinite scroll, shows the latest activity of all the members of the social network. Models page includes application models described in the Camel Meta-model [D2.1.2]. Finally, the Community supplies the necessary functionality for users to ask questions, get feedback from other users and create / join groups. In the right corner of the *top-bar navigation menu*, the following shortcuts are supplied:







	My Area	The <i>My Area</i> section contains the user's configurations, runs, models, components, and credentials
	Notifications	Any system notification to the user.
	The user profile photo	Redirects to the user's profile.
	Friends	Redirects to the user's friends.
	Messages	The text messages sent from other users.
	Settings	The user's settings.

Table 2: Top-bar navigation menu shortcuts

The *Main Window* displays the main information that the user wants to see according to the user action/selection performed. For instance, if the user clicks the Community link from the top-bar, the *Main Window* will contain the home page of the community. The *Sidebar* section contains neighboring information about the *Main Window* and some actions that maybe the user wants to perform, such as filtering the *Main Window* information.

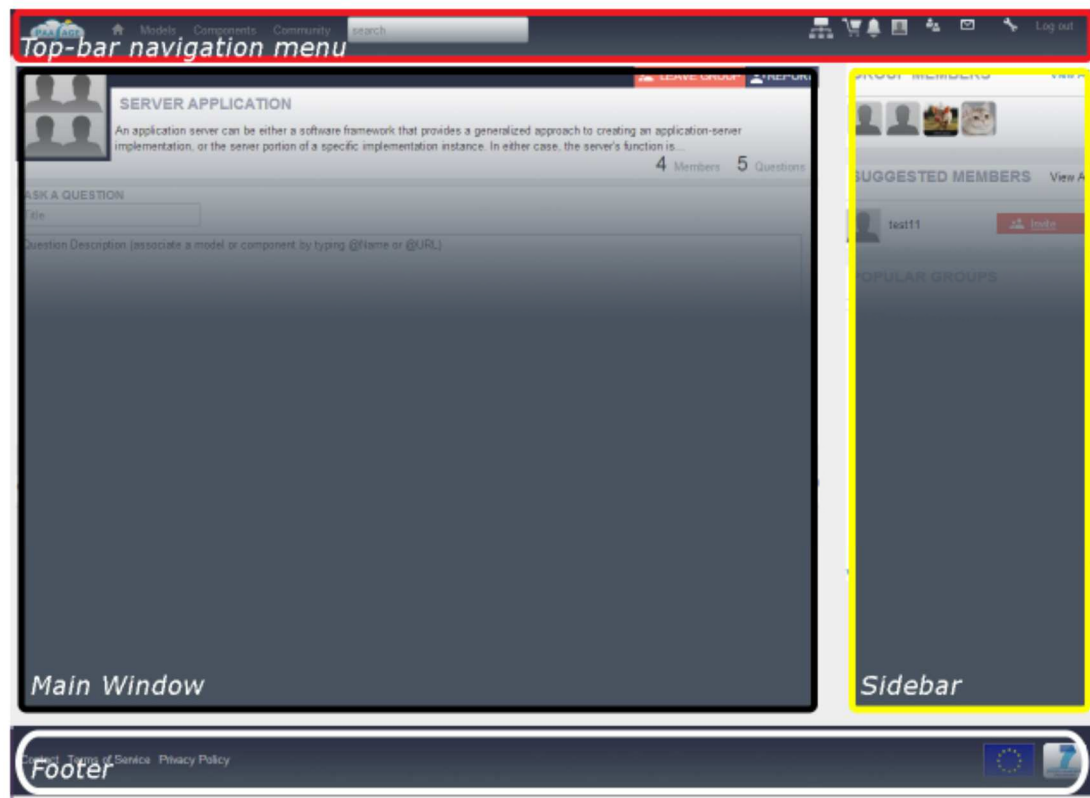


Figure 21: Site Sections: 1) The top-bar section provides the basic navigation options. 2) The Main Window provides the main information that the user desires to see. 3) The sidebar sections exist basically for additional, secondary information and 4) the footer of the site

Finally, the last element of the page is the *Footer* which is kept simple and provides the links to:

- Contact Information
- Terms of Service and
- Privacy Policy

5.2 User Login / Register

The introductory page of Social Network (SN), shown in Figure 22, has the basic functionality for login, register and additionally some information about the SN capabilities. If the user has an account on the SN, then he/she can log-in by typing his/her username or email as well as password in the top right section. If the user is a new visitor then he/she can register to the SN. The registration section is kept simple by enabling the user to provide only the necessary details for the sign up process

(display name, email address, username and password). It must be highlighted that the user can provide additional details in the profile settings pages which are analyzed in the following subsection. The easiest and quickest way for a new user of the SN to log-in is to make use of their Facebook or Twitter account. In that way an account is created in the SN taking information from the Facebook or Twitter account respectively. After the sign-up or log-in process, the user is redirected to the models home page, which is analyzed later on.

Figure 22: Log-in and Registration Page

Figure 23: User General Settings

5.3 User Profile

The user's profile page is shown in Figure 24. In the top of the main page the user can see the models he/she has contributed to the SN. Such contribution items are shown throughout all the web site in order to incentive the user to provide feedback to the community. After the contributions page part, the user can see the top discussion topics, nominated according to the rating of other users. In the next section of the main page body, the user can see his/her activity feed. Finally, in the sidebar section,

the user can add his/her skills (e.g., web application development) and areas of interest (e.g., servers).

When the mouse is over a tag in the skills or in areas of interest section, a remove button appears to assist the user in deleting the desired entry/tag.

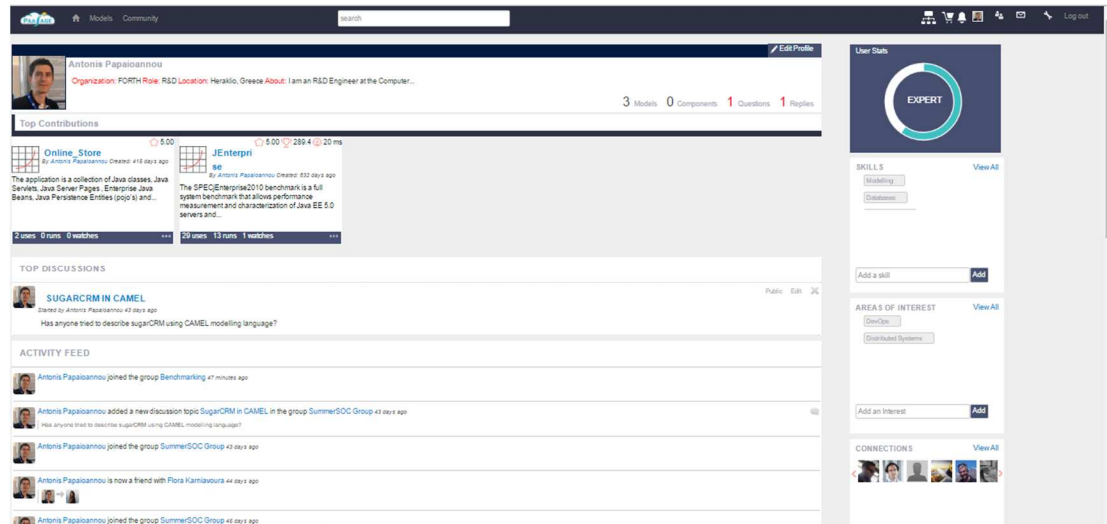


Figure 24: User's profile page

Bellow the area of interest there is a list with all the connections of the user. The *View all* link redirects to the friend's page, shown in Figure 25. The user's friends' page can be also accessed by clicking the *Friends* shortcut at the top-bar menu. In this page the user can perform three actions:

- See his/her friends' profiles
- Send a message to a friend or
- Remove a particular friend from his/her friend list.

The actions *Send* and *Remove* appear when the user mouse is over a specific user.

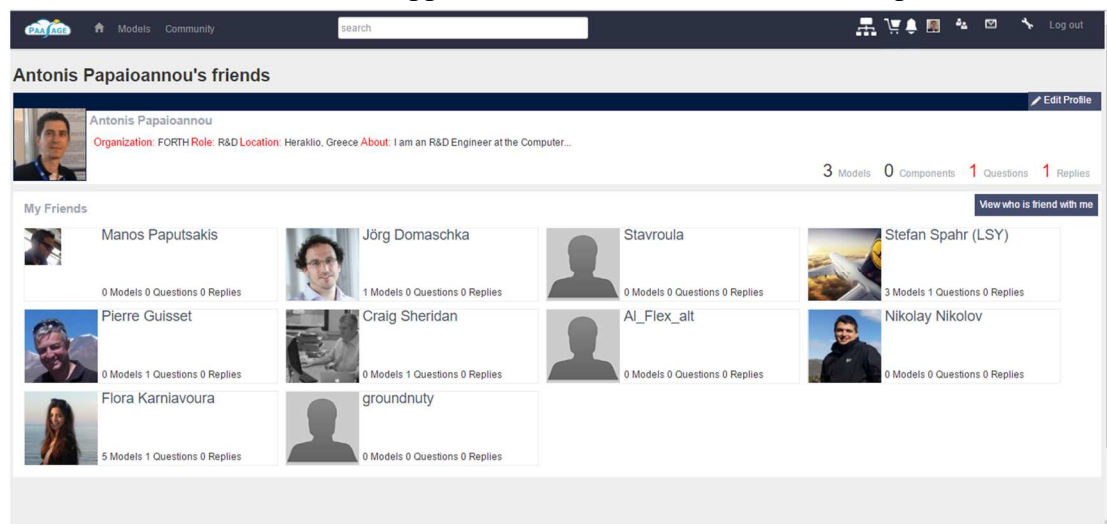


Figure 25: User's Friends page

Finally, below the Connections area groups that the user is member of are depicted. Groups are part of the Community page of the social network which is presented in the next session.

5.4 Social Network Community

The SN Community delivers to the user the functionality provided below:

- Create, Join or Leave a group
- Ask a question or begin a topic
- Reply to a question or a topic and
- Associate models and application executions to a question or a reply.

The home page of the community page is shown in Figure 26. In this page the user can join the suggested groups, navigate through the discussion feed or find new friends in the suggested connections section.

The outlook of a group of which the user is already a member is shown in Figure 27. The member can do the followings actions:

- Ask questions,
- Reply to questions
- See the group members
- Invite her friends to join the group and
- See other popular groups

When a user is member of a group can also see the group activity where information about which member of the groups started which discussion topic is available. Furthermore, a list with all the discussion topics of the group is depicted next to group activity. When a new discussion topic is started or when a group member shares an answer a notification is sent to all group members. Notifications can be reached using the *Notification* shortcut in the top-bar menu.

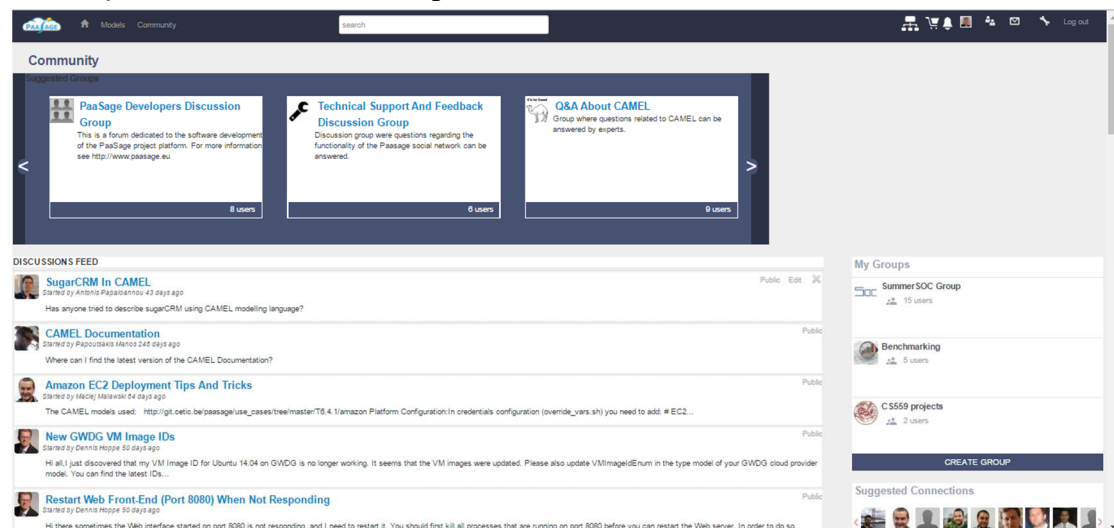


Figure 26: Community Home Page

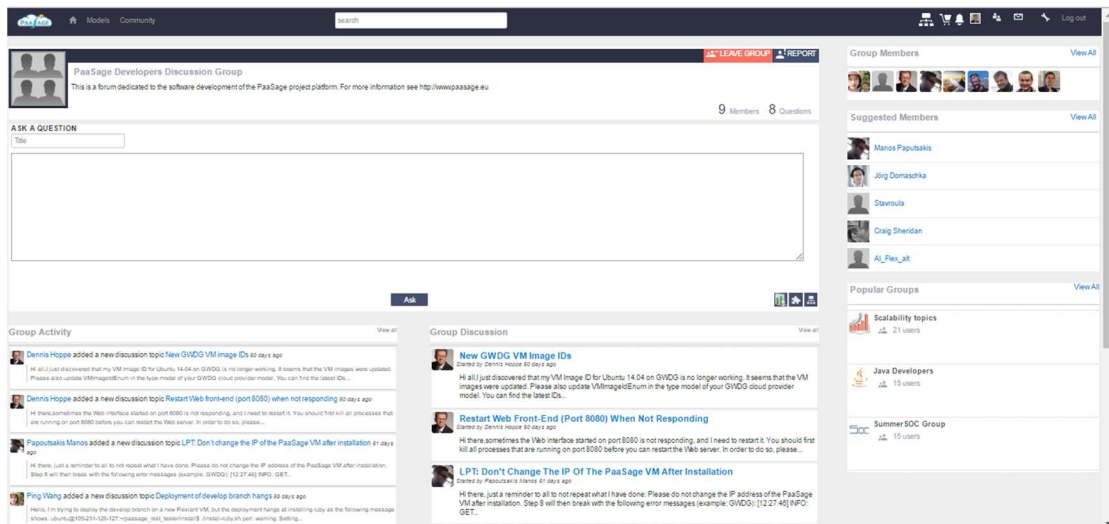


Figure 27: A group's view

5.5 Models

All the applications live in the *Models* section of the SN. The home page of the models is shown in Figure 28. The user can perform the following actions on models he / she has access too:

- Search for a specific model
- Apply filters such as the deployment cloud, the geography and the tags of a model
- Visit the application model page of a model and have access to more details about it (full model view)

In full model view the user can:

- Navigate through the past executions in *Runs* tab
- Leave a comment about the model in the Discussions tab
- See or add a review in *Review* submenu () and
- Find similar models based on their tags

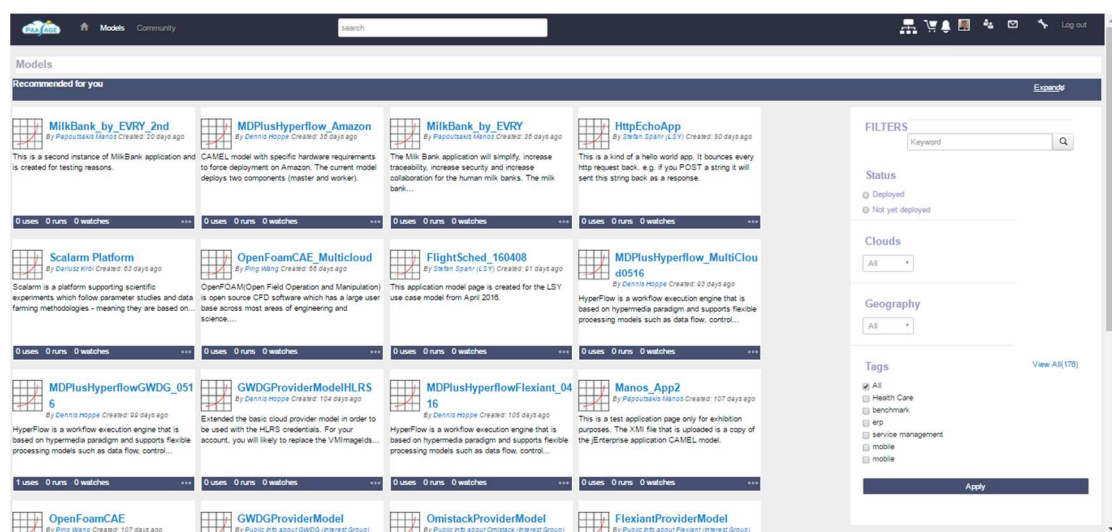


Figure 28: The home page of models

As mentioned above, the user can see the past executions of an application model (Figure 29). The user can filter these executions based on their cost, date, uptime, geography, cloud provider and up to 2 of their metrics. Moreover, the executions can be sorted by any of their metrics. A right click on any execution reveals its configuration.

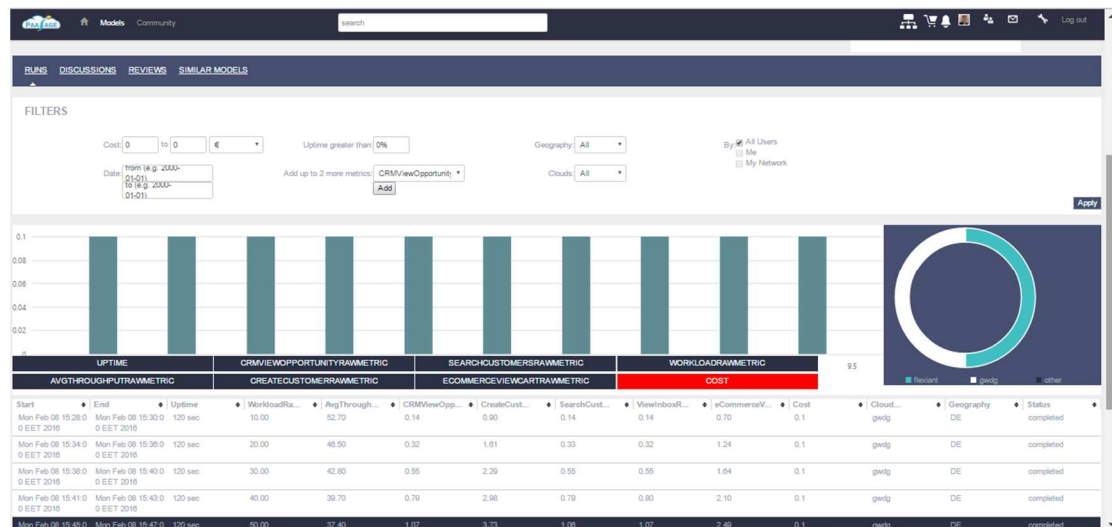


Figure 29: Models: Past Executions Page Section

Figure 30 shows the *Similar Models* tab. Under *Similar models* tab we can see other models that are shared on the platform and are considered similar to sugarCRM based to their tags. The higher a model is depicted, the more same tags it shares with the *SugarCRM* model. In Figure 30 below, the first similar model shares 4 tags with *the model in question*, the second shares 2, while the two last ones share only 1.

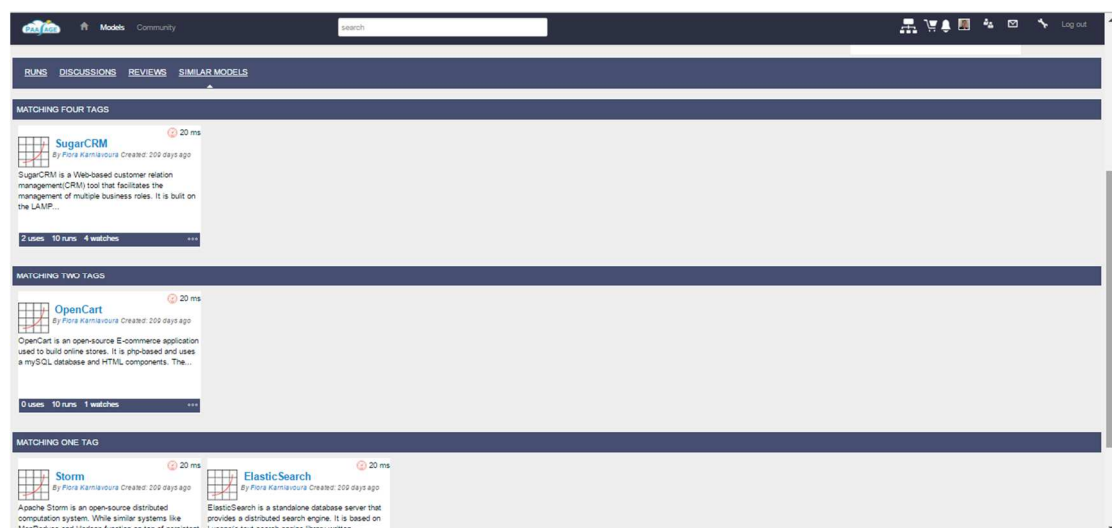
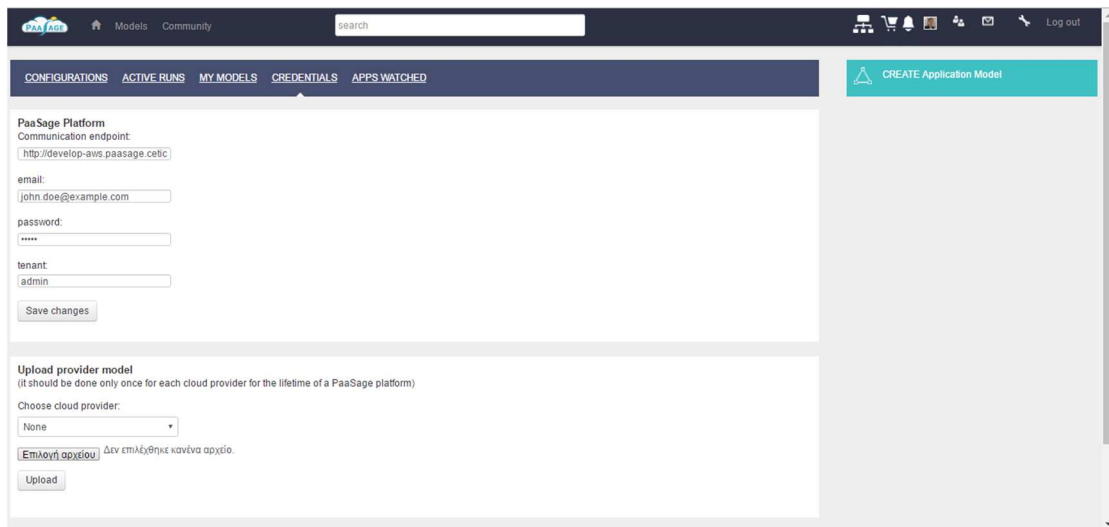


Figure 30: Similar Models tab

5.6 Deployment of an application

The social network can be used as a UI for the PaaSage platform and for the deployment of an application. The first step is the user to fill in their *PaaSage* platform credentials. These credentials will allow the social network to connect with the PaaSage platform using a RESTfull API. The *My Area* shortcut redirects the user to the *My Area* page. Under the *credentials* tab the platform credentials that must be filled in are an endpoint, an email, a password and a tenant.

The next step is to upload the provider model to the *PaaSage* platform. Right bellow the PaaSage platform credentials the user can browse for the provider model and upload it. Figure 31 shows the outlook of the *credentials* tab.



The screenshot shows the 'CREDENTIALS' tab of the PaaSage Platform. The interface includes a top navigation bar with links for 'CONFIGURATIONS', 'ACTIVE RUNS', 'MY MODELS', 'CREDENTIALS', and 'APPS WATCHED'. A search bar is located in the top right corner. The main content area is divided into two sections. The first section, titled 'PaaSage Platform', contains fields for 'Communication endpoint' (with a value of 'http://develop-aws.paasage.cetic'), 'email' (with a value of 'john.doe@example.com'), 'password' (masked with asterisks), and 'tenant' (with a value of 'admin'). A 'Save changes' button is located below these fields. The second section, titled 'Upload provider model', includes a note '(it should be done only once for each cloud provider for the lifetime of a PaaSage platform)'. It features a 'Choose cloud provider' dropdown menu currently set to 'None', a button labeled 'Επιλογή αρχείου' (File selection) with a tooltip 'Δεν επιλέχθηκε κανένα αρχείο.' (No file selected), and an 'Upload' button.

Figure 31: Credentials tab

The last thing for the application deployment is the model of the application itself. The deployment actually consists of two phases, the reasoning and the actual deployment. In the model page of an application there is a button for each phase as we can see in Figure 32 bellow.

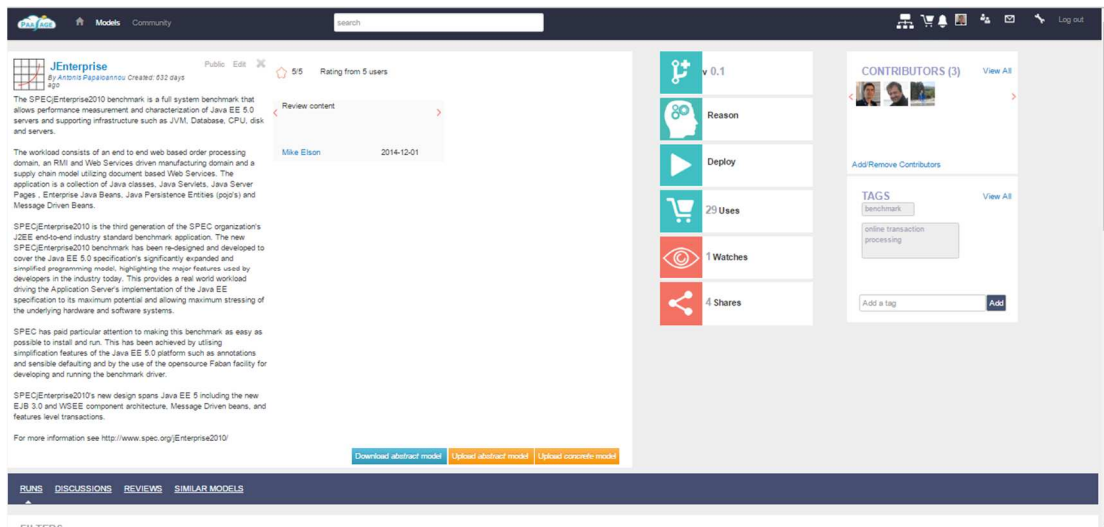


Figure 32: Reason and Deploy button in the application model page

First the user clicks on the *Reason* button. The reasoning lasts up to a minute. Finally, the *Deploy* button must be clicked. The deployment may take several minutes due to many time-consuming processes that take place. At any time the user can be informed about the deployment of their application under the *Active runs* tab. Every application model that is shared by the user and is deployed is shown here. The page is refreshed every 5 minutes and there is also a *Refresh* button. The name of each application, its state and substate are depicted. All these are shown in Figure 33 below.

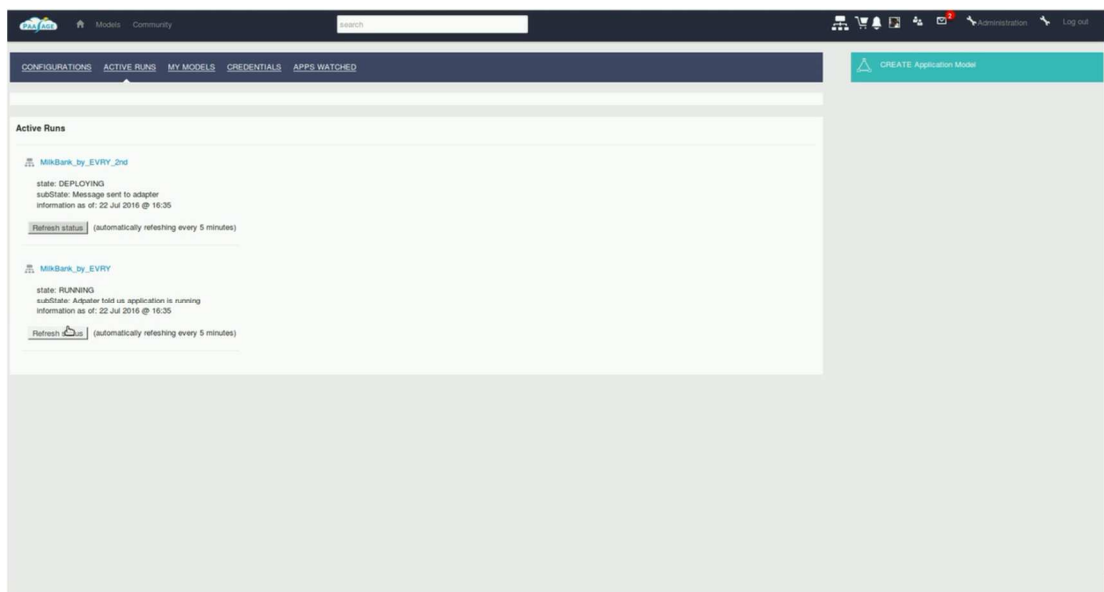


Figure 33: Active runs tab

5.7 Summary

In this section, several aspects of the PaaSage social network have been discussed, such as the specification of user profiles, creation of a discussion group enhancing collaboration among users, management and search of CAMEL models, exploitation of knowledge produced of the execution history of the same or similar applications and deployment of an application using the PaaSage platform. The presence of experts in the social network makes it an important alternative to any documentation or tutorial that a user would need for using the PaaSage platform.

6 Training Workshops

6.1 Design and implementation

In this section the conduct of a training workshop is described providing a step by step tutorial. The instructions of this tutorial were used for the conduct of a 1 hour training session about PaaSage project during the 10th Symposium and Summer School on Service-Oriented Computing, which took place on June27 – July 1, 2016 in Crete, Greece. However, they can also be used for an extended training workshop, which could last half or even a whole day, with some additions to the existing agenda.

These headlines can be referred to the main parts of the training workshop tutorial: i) general overview of the PaaSage project; ii) success stories; iii) introduction to CAMEL; iv) application deployment using the ruby client of the PaaSage platform; v) introduction of the PaaSage social network and integration with the PaaSage platform; and vi) questionnaires.

6.1.1 General overview of the PaaSage project

For the entry part of the tutorial a PowerPoint presentation has been created named *paasageInNutshell.pptx*, which includes 11 slides. The presenter talks about what PaaSage is and the need for developing once and deploying on many clouds that it serves. After that, the new opportunities are mentioned, such as the cross-cloud application provisioning, the ability to select the best cloud service based on user requirements and the avoidance of vendor lock-in. Features of PaaSage such as multi/cross cloud application, deployment optimization, modeling of application aspects, e.t.c and the architecture of the PaaSage platform are next in the line. The presentation also includes the scope of the tutorial, it focuses on usage of respective tools and entry points of the PaaSage platform and on how to use the platform to achieve multi-cloud deployments as well as share knowledge.

Finally, the presenter asks the participants to provide their feedback in the end of the workshop by filling in a questionnaire. The duration of this part could range between 10 and 20 minutes depending on the level of detail. The skeleton of the presentation is shown below:

- a. what PaaSage is
- b. need served by the PaaSage platform
- c. opportunities PaaSage offers
- d. architecture of PaaSage platform
- e. features of PaaSage
- f. scope and outline of the tutorial
- g. feedback from the participants

6.1.2 Success stories

This part is devoted to successful usage of the PaaSage platform in real-life scenarios. This is the case of use-case partners of PaaSage project, which are going to produce a success story video for their applications. For the time being and while this document is being created, 2 of these videos are available, ASC-S use-case video and Lufthansa Systems use-case video. The duration of each video is approximately 3:30 minutes, so

if all of them are going to be watched the duration of this part as a whole would be around 20 minutes.

6.1.3 Introduction to CAMEL

During this section the audience has the opportunity to watch and practice on how to express an application using CAMEL. However, the participants have to become familiar with CAMEL first; the PowerPoint presentation named *CAMEL.pptx* of 27 slides serves this purpose. As a start, the presenter refer to the features of this modeling language such as the re-usage of existing languages, the fact that it is an integration of other DSLs, the existence of syntactic and semantic model validation, e.t.c. After that, how CAMEL is involved in the PaaSage workflow and how the integration of the language was achieved is explained. Moreover, respective tools are mentioned; the three editors: i) tree, ii) textual and iii) graphical editor. The presenter explains which the capabilities of each editor are and what kind of sub-models they can create.

The last and maybe most important part is the *tool training*. The focus is on the textual editor, the purpose is to create a CAMEL model for an application and all this is done through exercises on how to use the editor. The presenter asks the audience to use an editor of their choice in order to complete some tasks on application modeling. They can of course use the CAMEL textual editor itself. A basic model, which can be downloaded by the participants, can be used as a starting point for the exercises. The participants watch the presenter to add requirements to the application model, edit existing requirements, add/remove application metrics, create requirement sets, e.t.c. They are asked to perform similar tasks. The difficulty of the tasks is incremental. In the end of this part the audience has seen how most of the relevant aspects are covered by CAMEL. The duration of this section is about 60 minutes but it can range depending on the tasks the participants are asked to perform and their difficulty. The skeleton of the CAMEL presentation is shown below:

- a. CAMEL features
- b. CAMEL in PaaSage workflow
- c. CAMEL integration
- d. tool support
- e. sustainability (documentation, deliverables, code, webpage)
- f. tool training
 - ✓ focus on textual editor
 - ✓ CAMEL model for SugarCRM application
 - ✓ exercises on how to use the editor

6.1.4 Application deployment using the ruby client of PaaSage platform

What follows is a demonstration of the PaaSage platform. A Ruby client is used in order the presenter to communicate with the PaaSage platform, upload at least two models (cloud provider model and application model), start the reasoning phase and start the deployment phase. The PaaSage platform should be already installed somewhere and be accessible. Since the deployment of an application takes a lot of time, this section of the training workshop could start before the coffee break and end after it. In such a case the deployment can be done in real time and the results can be

seen after the break. The only thing the presenter will do after the break is to connect to the VM the application is deployed and show that it works as expected. A good strategy would be the model that is used for this deployment to be the same with the model that was constructed during the previous section.

The focus is on the familiarization of the participants with the PaaSage platform. They do not have to follow the steps of the presenter; just watch the feedback of the platform. The duration of this section could be up to 60 minutes, including a 45 minutes break. The skeleton of this section is shown below:

- a. create resource for cloud provider model
- b. upload cloud provider model
- c. check if cloud provider model is uploaded
- d. create resource for application model
- e. upload application model
- f. check if application model is uploaded
- g. start reasoning
- h. check if reasoning is finished
- i. start deployment

6.1.5 Introduction of PaaSage social network and integration with PaaSage platform

During this part of the training workflow the participants become familiar with the PaaSage social network and learn how to use it as an entry point to the PaaSage platform. They are prompted to create profiles in the social network and follow the steps of the presenters according to specific scenarios.

The first task is to visit the website and login. Since the majority of them have not created a profile in the past, they are asked to create one by filling in the sign up form or by just making use of their Facebook/Twitter account. What follows is a small description, by the presenter, of the social network entry page and then they are prompted to browse for a specific application model page. In order to do that they have to visit the *Models* page and scroll down until they discover the name of the application in question or use the search input of the page. They do not have to type the whole name of the application, just a part of it. In any case, they click on the name of the application and visit the application model page.

Once again, the presenter describes what can be found in that page emphasizing the *Tags* feature. The focus is on the tabs *Runs* and *Similar models*. Under *Runs* tab the user can find the visualization of executions. There are charts for each application specific metric, a table where each row is an actual execution and a filter area that can be used in order the user to focus on those executions that they are interested in. As an exercise the presenter asks the participants to filter the executions of the application asking only those that the value of a specific metric ranges between two predefined values. Under *Similar models* tab the user can find other models that are

shared on the platform and are considered similar to the application in question based to their tags. The higher a model is depicted, the greater the similarity.

The next task is to visit the profile page of the contributor of the application. The participants are asked to it by clicking the name of the contributor, under the name of the application. What they can do here is to add the contributor to the users they follow and discover other models and discussion topics that are shared on the social network by the contributor. Finally, they can have a look to the whole activity of the contributor in the social network.

A strong feature of the PaaSage social network is its community. The participants are prompted to visit the *Community* page and browse for a specific group. This group can be part of the suggested groups but in any case they can use the search input of the page. After they visit the page of the group they are asked to become members using the *Join group* button. The presenter, which is also member of this group, starts a new discussion topic by asking a question and the participants should answer to that question taking part to the discussion. An extra exercise is to associate an application model or an application execution with a specific question.

The last part of this introduction to the social network is the presenter to show how the social network can play the role of a UI for the PaaSage platform and use it to deploy an application. Three different steps should be followed for a successful deployment, the user must add the platform credentials, the provider model must be uploaded and the user must make use of the *Reason* and *Deploy* buttons. For the first step, a form must be filled in at *My Area* page, under the *Credentials* tab. Right below this form, there is the area where the user can upload any provider model that is needed for the deployment. In order the last step to be fulfilled the user must visit the application model page, click first the *Reason* button and, after the reasoning is done, click the *deploy* button. Since the deployment of an application is already demonstrated during the training workshop, there is no need for the presenter to actually start a second deployment process. If we also consider that deployment is a time consuming process, the presenter can mention the existence of the *Deploy* button but not actually click on it. The participants should not follow the steps of the presenter, just watch the whole process.

The duration of this section is about 60 minutes and it depends on how quickly the participants can follow the presenter's instructions and what kind of difficulties they will face. A Word document named *SocialNetworkLiveDemo.docx* is actually a step by step tutorial presenting the main functionalities of the social network and can be used for the conduct of this introduction to social network section. The skeleton of this section is shown below:

- a. login to the social network
- b. browse for specific application model
- c. browse information of the chosen model
 1. Runs tab
 2. Similar models tab
- d. visit profile page of the contributor
- e. visit community page

- f. deployment of the application
 - 1. add platform credentials
 - 2. upload provider model
 - 3. reason and deployment buttons

6.1.6 Questionnaire

In the end the participants are asked to complete a questionnaire in order to rate and comment their experience. The duration of this section is about 10 minutes and it depends on the size of the questionnaire. In any case, what has to become clear to the participants is that this is an essential part of the training workflow. A good strategy is the questionnaires to be delivered as a hard copy to the participants in order their completion not to be postponed.

6.1.7 Additional sections

If there is enough time more sections could be added in the training workshop such as:

- a. Discussion about the different steps of the PaaSage workflow. What happens during reasoning, which components participate, e.t.c.
- b. How participants can contribute to PaaSage and use it in the future

7 Industrial workshops

7.1 Product launch strategy

As the PaaSage platform gained in stability and functionalities, a launching strategy for potential end users has been developed and implemented in years 3 and 4 of the project lifetime.

On basis of discussions within the Project Executive Board members, it has been decided to organise several ‘product launch events’, under the form of industrial workshops distributed over several countries, instead of organising a single workshop.

The PaaSage product launch strategy is thus based on following action plan:

- Select a professional open source repository (OW2) to host the PaaSage source code and support the developer’s community.
- Revamp the PaaSage web site, and convert it from a ‘project-based’ web site towards a ‘product-based’ web site. This process and the results are extensively documented in deliverable D9.1.2 (“Website product”).
- Organise industry focused events, under the form of ‘Industrial Workshops’, following the plan exposed in D9.4.2 (“Industrial workshops Planning”).
- Publish a professionally-looking leaflet presenting PaaSage as a product.

7.2 Structure of the industrial workshops

In order to maximize the impact of the industrial events, these have been organised by PaaSage industrial partners and targeted to that specific industrial partner ecosystem in its region.

Two kinds of industrial events have been organised:

1. Joining a related event, and having a specific session / booth for presenting PaaSage:
 - a. In Cyprus, coordinated by IBSCY, with the support of UCY (March 2015).
 - b. In UK, coordinated by STFC, with the support of Flexiant (March 2015)
 - c. In Belgium, coordinated by CETIC (2 events in November 2015)
 - d. In Norway, coordinated by EVRY, with the support of SINTEF and UiO (February 2016)
 - e. In France, coordinated by INRIA⁶ (at the OpenStock Workshop Lyon 2016, June 2016)
2. PaaSage-specific events, synchronized with a PaaSage consortium meeting:
 - a. In Belgium, coordinated by BE.WAN (September 2015)
 - b. In Germany, coordinated by ASC(S (April 2016)
 - c. In Norway, coordinated by EVRY (September 2016)
 - d. In Hungary, coordinated by LSY (September 2016)

For the second kind of events (that are PaaSage-specific), the event organisation as well as the related communication campaign are in the hands of a PaaSage industrial partner. The workshop duration is typically ½ day, in the afternoon. As the industrial workshops are co-located together with a PaaSage consortium meeting, PaaSage

⁶ The scope of the French industrial workshop has been reduced due to the withdraw of the French industrial partner (SYSFERA).

partners are able to attend and contribute to the workshop presentations without additional cost, and also to gain experience for subsequent workshop organisation.

The typical agenda of Industrial Workshops is as follows:

- Welcome and introduction
- Presentation of PaaSage & CAMEL (objectives, architecture, main features...)
- Use case presentation (focus on the industrial partner use case)
- Demo or open/interactive booth
- Panel discussion, Q/A
- Cocktail and networking.

Of course, each organiser has the possibility to tune the standard agenda to its specific context and needs.

7.2.1 ASC(S Industrial Workshop (Stuttgart, Germany)

Invitation and venue

11.04.2016
13:00
University of Stuttgart
Allmandring 30

Participation is free of charge.
Sponsored by ALTAIR
In collaboration with PAAUSAGE

Expired

simpulseyday numerics and digitalization

From Cloud to Road

The very fast growing global competitive pressure, new legal regulations and distributed development environments during the last few years lead to a complete rethink of Computer-Aided-Engineering (CAE) and High-Performance-Computing (HPC) processes. It is now a question of going further to flexible and easy to access / operate CAE work environments. Therefore Cloud-Computing and High-Performance-Computing could be an opportunity for the overall European automotive, aerospace and engineering industry, especially for innovative small and medium enterprises – SMEs. A future perspective and state-of-the art overview of CAE simulation tools and environments that are compatible with the use of Cloud-Computing

and HPC will be presented and discussed.

> Show Details

The event has been organised on 11 April 2016 by the Automotive Simulation Center Stuttgart e.V. in close cooperation with the large scale European project PaaSage.

See <https://www.asc-s.de/en/workshops>.

Target audience

Tier-2 and Tier-3 suppliers, and especially SMEs in automotive industry, as well as automotive OEM's.

Workshop programme

13:00	WARM-UP Arrival of the guests and welcoming them over a cup of coffee
-------	--

13:30	INTRODUCTION ROUND Introduction to the workshop
13:40	TAKE THE POLE POSITION - PART 1 KEYNOTE: Driving Vehicle Development from Road to Rig to Simulation Jürgen Kohler – Daimler AG
	Cognitive Computing meets Computer Aided Engineering Dr. Stefan Suwelack - Karlsruhe Institute of Technology (KIT), Institute for Anthropomatics Humanoids and Intelligence Systems Lab
	PaaSage: Model-based Cloud Platform Upperware Prof. Dr. Keith Jeffery, Dr. Geir Horn, Dr. Alessandro Rossini, Prof. Dr. Kostas Magoutis, Carlos Falconi PaaSage Project Consortium (EU FP7)
14:50	PIT STOP Coffee break and PaaSage live demo
15:10	TAKE THE POLE POSITION - PART 2 Simulation Driven Innovation in the Cloud – A Game Changer? Dr. Detlef Schneider - Altair, Inc.
	Usage of GPU LS-DYNA in the Cloud Prof. Dr. Ulrich Göhner - DYNAmore GmbH
	COMSOL Server: Simulation without frontiers Dr. Bernhard Fluche - COMSOL Multiphysics GmbH
16:10	PIT STOP Coffee break and PaaSage live demo
16:30	TAKE THE POLE POSITION - PART 3 The NUBERISIM platform - A new way to predict fluid flow noise using cloud-based Computational Aero-Acoustics Carlos Falquez, Dr. Iris Pantle, Dr. Balazs Pritz NUBERISIM, Falquez, Pantle und Pritz GbR
	Overview of Cloud Systems and Services for CAE Christopher Woll and Jan Martini - GNS Systems GmbH
	CAEaaS: on-demand simulation out-of-the-cloud Markus Westhäufer – T-Systems International GmbH
17:30	CHEQUERED FLAG Get-together – networking and small snack and PaaSage live demo

Workshop outcome

The half day event was very well visited by an international audience of about 50 participants including key representatives from science and automotive industry. The agenda was composed of several interesting speeches which showed the state-of-the-art in cloud technology and gave the participants a hint of the road of future

technologies. The lectures presented encouraged an intense exchange among the European experts in the field of cloud computing.



During the breaks the participants exchanged their insights and the PaaSAGE platform was demonstrated to the audience showcasing the best qualities of the platform as the Multicloud deployment. The professional exchange was supported through a get-together at the end of the event, where plenty of opportunities were afforded to build new business networks around the PaaSAGE technology.

7.2.2 EVERY Industrial Workshop (Oslo, Norway)

Invitation and venue

EVERY has organised an industrial workshop in its premises in Oslo. See below the invitation leaflet.



Develop once—deploy to the full spectrum of the Cloud!



Cloud platforms are not homogeneous. API and architectures are not standardised. Applications are often dependent on the specific Cloud platform. The barriers between the clouds limits the big potential on cross-cloud computing and can cause vendor lock-in.

PaaSAGE is an open source platform to support design and deployment of Cloud applications. Model-based development, configuration, optimisation and deployment of applications independently of the existing underlying Cloud infrastructures. Supports cross-cloud deployment and execution.

The PaaSAGE platform has been designed and developed by a consortium of 19 European members. EVERY, SINTEF and UiO are Norwegian partners. Project has research funding by the European Union 7th framework programme.

The seminar will address the dark side of the cloud and show case the PaaSAGE technology.

The seminar is free and open for everyone.

Sign up by contacting frode@evry.com or register on: <http://bit.ly/2ccn7IY>

Date: 20th September
Location: EVERY Academy,
Snarøyveien 30A
Fornebu 1360



Key note speaker:
Peter Hidas
Gartner



Geir Horn
University of Oslo



Alessandro Rossini
SINTEF



Jörg Domaschka
University of Ulm



Frode Finnes Larsen
EVERY









Target audience

The target audience consists of EVRY partners and customers.

Workshop programme



AGENDA		
09:00	Registration and morning coffee	
09:30	Welcome and introduction of the agenda	Frode Finnes Larsen
09:40	Key Note: The dark side of the cloud: Vendor lock-in	Peter Hidas
10:15	EVRY's Milkbank use case and other PaaSage case studies	Frode Finnes Larsen
10:30	Coffee break	
10:45	Develop once — deploy to the full spectrum of the Cloud <ul style="list-style-type: none">• The nightmare of DevOps• Introduction to PaaSage• Layers of PaaSage	Geir Horn
11:00	Cloud application modelling with CAMEL <ul style="list-style-type: none">• Model-driven engineering and domain-specific languages• CAMEL: Cloud Application Modelling and Execution Language• Provisioning and deployment models• Optimisation requirements	Alessandro Rossini
11:30	Autonomic cloud deployment <ul style="list-style-type: none">• Understanding configurations• Application utility• Role of adaptation	Geir Horn
12:00	Cross-Cloud application execution <ul style="list-style-type: none">• Cloudiator: A multi-tenant, cross-cloud orchestration framework• Metric monitoring, collection, and use• Platform-specific scaling	Jörg Domaschka
12:30	FREE LUNCH	
13:30	Hello world! Demonstration of a minimalistic CAMEL model	Jörg Domaschka and Alessandro Rossini
14:00	Conclusions and outlook	Frode Finnes Larsen
Extra session:		
14:15	Do it yourself: Hands on workshop on specifying a CAMEL model	Alessandro Rossini
15:30	Close	

Workshop outcome

The workshop was attended by 25 partners and customers of EVRY. A lot of interest was expressed around the PaaSage platform and the CAMEL language. Specifically, the Gartner representative (Peter Hidas), invited as keynote speaker, was extremely impressed by the strength of the PaaSage platform.




7.2.3 LSY Industrial Workshop (Budapest, Hungary)

Invitation and venue

The LSY industrial workshop has been organised in Budapest on 27 September 2016, under the title “Future of Clouds Conference”.

Future of clouds conference



LEARN ABOUT WHATS NEXT IN CLOUDS!
27. SEPTEMBER 2016.
13:00 ELTE CONGRESS CENTER 1117, PÁZMÁNY PÉTER STNY. 1A HUNGARY

Join this workshop to learn how the PaaSage platform may benefit to your industry. The event is organised by Lufthansa Systems and will feature presentations and demonstrations by PaaSage and other EC-funded research projects.

[More information](#)

The conference was followed by an “Aviation IT Technology Fair” where LSY presented several internal projects as well as its contribution to 3 EC-funded projects (PaaSage, Beacon and Musa).

Target audience

LSY employees, external community from other IT companies, universities, members of EU funded projects.

Workshop programme

WELCOME & KEYNOTE I.

Dr. Dirk Muthig,

Head of CTO & Innovations, Lufthansa Systems

Cloud is about how computing is done and not about where. By coupling research and innovation, the EU projects make it easier for the public and private sectors working together in delivering innovation and securing Europe's global competitiveness. The leading universities, research companies and IT companies of Europe and Lufthansa Systems work together in order to help new developments to be designed for Cloud usage.

KEYNOTE II. CLOUDS: DRAWBACKS AND VIRTUES

Ferenc Frész, Chief Executive Officer of Cyber Services PLC

What are the benefits of developing and deploying applications on multiple Cloud infrastructures? Which questions need to be answered regarding security issues?

MULTI-CLOUDS: POSSIBILITIES, AND SECURITY ISSUES - PODIUM DISCUSSION

Prof. Dr. Keith G. Jeffery, European Research Consortium for Informatics and Mathematics;

Philippe Massonet, Scientific Coordinator, CETIC - ICT Applied Research Center,
Stefan Spahr, Software Architect, Lufthansa Systems (Lufthansa Systems representative of the MUSA project)

How can we create an open and integrated platform to support both the design and deployment of Cloud applications, hiding the complexity of working with Clouds from developers? What are the most important security solutions within Clouds? Is Multi-Cloud the ultimate answer?

WHY MODEL-BASED DEVELOPMENT IS NEEDED

Dr. Alessandro Rossini, Research Scientist, SINTEF

Stefan Spahr, Software Architect, Lufthansa Systems

What is the practical usefulness of a modelling language that can be executed and thus used as a way of controlling applications automated deployment in Clouds for Aviation IT?

OPTIMIZATION OF CLOUD APPLICATIONS

Dr. Geir Horn, Head of European ICT Projects, University of Oslo

Can "elasticity" be the key feature of Clouds? How can the pay per use model minimize the infrastructure costs and increase the performance when needed?

NEW LEVELS OF CLOUD SECURITY - PODIUM DISCUSSION

Dr. Geir Horn, Head of European ICT Projects, University of Oslo

Philippe Massonet, Scientific Coordinator, CETIC - ICT Applied Research Center,

Ferenc Frész, Chief Executive Officer of Cyber Services PLC

Security is always one of the major issues when looking at the utilization of Cloud infrastructures. Therefore these concerns of cloud service consumers' needs to be addressed; even though this is an exhaustive challenge, in most cases, it is achievable.

WHAT IS THE VIEW OF THE HUNGARIAN CLOUD EXPERTS? - OPEN DISCUSSION

The developed tools of the Horizon 2020 and FP7 Programs are all published in Open Source. What is the view of the Hungarian Cloud experts on Open Source in the context of its commercial application?

Workshop outcome

The workshop was very well attended, with close to 200 participants. Below are some pictures of the event.



7.3 New PaaSage leaflet

In order to support the launch of the PaaSage platform, a new leaflet has been designed and created, focusing on key features, benefits and success stories. It is available on the web site as a PDF document and has been distributed at the latest PaaSage industrial workshops.

The following screenshots illustrate the new PaaSage leaflet:

PaaSage key features

- Modelling of the application components and its requirements
- Based on the CAMEL language
- Automated cross-Cloud deployment
- Social network allowing knowledge and model sharing and re-use
- Deployment optimisation based on requirements, scalability rules and knowledge collected from social network
- Auto-scaling, triggered by runtime scalability rules
- Dynamic reallocation of application components to different Clouds (Cloud bursting)
- Distinct layer from the infrastructure
- Open source

About PaaSage

PaaSage is an open source integrated platform to support both design and deployment of Cloud applications, together with an accompanying methodology that allows model-based development, configuration, optimisation and deployment of existing and new applications independently of the existing underlying Cloud infrastructures.

PaaSage is available through the OW2 open source repository and community.

The origin of PaaSage

The PaaSage platform has been designed and developed by a consortium of 19 members under the leadership of ERCIM.

The PaaSage consortium is a combination of scientific research partners, technology transfer centres, Cloud technology providers and application developers.

The PaaSage consortium received research funding from the European Union's 7th Framework Programme under contract FP7-ICT-317715 (active from 2012 until 2016).

Contact

Mail: info@paasage.eu
Web: www.paasage.eu
Code: [opensource.paasage.eu](https://github.com/paasage)



A Model-based Cross-Cloud development and deployment platform



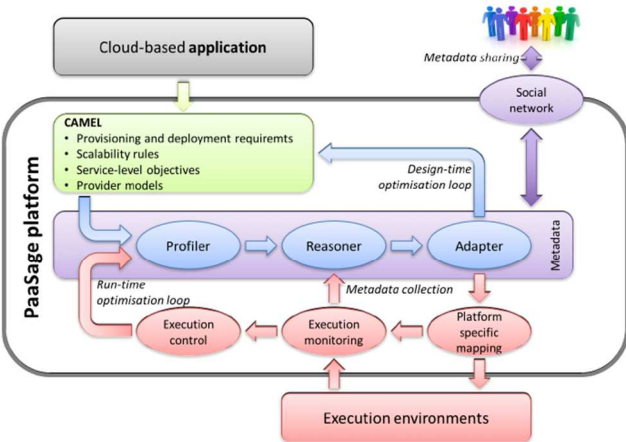
Define your application **once**

Deploy it at the **full spectrum** of the Clouds

www.paasage.eu

© Copyright ERCIM GEIE 2016

Figure 34: PaaSage leaflet 1



A solution for easier Cloud application development

The pain

Cloud platforms are not homogeneous. API and architectures are not standardized. Applications are often dependent on the specific Cloud platform. Generic tools for supporting application deployment and runtime monitoring are lacking.

The solution

PaaSage delivers an application deployment platform that solves the vendor lock-in paradox and increases productivity and user satisfaction.

The PaaSage environment is open source and is extensible and modifiable. It uses the CAMEL language to model the application down to its components.

PaaSage allows for deployment on multi-Cloud platforms (of any kind).

In producing an optimal solution, PaaSage takes into account the characteristics of the available Cloud platforms, the data set to be used and the end-user preferences (such as location, availability, price...).

Success stories

PaaSage for ERP

be.wan (Belgium) takes advantage of the PaaSage methodology and platform, as well as the CAMEL language, to increase the efficiency of its development team.

"With PaaSage, we can now benefit from the real capabilities of the Cloud without spending a lot of time on deployment, monitoring and scaling scenarios and procedures."
Franky V., be.wan

PaaSage for flight scheduling

Lufthansa Systems GmbH & Co. KG (Germany) is developing its next generation flight scheduling solution that is used every day by more than 60 airline companies around the globe.

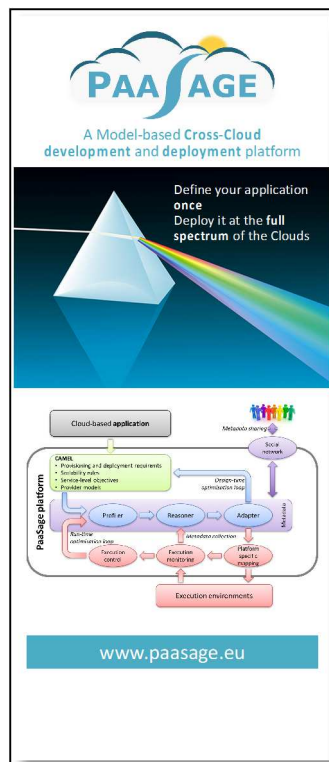
"Cloud provider dependency is a major obstacle for application providers with worldwide customers (...) PaaSage gives the freedom to model the application independently of the Cloud providers, which are selected afterwards by the PaaSage platform according to the requirements."
Stefan S., Lufthansa Systems

For further learning about PaaSage functions and benefits, more demonstrators and training materials are available at www.paasage.eu.

Figure 35: PaaSage leaflet 2

7.4 PaaSage roll-up

For supporting PaaSage presence at various events (industry fairs, conferences...), a roll-up has been designed (see following picture).



8 Conclusion

This deliverable has provided training material and documentation which can be used to guide a practical understanding of the PaaSage platform. The material is geared towards different types of users so as to streamline a variety of tasks, such as the deployment and configuration of the PaaSage platform, specific management actions concerning its Executionware, the modeling of end-user requirements in CAMEL and the sharing and exploitation of knowledge incarnated in PaaSage's Social Network.

By leveraging training material on how to configure, build and deploy the PaaSage platform (Section 2), the level of difficulty of using the PaaSage platform is significantly lowered, since the goal was to make it as easy as possible for a newcomer to get it running. This is illustrated with the description of the Executionware components in this document.

Outside of the core technical components the CAMEL standard developed on this project expresses user requirements through the Cloud lifecycle. CAMEL is vital to both the use and understanding of the platform project and is used to capture the needs of application developers/owners, focusing mainly on deployment, non-functional requirements and scalability rules. The approach to training and documentation around CAMEL has focused on usability and the provision of user friendly textual and web based editors supported by the Social Network.

An ideal starting point to both understanding the PaaSage platform's main component functions and use of CAMEL is the Social network. Training material on the PaaSage social Network has been a core focus of the project. Using the Social Network training providers can walk users through the specification of user profiles, creation of a discussion groups to enhance collaboration among users, management supporting the discovery and sharing of CAMEL models. This collaboration is further enhanced by the shared exploitation of knowledge produced from the execution history of the same or similar applications deployed on the PaaSage platform.

This technical material formed the base of the delivery of training workshops in the project. For example the PaaSage training workshop organized in the main track of the 10th Symposium and Summer School on Service-Oriented Computing (SummerSOC) in July 2016, used several of the training materials described in this deliverable.

Through the life of this project we have delivered over a dozen workshops using the wide resource of training material created during the life of PaaSage. The future of PaaSage depends on the provision and access to this material; this is done via the PaaSage website and code via the OW2 presence of the project. Videos and other training aids have been made available on public sites such as YouTube. Thus providing a legacy for PaaSage and a spring board by which further development and customization beyond the project can take place.

9 Bibliography

[CloudML] Nicolas Ferry, Franck Chauvel, Alessandro Rossini, Brice Morin and Arnor Solberg. “Managing multi-cloud systems with CloudMF”. In: *NordiCloud 2013: 2nd Nordic Symposium on Cloud Computing and Internet Technologies*. Ed. by Arnor Solberg, Muhammad Ali Babar, Marlon Dumas and Carlos E. Cuesta. ACM, 2013, pp. 38–45. ISBN: 978-1-4503-2307-9. DOI: 10.1145/2513534.2513542.

[D1.6.1] The PaaSage Consortium. D1.6.1—Initial Architecture Design. PaaSage project deliverable. Oct. 2013. Available at:

http://www.paasage.eu/images/documents/paasage_d161_final.pdf

[D2.1.2] Alessandro Rossini, Nikolay Nikolov, Daniel Romero, Jörg Domaschka, Kyriakos Kritikos, Tom Kirkham, Arnor Solberg, CloudML Implementation Documentation. PaaSage Deliverable D2.1.2, March 2014. Available at:

http://www.paasage.eu/images/documents/paasage_d2.1.2_final.pdf

[D3.1.1] Amin Bsila, Nicolas Ferry, Kamil Figiela, Geir Horn, Tom Kirkham, Maciej Malawski, Nikos Parlavantzas, Christian Perez, Jonathan Rouzaud-Cornabas, Daniel Romero, Alessandro Rossini, Arnor Solberg, Hui Song, Upperware Prototype. PaaSage Deliverable D3.1.1, March 2014. Available at:

http://www.paasage.eu/images/documents/paasage_d3.1.1_full.pdf

[D4.1.1] Kyriakos Kritikos, Maria Korozi, Bartosz Kryza, Tom Kirkham, Asterios Leonidis, Kostas Magoutis, Philippe Massonet, Stavroula Ntoa, Antonis Papaioannou, Christos Papoulas, Craig Sheridan, Chrysostomos Zeginis, Prototype Metadata Database and Social Network / Prototype of Metadata Integration Extension. PaaSage Deliverable D4.1.1, March 2014. Available at:

http://www.paasage.eu/images/documents/PaaSage-D4.1.1_final.pdf

[D5.1.1] Anthony Sulistio, Panagiotis Garefalakis, Damianos Metalidis, Chrysostomos Zeginis, Craig Sheridan, Kuan Lu, Jörg Domaschka, Bartosz Balis, Dariusz Król, Edwin Yaqub, Prototype Executionware and Prototype new Execution Engines. PaaSage Deliverable D5.1.1, March 2014. Available at:

http://www.paasage.eu/images/documents/PaaSage_D5_1_1_final.pdf