



PaaSage

Model Based Cloud Platform Upperware

Deliverable D9.3.1

Initial Training Materials

Version: 1

D9.3.1

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

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 website 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:	03/07/2014
Document	
Period covered:	
Deliverable number:	D9.3.1
Deliverable title	Initial Training Material
Contractual Date of Delivery:	30/09/2014 (M24)
Actual Date of Delivery:	30/09/2014
Editor (s):	Kyriakos Kritikos, Kostas Magoutis
Author (s):	Daniel Bauer, Etienne Charlier, Jörg Domaschka, Tom Kirkham, Kyriakos Kritikos, Kostas Magoutis, Christos Papoulas, Michaël Van de Borne
Reviewer (s):	Franky Vanraes, Stéphane Waha
Participant(s):	FORTH, UULM, CETIC, STFC
Work package no.:	9
Work package title:	Training and Dissemination
Work package leader:	Pierre Guisset
Distribution:	PU
Version/Revision:	1.1
Draft/Final:	Final
Total number of pages (including cover):	59

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.int/>)



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

Executive Summary

A success of any platform, either commercial or research-based, cannot solely rely on the exhibited functionality, which obviously constitutes a main differentiation point, as such functionality needs to be complemented with sufficient documentation and training materials which explicate the way this functionality can be exploited in the context of the user goals and requirements. Otherwise, the user will get lost in the overwhelm of features and capabilities of the respective platform and eventually resort to another solution which is better documented. To this end, the purpose of this deliverable is to provide an initial but enough set of material which will indicate the way the PaaSage platform can be utilized in accordance to its development status. The latter also justifies the term "initial" as the initial functionality of the platform will not be as extended and sophisticated as possible as in its final release.

A variety of material is provided, from instructions on how to deploy and configure the PaaSage platform or its modules, to the description of the platform's components, to detailed processes on how users can specify their requirements in Camel in order to be exploited by the platform according to the user purposes and goals, and down to the ways the PaaSage's SN can be utilized in order to achieve some particular user tasks, such as the discovery and sharing of critical knowledge which could be exploited in the management of multi-cloud applications.

As different users might have different goals in exploiting the platform and as an organization is made up of different types of users, the presented material is able to target a variety of user types in the context of the possible exploitation scenarios of the platform which include: (a) exploitation of a running PaaSage platform for managing multi-cloud applications and (b) extension of the platform in order to produce an added-value product which can make a differentiation in the market. The targeted user types include: (1) *business users* which are guided in expressing business requirements and organizational information in Camel, (2) *application developers* which can utilize the provided documentation in order to express application requirements as well as discover and share application design knowledge, (3) *software engineers* which are supplied with useful information towards extending the platform in order to produce added-value functionality, and (4) *system administrators* which are guided in deploying and configuring the PaaSage platform. In addition, the provided guide in using the SN can be exploited by any type of user with the desire to discover and share any type of knowledge which can be deemed useful towards the managing of multi-cloud applications.

The PaaSage platform is under continuous development but it will stabilize into a final prototype at M45 with the development of the respective modules to end at M36. In this way, it will be possible to finalize the existing material that is presented in this document as well as provide additional material which could indicate new ways to exploit the platform based on the extended or novel capabilities exhibited by the final prototype. Such a material will also highlight the success under which the use-cases have been addressed by the platform as well as the main benefits and differentiation points of the platform. Such a material will also be extensive and clear enough to cover all possible exploitation and utilization details such that no respective inquiries and questions will need to be asked by potential exploiters of the platform. All this material will be incarnated in the next and final version of this deliverable named as "D9.3.2 - Final Training Material and Workshop Product Launch" and due in M45.

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	6
Contents	8
1 Introduction.....	11
2 PaaSage Platform Configuration & Deployment.....	15
2.1 Supported platforms & Resource Requirements	16
2.2 Integration environment setup (Linux)	17
2.3 Integration environment setup (MacOSX)	18
2.4 Deployment of the PaaSage platform	20
3 PaaSage Platform Documentation	21
3.1 Upperware Components	21
3.2 MetaDataDataBase Components	23
3.3 Executionware Components	23
4 Executionware Deployment & Usage.....	25
4.1 Installation	25
4.2 Configuration and Setup	27
4.3 Example: Deploying an Application	30
4.3.1 Creating a Cloud.....	30
4.3.2 Creating Hardware Configurations, Operating Systems and Locations	32
4.3.3 Bootstrap	34
4.3.4 Applications, Services, and Installations.....	35
5 Camel Model Creation	37
5.1 Overview.....	37
5.2 Camel Modelling Process	38
5.2.1 Step 1: High Level Requirements Capture.....	38
5.2.2 Step 2: Detailed requirements capture	39
5.2.3 Step 3: CAMEL.....	40
5.2.4 Example: Simple Requirements Capture	42
5.2.5 Summary	45
6 Social Network User Guide	46
6.1 Site Sections.....	46
6.2 User Login / Register.....	48
6.3 Profile Configuration	48
6.4 User Profile	49
6.5 Social Network Community	50
6.6 Models	52
6.7 Components	55
7 Conclusion	57

Bibliography	58
--------------------	----

1 Introduction

The PaaSage product promises to differentiate by developing a platform which is capable of managing cross-cloud applications. Such a platform could be exploited by various organisations which might have different 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 a platform, sufficient documentation and training material should be in place which will indicate the various exploitation ways of the platform and provide in details all necessary steps and required knowledge that is needed for performing the appropriate steps towards fulfilling these exploitation ways. 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. As the platform is under heavy development, not all material is presented but a small subset which is enough for getting acquainted with the PaaSage platform and using it to perform cloud-based application deployments. A complete set of training material will be presented in the next related deliverable of WP9, namely D9.3.2, which will be available at M45 of the project when the PaaSage platform prototype will be complete.

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 which 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 supplies sufficient documentation about all the components that comprise the PaaSage platform. Such a documentation could be quite interesting for organisations which intend not only to exploit but also adapt the platform for their own purposes. The respective material is more appropriate for software engineers involved in such organisations which have experience in developing new or modifying existing code which will expose additional, added-value functionality.
- Section 4 focuses on a particular module of the PaaSage platform, namely the Executionware, with the intention to exemplify how this module can be deployed and utilized in order 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 execution-ware deployment process described in Section 2. The presented material is more 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. In addition, the content is suited for *developers and testers* that aim at gathering a deeper understanding of the interplay of software components in the PaaSage Executionware as well as of

the low-level Executionware functionality that can be used to verify the correct functioning of this module before linking it to the Upperware and the meta-data database. Finally, the example subsection contained in this section, is suited for any type of user of the platform (e.g., an application owner or a user desiring to deploy an application belonging to a specific organization) targeting at performing application deployments in the cloud, as it shows how to register existing cloud accounts in the PaaSage platform.

- Sections 5 and 6 play complementary roles towards enabling organisations in providing the appropriate information as input to the platform in order to fulfil particular organisation-required tasks, such as the deployment of applications. In particular, Section 5 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 D2.1.1 and D2.1.2) that can 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 6 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 in order to pose in a better and more precise way their requirements. 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 5 is intended to a variety of type 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 6 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.

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 bypassing any technical obstacles.
- *business users*: Such users, which might not have a background on IT, can benefit from the material provided in order to obtain and share knowledge as well as publish their application models but also to express the 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 which can then be issued into the PaaSage platform for the proper management of the respective applications in the cloud.

- *application developers / IT Architects*: They 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 properly providing the 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 in 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*: They can exploit the material and especially the one provided by the SN in order to find out applications that interest them and deploy them in the cloud by just specifying some quality requirements, if needed, and letting the system deriving the additional requirements and information required 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. It should also be noted that 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*).

For each type of user indicated above, we demonstrate in the following table 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.

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	√			~	
PaaSage platform documentation	~			√	
Execution-ware deployment & usage	√			~	

Camel model creation	√	√	√	~	
Social Network User Guide	√	√	√	√	√

As the PaaSage platform evolves, new training and documentation material will be produced which will then be used by the above types of users as well as additional ones in order to better exploit the PaaSage platform according to their organisations' needs.

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 the respective material showed in this deliverable in better training the potential users in exploiting the PaaSage platform.

2 PaaSage Platform Configuration & Deployment

A lot of effort has been put into the automation of configuration and deployment of the PaaSage platform so that it is as easy as possible for a newcomer to get it running. The PaaSage platform is made out of several software components (see Section 3 for a complete reference). Those components code is currently hosted in a Git repository (<http://git.cetic.be>) located in the CETIC datacenter but it will soon be moved to a final and permanent hosting platform. Most of component code is implemented in Java and comes with a maven build file (pom.xml) in order to ease the build. The "jar-with-dependencies" option is enabled into those pom.xml files, so that it can be guaranteed that as few problems as possible occur at run time. A Jenkins¹ instance is connected to the Git repo and builds the components every time a new commit is pushed on the origin. Next, multiple Chef² cookbooks and recipes have been developed in order to automate the deployment and configuration of the jar files Jenkins builds. An additional tool, named Kitchen³, a layer on top of Chef, takes care of the PaaSage platform VM provisioning and triggers the Chef cookbooks installation.

In order to ease the maintenance of the components, various guidelines and best practices have been decided:

- The PAASAGE_CONFIG_DIR environment variable stores the path to the default PaaSage platform configuration directory, which contains the configuration files for each component
- Each component configuration file sticks to the java properties format.
- Log files are stored in /var/log/paasage
- Java components must come with a pom.xml build file so that they can be automatically build via Maven⁴
- The jar-with-dependencies option in pom.xml must be enabled so that all code dependencies are resolved as they are shipped with their respective components

¹ jenkins-ci.org

² www.getchef.com/chef/

³ <https://docs.getchef.com/kitchen.html>

⁴ maven.apache.org

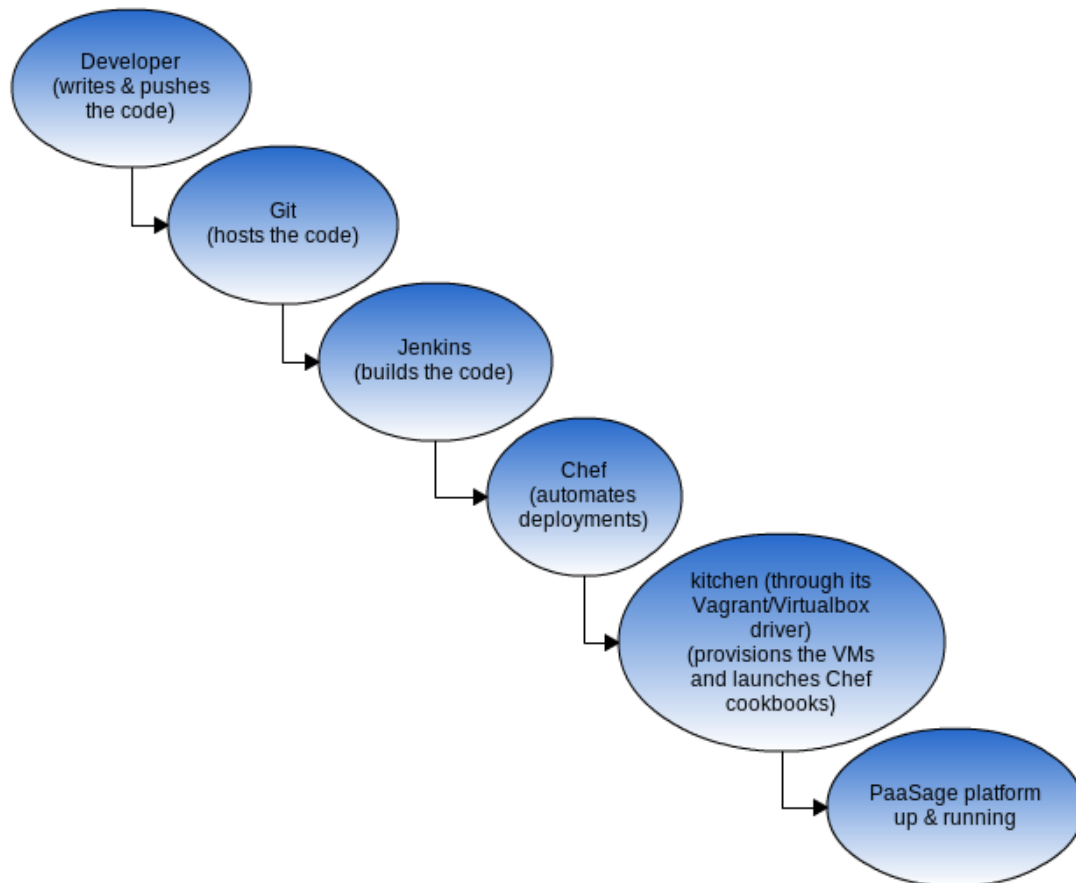


Figure 1: The PaaS platform integration and building process

Thanks to this continuous integration procedure (see also Figure 1), PaaS platform deployments and tests are straightforward and guarantees the use of latest code versions.

In the following, the procedure to set up the PaaS platform on your local workstation is analyzed for each supported platform by first explaining what these supported platforms are and which resource requirements must be met for the local workstation. This procedure might be modified in the near future to become more automated and possibly signify that PaaS can deploy itself. The latter capability could be enabled by, e.g., describing the actual deployment requirements of the platform in Camel and then allowing the respective deployment system to deploy the platform in one or even multiple clouds.

2.1 Supported platforms & Resource Requirements

The currently supported platforms are Linux and Mac OS, where: (a) although any modern Linux distribution can be exploited, the PaaS platform deployment has

only been tested under Ubuntu 14.04; (b) similarly, although previous Mac OS might be exploited, only deployment on MacOSX 10.9 Maverick has been tested.

Apart from the above supported platforms, the host of the platform should preferably have at least 4 GB of RAM, while 8 GB lead to even much better performance.

2.2 Integration environment setup (Linux)

The steps provided below must be performed to setup the integration environment in Linux. After all these steps are performed, your workstation will be ready to launch the PaaSage platform's VM.

1. Install Git

```
sudo apt-get install git
```

2. Download and install ChefDK (<http://www.getchef.com/downloads/chef-dk/ubuntu/>).
3. Download and install VirtualBox and Virtualbox Extension Pack (<https://www.virtualbox.org/wiki/Downloads>) according to you operating system.
4. Download and install Vagrant (<http://www.vagrantup.com/>) according to you operating system.
5. Install Vagrant plugins

```
vagrant plugin install vagrant-vbox-snapshot
```

6. Configure Virtualbox network
 - Open VirtualBox -> File -> Settings -> Networks -> Host Only
 - Edit the vboxnet0 so that: IPv4 address = 10.19.65.1 ; Netmask = 255.255.255.0; DHCP server is enabled (see Figure 2)

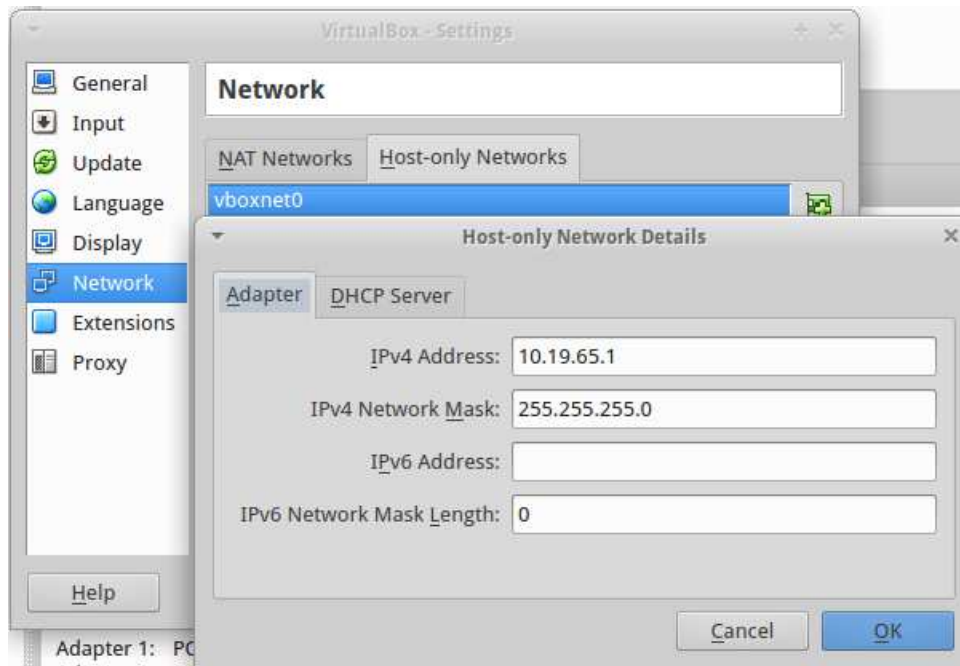


Figure 2: VirtualBox network settings

7. Clone wp6_integration git repository

```
cd $HOME/WorkingCopies
git clone ssh://git@git.cetic.be:61011/paasage/wp6_integration.git
git checkout kitchen
export WP6_INTEG=$HOME/WorkingCopies/wp6_integration
cd $WP6_INTEG/admin
bundle install
# show the list of vm that could be deployed
kitchen list
```

2.3 Integration environment setup (MacOSX)

The steps provided below must be performed to setup the integration environment in MacOS. After all these steps are performed, your workstation will be ready to launch the PaaSage platform's VM.

1. Download and install homebrew (<http://brew.sh/>) + the command line tools
2. Download and install ChefDK (<http://www.getchef.com/downloads/chef-dk/mac/>).
3. Install VirtualBox and Virtualbox Extension Pack (<https://www.virtualbox.org/wiki/Downloads>) according to your operating system.

4. Download and install Vagrant (<http://www.vagrantup.com/>) according to your operating system.
5. Install Vagrant plugins

```
vagrant plugin install vagrant-vbox-snapshot
```

6. Configure virtualbox network
 - Open VirtualBox -> File -> Settings -> Networks -> Host Only
 - Edit the vboxnet0 so that: IPv4 address = 10.19.65.1 ; Netmask = 255.255.255.0; DHCP server is enabled (see Figure 3)

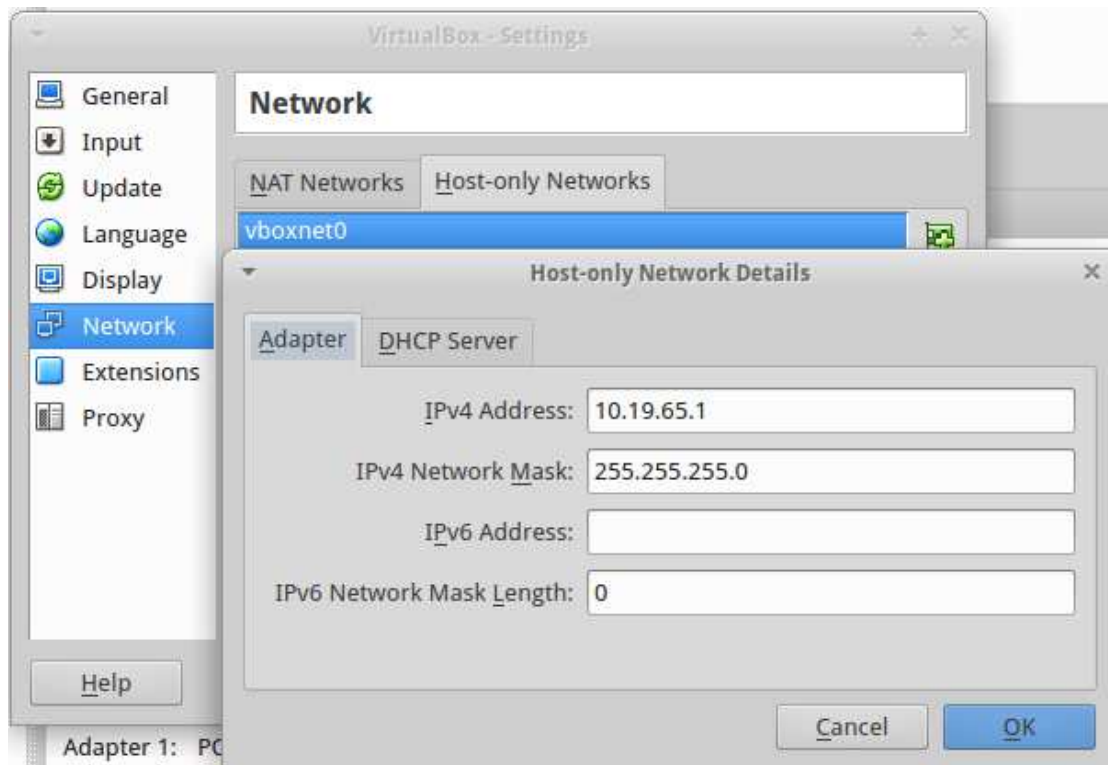


Figure 3: VirtualBox network settings

7. Clone wp6_integration git repository

```
cd $HOME/WorkingCopies
git clone ssh://git@git.cetic.be:61011/paasage/wp6_integration.git
git checkout kitchen
export WP6_INTEG=$HOME/WorkingCopies/wp6_integration
cd $WP6_INTEG/admin
bundle install
# show the list of vm that could be deployed
kitchen list
```

2.4 Deployment of the PaaSage platform

From a functional point of view, nothing prevents the distribution of various platform modules on separate VMs. Every platform module or component was developed with a share-nothing concept in mind so that each one can run separately. However, as the project is still under development, the chef recipes have not yet been adapted to allow such a deployment and, by the time of writing, chef deploys all platform modules and respective components on the same Virtual Machine. The components deployed by the current integration work which provide a basic platform functionality are provided below:

- Adaptation manager
- CDO Server
- Cerif mapper
- CP generator
- Rule processor
- Solver to deployment
- MILP Solver
- Meta solver
- Deploy-play
- Execware frontend
- Cloudify

The following three commands trigger the PaaSage platform deployment:

```
cd $WP6_INTEG/admin
kitchen converge full_paasage_platform
#This will download and install a lot of things, so please be patient for the first time
kitchen login full_paasage_platform
```

Once you've finished working / examining the platform, you can issue:

```
kitchen destroy <instanceName>
```

to destroy the PaaSage platform's VM.

3 PaaSage Platform Documentation

For each component that belongs to the PaaSage platform, specific documentation information has been generated by the respective component developer in the git repository hosting the component's code. This information includes a README file which explicates the main functionality and the way this component can be configured and run. Apart from this information, Jenkins was used to produce the java documentation for each component which is made available at a specific URL for the potential exploiters by invoking the respective *javadoc* module of the corresponding code building tool (e.g., *maven* or *sbt* for *scala*). In the following, the section is split into three main sub-sections, each dedicated to supplying respective component information for each main module of the PaaSage platform, i.e., the Upperware, the MetaDataDataBase (MDDb) and the Executionware. The basic information for each component includes: (a) the location of the component's code in git⁵, (b) the programming language used to implement the code, (c) the organisation which owns and has obviously developed the code, (d) the URL to the java documentation of the code, and (e) a reference to the respective deliverable where additional details about the component are provided (as well as an architectural analysis indicating how this component is connected with other components of the same or different module). Moreover,

It must be noted that as the component code is still under heavy development, the produced java documentation may not contain as much details as could be needed for a potential exploiter. However, it is expected that as the component code gets finalized and the code developers will have additional time to provide more focused code comments in the places needed, the java documentation will become much richer and quite self-explanatory.

It must also be highlighted that apart from particular exceptions, most of the code is not mapped to a particular licence. However, it is expected that quite soon this will be finalized and the respective licence will be included in the git repository hosting the respective code (apart from the code itself which will probably be also annotated). The exceptions concern: (a) the components developed by FORTH (CDO Server and Metrics Collector) which for the moment have a proprietary licence, (b) the components developed by ULM (Execution Engine and Executionware Frontend) which have an Apache Public Licence 2 and (c) the Cerif Mapper developed by AGH which has an Eclipse licence.

3.1 Upperware Components

- CP Generator
 - Git repository:

http://git.cetic.be/paasage/wp3_profiler/tree/master/paasage-wp3-profiler/wp3-cp-generator

⁵ Please be aware that the component location as well as the URL to the respective java documentation will be modified as soon as the component code is moved to the final hosting platform.

- Programming language: Java
- Owning organization: INRIA Lille
- Code documentation:
http://jenkins.paasage.cetic.be/job/WP3_PROFILER_JAR/Javadocs
- Reference: PaaSage Deliverable D3.1.1 [D3.1.1]
- Rule Processor
 - Git repository:
http://git.cetic.be/paasage/wp3_profiler/tree/master/rule_processor
 - Programming language: Java
 - Owning organization: STFC
 - Code documentation:
http://jenkins.paasage.cetic.be/job/WP3_PROFILER_JAR/Javadocs
 - Reference: PaaSage Deliverable D3.1.1 [D3.1.1]
- Upperware metamodel
 - Git repository: http://git.cetic.be/paasage/wp3_model
 - Programming language: Java
 - Owning organization: INRIA Lille
 - Code documentation:
http://jenkins.paasage.cetic.be/job/WP3_MODEL/Javadoc
 - Reference: PaaSage Deliverable D3.1.1 [D3.1.1]
- Metasolver
 - Git repository: <http://git.cetic.be/paasage/metasolver>
 - Programming language: Java
 - Owning organization: STFC
 - Code documentation:
http://jenkins.paasage.cetic.be/job/WP3_META_SOLVER/Javadoc
 - Reference: PaaSage Deliverable D3.1.1 [D3.1.1]
- MILP Solver
 - Git repository: <http://git.cetic.be/paasage/milp-solver>
 - Programming language: Scala
 - Owning organization: AGH University of Science and Technology
 - Code documentation:
http://jenkins.paasage.cetic.be/job/WP3_MILP_SOLVER/Javadoc
 - Reference: PaaSage Deliverable D3.1.1 [D3.1.1]
- Solver to deployment

- Git repository: <http://git.cetic.be/paasage/solver-to-deployment>
- Programming language: Java
- Owning organization: INRIA
- Code documentation:
http://jenkins.paasage.cetic.be/job/WP3_SOLVER_TO_DEPLOYMENT/Java_doc
- Reference: PaaSage Deliverable D3.1.1 [D3.1.1]
- Adaptation manager
 - Git repository: <http://git.cetic.be/paasage/adaptation-manager>
 - Programming language: Java
 - Owning organization: INRIA
 - Code documentation:
http://jenkins.paasage.cetic.be/job/WP3_ADAPTATION_MANAGER/Javadoc_c
 - Reference: PaaSage Deliverable D3.1.1 [D3.1.1]

3.2 MetaDataDataBase Components

- Metadata database / CDO Server
 - Git repository: <http://git.cetic.be/paasage/cdo-server>
 - Programming language: Java
 - Owning organization: ICS FORTH
 - Code documentation:
http://jenkins.paasage.cetic.be/job/WP4_CDO_SERVER/Javadoc
 - Reference: PaaSage Deliverable D4.1.1 [D4.1.1]
- CERIF mapper
 - Git repository: <http://git.cetic.be/paasage/wp4-cerif-mddb-plugin>
 - Programming language: Clojure
 - Owning organization: AGH University of Science and Technology
 - Code documentation:
http://jenkins.paasage.cetic.be/job/WP4-CERIF-MDDB-PLUGIN/Code_docs
 - Reference: PaaSage Deliverable D4.1.1 [D4.1.1]

3.3 Executionware Components

- Execution engine
 - Git repository:

https://github.com/dbaur/cloudify/tree/2.7.0_paasage

- Programming language: Java
- Owning organization: Ulm University
- Code documentation: <http://getcloudify.org/>
- Reference: PaaSage Deliverable D5.1.1 [D5.1.1]
- Executionware frontend
 - Git repository: <http://git.cetic.be/paasage/execwarefrontend>
 - Programming language: Java
 - Owning organization: Ulm University
 - Code documentation:

http://jenkins.paasage.cetic.be/job/WP5_EXECWARE_FRONTEND_PLAY/Javadocs/

- Reference: PaaSage Deliverable D5.1.1 [D5.1.1]
- Metrics wrapper
 - Git repository: <http://git.cetic.be/paasage/metrics-wrapper>
 - Programming language: Java
 - Owning organization: ICS FORTH
 - Code documentation:
 - Reference: PaaSage Deliverable 5.1.1 [D5.1.1]

4 Executionware Deployment & Usage

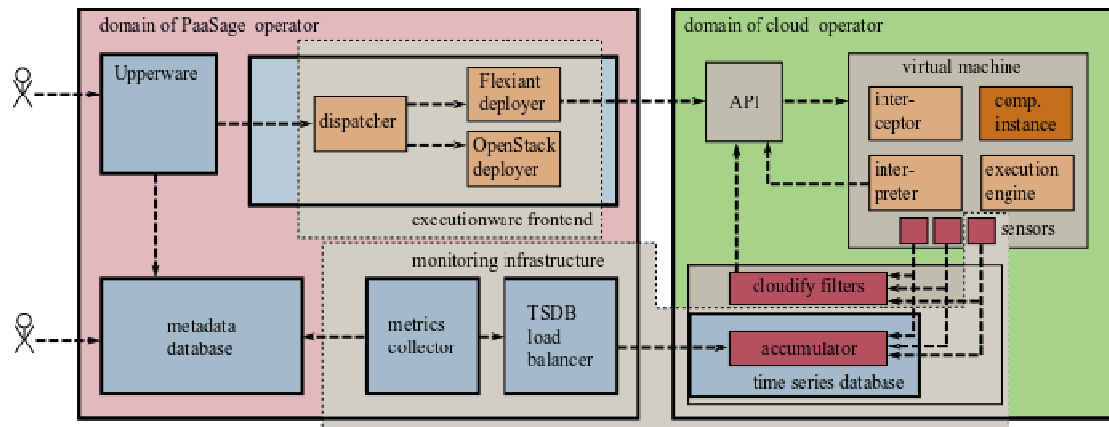


Figure 4: Executionware architecture within PaaSage

As stated in Deliverable D5.1.1 [D5.1.1], the PaaSage Executionware has the primary purpose of bringing applications to execution and to monitor their run-time performance. Figure 4 sketches the Executionware architecture and its role in PaaSage.

The Upperware components [D3.1.1] and the MDDB [D4.1.1] are installed at the premises of the PaaSage operator. The PaaSage operator as well as PaaSage users interact with these two parts in order to access PaaSage functionality. Similar, the Executionware consists partially of components that have to be installed at the premises of the PaaSage operator. This is the Executionware frontend and the monitoring infrastructure. Other parts of the Executionware are then automatically set-up by the frontend when an application is deployed through PaaSage.

Besides being integrated in PaaSage, the Executionware frontend can be used as a stand-alone component for deploying applications. For that purpose, it comes with a graphical, web-based user interface. This feature is useful in order to test the current set-up and ensure the Executionware is working. This section deals with the installation and configuration of the Executionware frontend. In contrast to Section 2, it presents the individual technical requirements for each dependency it has. It also instructs how to build the code from scratch. This deepens the understanding of the overall Executionware architecture and functionality; in particular for developers and testers. This is also the reason why this section concludes with a brief description of how to deploy an application through the user interface.

4.1 Installation

Table 23: List of components required for running Executionware frontend

Java developers kit, SunJDK or OpenJDK	Version 6 or higher
Play framework	2.3.2
Cloudify	2.7-PaaSage

Hibernate-capable database	e.g. MySQL
----------------------------	------------

The Executionware frontend is implemented on top of the Play framework⁶ and makes use of a Hibernate-compatible⁷ database. Furthermore, for deployment it uses a customized, PaaS-aware version of Cloudify⁸. In the following, we sketch the installation of the frontend components based on an Ubuntu-based Linux using MySQL as a database and OpenJDK 7. We assume that the application is installed by a user **myuser** to an installation directory **~/install**. The installation steps that have to be performed are the following:

1. Install git:

```
sudo apt-get install git
```

2. Install OpenJDK by using your operating system mechanism:

```
sudo apt-get install openjdk-7-jdk
```

3. Install MySQL database:

```
sudo apt-get install mysql-server
```

4. Create and enter installation directory:

```
mkdir ~/install;  
cd ~/install
```

5. Download the Typesafe⁹ reactive platform:

```
wget http://downloads.typesafe.com/typesafe-activator/1.2.10/typesafe-activator-1.2.10.zip
```

The Executionware frontend is implemented on top of the . In the following, we sketch the installation of the frontend components based on an Ubuntu-based Linux using MySQL as a database and OpenJDK 7. reactive platform:

6. Extract the Typesafe platform that will result in a directory

```
~/install/activator-1.2.3-minimal  
unzip typesafe-activator-1.2.10.zip
```

7. Clone the Executionware repository using git:

⁶ <https://www.playframework.com/>

⁷ <http://hibernate.org/>

⁸ <http://getcloudify.org/>

⁹ <http://typesafe.com/platform>

```
git clone ssh://git@git.cetic.be:6100/paasage/execwarefrontend.git
```

8. Create a (so far) empty configuration file (name free to choose):

```
touch dev.conf
```

9. Download the PaaSage-capable version of Cloudify:

```
wget http://eladron.e-technik.uni-ulm.de/cloudify/gigaspaces.cloudify-2.7.0-ga-b5996.tar.gz
```

```
wget http://eladron.e-technik.uni-ulm.de/cloudify/gigaspaces.cloudify-2.7.0-ga-b5996.tar.gz
```

10. Unpack the archive that results in a directory:

```
~/install/ gigaspaces-cloudify-2.7.0-ga  
tar -xzf gigaspaces-cloudify-2.7.0-ga-b5996.tar.gz
```

4.2 Configuration and Setup

Now that all software has been put in place, the individual components must be configured as follows:

MySQL

In mySQL we need to set-up a database as well as a user that is allowed to access this database.

1. Connect to database using: `mysql -u root -p`.
2. Create database dbName.
3. Grant all privileges on dbName.* to 'dbUser'@'localhost' identified by 'dbPassword'.

Fill Config File

Open the config file (dev.conf) using your favourite editor and add the content that is shown in Figure 5.

```
# Secret key properties
# The secret key is used to secure cryptographic functions.
# If you deploy your application to several instances be sure to use the same key!
application.secret="A_VERY_SECRET_KEY"

# Database configuration
db.default.driver=com.mysql.jdbc.Driver
db.default.url="mysql://dbUser:dbPassword@localhost/dbName"

# Cloudify properties
# Cloudify Path, directory for temporary files, tell cloudify to save files
cloudify.path = "/home/myuser/install/gigaspaces-cloudify-2.7.0-ga"
cloudify.temp = "/tmp"
cloudify.safe = true
```

Figure 5: Content for the configuration file dev.conf

Run Play Server

In order to bring up the server running the Executionware, execute the following steps:

1. Change to the Executionware directory:

```
cd /home/myuser/install/execwarefrontend
```

2. Run the play application server:

```
~/install/activator-1.2.3-minimal/activator -  
Dconfig.file=/home/myuser/install/dev.conf start
```

3. This will lead to a running PaaSage Executionware GUI that is accessible at port **9000** and can be accessed with a browser

Please sign in

Display Name

john.doe@example.com

Required

Password

.....

Required

Sign in

Figure 6: Login screen after a successful installation

4. Login using the following default username/password combination:

- Login: *john.doe@example.com*
- Password: *admin*

Logging in results in an empty Executionware configuration that we will use in order to deploy a sample application

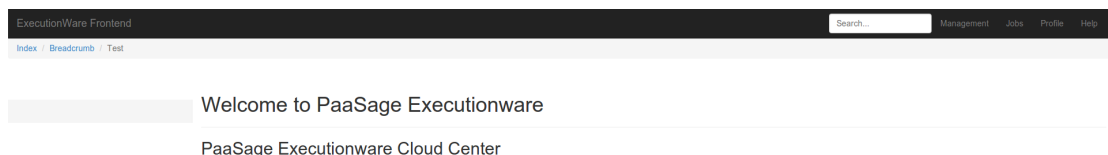


Figure 7: Screen after first login

4.3 Example: Deploying an Application

This section deploys an example application using the Web-based user interface of PaaSage's Executionware. While this GUI-based approach is not PaaSage's primary approach for deploying applications, the GUI helps in order to figure out whether the Executionware has been configured correctly and is indeed operational.

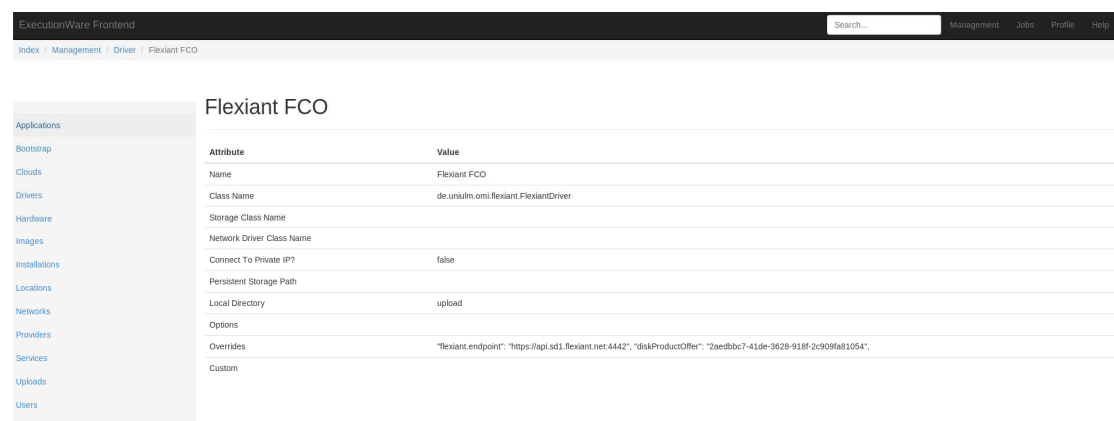
When changing to the **Management** view using the upper right "Management" button in the user interface, the GUI changes and displays a left column listing the core abstractions of the Executionware in alphabetic order. We iterate through all of them in logical order in the following subsections. By doing this, we sketch the functionality of each of them and create a working example.

4.3.1 Creating a Cloud

A **Cloud** is defined by the **User** credentials to access the Cloud Instance offered by a particular Cloud **provider**. The cloud also has a type that is defined by the **driver** used to access the Cloud.

The Cloud driver defines the software used at the server-side (e.g. Amazon EC2, OpenStack, and Flexiant FCO), but also the URI of the service. Hence, for accessing the PaaSage testbed provided by Flexiant, we configure a *Flexiant FCO Driver* as shown in Figure 8 and a *Flexiant FCO Provider* as shown in Figure 9.

The **Driver** configuration consists of an arbitrary driver name (Flexiant CFO in the figure) and class names for accessing various aspects of IaaS offers; in particular, compute services, storage services, and network services. For FCO, the compute driver is capable of handling the other aspects as well so that the remaining aspects can be left empty. *Connect to private IP* denotes whether the virtual machines created for this cloud can be accessed using a public IP or not, where the first case obviously eases the deployment. The most important configuration property, however, is the *overrides* field that defines the URI of the cloud operator.



The screenshot shows the 'ExecutionWare Frontend' interface. At the top, there is a navigation bar with 'Index', 'Management', 'Driver', and 'Flexiant FCO'. A search bar is also present. On the left, a sidebar lists various categories: Applications, Bootstrap, Clouds, Drivers, Hardware, Images, Installations, Locations, Networks, Providers, Services, Uploads, and Users. The main content area is titled 'Flexiant FCO' and displays a table of configuration attributes and their values.

Attribute	Value
Name	Flexiant FCO
Class Name	de.univm.ont.flexiant.FlexiantDriver
Storage Class Name	
Network Driver Class Name	
Connect To Private IP?	false
Persistent Storage Path	
Local Directory	upload
Options	
Overrides	"flexiant.endpoint": "https://api.sd1.flexiant.net:4442", "diskProductOffer": "2aadb7c7-41de-3628-918f-2c909fa81054",
Custom	

Figure 8: The Flexiant FCO Driver

ExecutionWare Frontend

Search...

Management

Jobs

Profile

Help

Index

Management

Provider

Flexiant FCO

Applications

Bootstrap

Clouds

Drivers

Hardware

Images

Installations

Locations

Networks

Providers

Services

Uploads

Users

Flexiant FCO

Attribute	Value
Display Name	Flexiant FCO
Cloudify Name	flexiant
Cloudify URL	http://eladron.e-technik.uni-ulm.de/cloudify/gigaspaces-cloudify-2.7.0-ga-85996
Machine Name Prefix	cloudify-agent
Management Only Files	
SSH Logging Level	WARNING
Management Group	cloudify-manager
Number of Management Machines	1
Reserved Memory Capacity per Machine (in MB)	1024

Figure 9: The Flexiant FCO Provider

The **Provider** configuration in turn comprises configuration properties regarding the set-up of all virtual machines started in a dedicated cloud, e.g., where to get the cloudify run-time, prefixes of virtual machines and management virtual machines. In addition, it defines how the system deals with management virtual machines by, e.g., defining their total number and the memory they use.

The **User** entity encapsulates the access credentials to the Cloud such as user name (called *user* in the GUI) and password (called *apiKey*). It also enables the specification of a key file that will later be used to access virtual machines using PKI authentication. For clouds with FCO driver, the key does not need to be set. For Clouds based on OpenStack, this is necessary, however. The combination of Provider, Driver, and User allows the definition of **Cloud** as shown in Figure 10.

ExecutionWare Frontend

Search...

Management

Jobs

Profile

Help

Index / Management / Cloud / Flexiant

Applications

Bootstrap

Clouds

Drivers

Hardware

Images

Installations

Locations

Networks

Providers

Services

Uploads

Users

Flexiant

Attribute	Value
Display Name	Flexiant
Cloudify Name	flexiant
User	joerg.domaschka@uni-ulm.de/88e8640c-7aea-35c1-9811-bca3aa955788
Provider	Flexiant CFO
Driver	Flexiant CFO

Figure 10: A Cloud configuration to access the Flexiant testbed

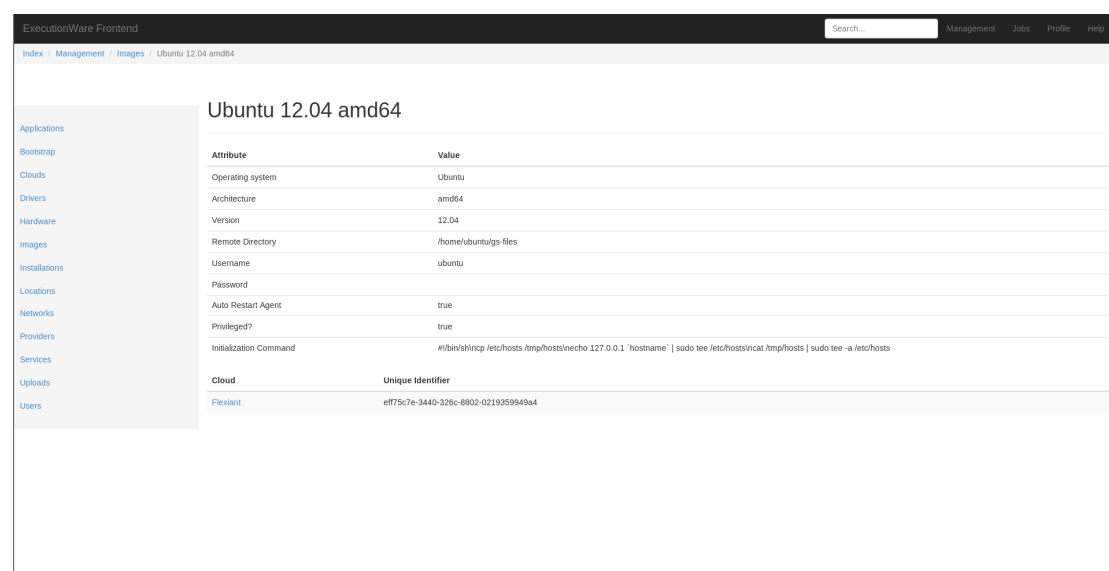
In addition each clouds requires an **Upload**. An Upload depicts custom files that will be used when setting up a virtual machine. They are uploaded as a zip file. There are mainly two important files regarding the upload:

- *bootstrap-management.sh*: This file needs to be contained in every upload zip. The *bootstrap-management.sh* can be downloaded from the executionware webpage.
- *Keyfile*: If you want to use public/private key authentication for your virtual machines, the private key which should be used, also needs to be contained in the uploads zip file.

When using specific cloud providers a **Network** may be required. Currently, this is true when using Flexiant and OpenStack. A Network configuration is depicted by its unique identifier given by the cloud provider, and the cloud it belongs to. In addition a *custom configuration* can be added, which can enable, e.g., the usage of floating IPs.

4.3.2 Creating Hardware Configurations, Operating Systems and Locations

When applications are to be deployed on IaaS systems, they are to be bound to a dedicated image that encapsulates the operating system (cf., Figure 11, Figure 12) and in addition a hardware configuration (cf., Figure 12) that encapsulates the number of CPUs and the available amount of memory.



The screenshot shows the 'ExecutionWare Frontend' interface. The breadcrumb trail is 'Index / Management / Images / Ubuntu 12.04 amd64'. The left sidebar lists various categories: Applications, Bootstrap, Clouds, Drivers, Hardware, Images, Installations, Locations, Networks, Providers, Services, Uploads, and Users. The main content area is titled 'Ubuntu 12.04 amd64' and contains a table of attributes and values.

Attribute	Value
Operating system	Ubuntu
Architecture	amd64
Version	12.04
Remote Directory	/home/ubuntu/gs-files
Username	ubuntu
Password	
Auto Restart Agent	true
Privileged?	true
Initialization Command	#!bin/sh;ncp /etc/hosts /tmp/hosts/echo 127.0.0.1 'hostname' sudo tee /etc/hosts/ncat /tmp/hosts sudo tee -a /etc/hosts
Cloud	Unique Identifier
Flexiant	ef75c7e-3440-326c-8802-0219359949a4

Figure 11: Image configuration

The image configuration covers basic information that is required by the runtime environment. It names the operating system contained in the image, together with the version number and the hardware architecture the image is built for. In addition, a username and an associated password can be set. This is used by the Executionware runtime to get access to the running virtual machine. The *remoteDirectory* property defines the directory the run-time will use for temporary files. The *initializationCommand* property contains a set of commands to be executed at bootstrap. In the example, the local IP address is set to the IP address assigned by the cloud platform. The *privileged* property defines if this image requires commands to be executed with *sudo*. This is mainly required for Ubuntu cloud images. The *autoRestartAgent* property should be set true, as this means that the cloudify agent installed on the machine gets automatically restarted after a reboot.

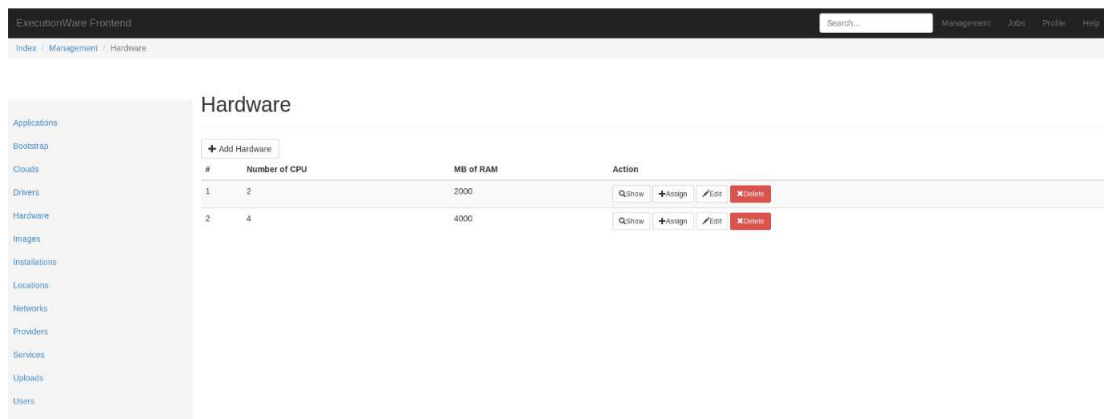


Figure 12: Hardware Configuration

The configuration basically defines the hardware properties of a virtual machine. In particular, it is possible to define a set-up consisting of a particular number of CPUs and amount of RAM. Figure shows two configurations, one with 2 CPUs and 2000 MB of memory and another one with 4 CPUs and 4000 MB of memory.

A **Location** denotes a particular location within a cloud. For Flexiant FCO this refers to a virtual data centre. For Amazon EC2 this refers to an Amazon location (data center) such as Europe, Asia, or West Coast. Regarding OpenStack, the location is not required as it is contained in the identifiers for image and hardware.

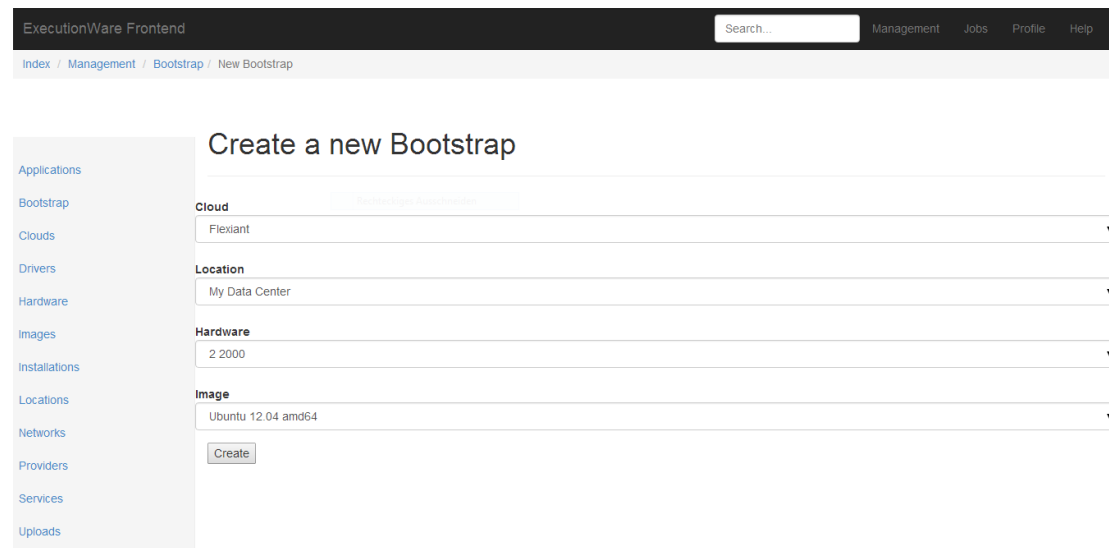
After creating an **Image**, **Hardware** or **Location**, those need to be assigned to the cloud they are available in. For this purpose, the *Assign*-Button (cf., Figure 12) can be used. By giving the unique identifier of the Image, Hardware or Location Resource in the specific cloud, the assignation can be completed (cf., Figure 13).



Figure 13: Assigning the Hardware to a specific cloud

4.3.3 Bootstrap

A **Bootstrap** is the action of actually activating a Cloud. This means that the management virtual machine gets started including the remote monitoring instances (time-series database). Besides being started, the latter is also linked with the metrics collector (monitoring infrastructure residing at the PaaSage operator's premises; cf., Figure 4) so that monitoring data reaches the meta-data database.



The screenshot shows the 'ExecutionWare Frontend' interface. At the top, there is a dark navigation bar with a search box and links for 'Management', 'Jobs', 'Profile', and 'Help'. Below this, a breadcrumb trail reads 'Index / Management / Bootstrap / New Bootstrap'. On the left, a sidebar lists various categories: Applications, Bootstrap, Clouds, Drivers, Hardware, Images, Installations, Locations, Networks, Providers, Services, and Uploads. The main content area is titled 'Create a new Bootstrap' and contains a form with the following fields: 'Cloud' (a dropdown menu with 'Flexiant' selected), 'Location' (a dropdown menu with 'My Data Center' selected), 'Hardware' (a dropdown menu with '2 2000' selected), and 'Image' (a dropdown menu with 'Ubuntu 12.04 amd64' selected). A 'Create' button is located at the bottom of the form.

Figure 14: Bootstrapping a Flexiant Cloud

Figure 14 shows the creation of a Bootstrap instance. Once this is created, the Cloud is bootstrapped and the management virtual machine is created. The result of this action can be seen by clicking on the *Jobs* icon in the right top corner (c.f., Figure 15). Here, also the output to the operation can be seen. Successful jobs are shown with a green outline, failed jobs with a red. More details to the jobs can be seen by clicking on the *Show*-Button.

ExecutionWare Frontend					
			Search...	Management	Jobs
Index / Jobs					
Jobs					
#	State	Created	Started	Finished	Actions
58	SUCCESS	2014-08-08 12:44:27.0	2014-08-08 12:44:27.0	2014-08-08 12:59:09.0	QShow
59	FAILED	2014-08-08 13:02:45.0	2014-08-12 11:38:05.0	2014-08-12 11:38:05.0	QShow
60	FAILED	2014-08-25 14:31:48.0	2014-08-25 17:46:42.0	2014-08-25 17:46:42.0	QShow
61	FAILED	2014-08-25 17:46:42.0	2014-08-25 17:46:42.0	2014-08-25 17:46:42.0	QShow
62	FAILED	2014-08-25 18:22:30.0	2014-08-25 18:22:30.0	2014-08-25 18:22:30.0	QShow

Figure 15: Job Overview

4.3.4 Applications, Services, and Installations

As described in D5.1.1 [D5.1.1], the Executionware has adapted from Cloudfify the concepts of *Services* and *Applications*. While the PaaSage Upperware and its high-level CAMEL models abstract from this low-level Executionware-specific view, using the Executionware directly requires to understand the difference between the two:

An **Application** provides a context to a set of **Services**. There is no logic attached to the Application except that it defines its Services and the interdependencies between them. Services may depend on each other in the sense that one service needs to be started before another one (e.g. the database needs to be started before an application server). Figure 16 shows an application representing a Cloudfify-enabled version of the Ghost blog application¹⁰ consisting of a load balancer service, an application server service, and a database service. All services can be scaled independently and are further deployed independently from each other.

¹⁰ <https://ghost.org/>

ExecutionWare Frontend

Search...

ManagementJobsProfileHelp

[Index](#) / [Management](#) / [Application](#) / [Ghost Blog](#)

Applications

Bootstrap

Clouds

Drivers

Hardware

Images

Installations

Locations

Networks

Providers

Services

Uploads

Ghost Blog

Attribute	Value
Display Name	Ghost Blog
Cloudify Name	ghost

✚ Assign Service

Assigned Service(s)	Actions
Apache Load Balancer	<div>✖ Unassign</div>
MySQL DataBase	<div>✖ Unassign</div>
WebApp	<div>✖ Unassign</div>

Figure 16: An application definition consisting of three services

A **Service** merely consists of a name and an archive file. The archive file contains a set of scripts that are executed at certain points in the service’s life-cycle (install, deployment, startup, shutdown, ...). The content and the structure of these files has been described in D5.1.1 [D5.1.1].

An **Installation** is the deployment of a particular application. The installation view offers an *install*-Button triggering the installation of the application within the defined cloud environment. As with the Bootstrap, the process can be tracked in the Jobs view (c.f., Figure 15).

5 Camel Model Creation

5.1 Overview

The main purpose of Cloud Application Modelling and Execution Language (CAMEL) [D2.1.2] is to capture user requirements for the execution of applications in the Cloud. CAMEL is formed via an aggregation of existing standards to capture user requirements and maintain them through the whole Cloud application lifecycle. The CAMEL models are created using the Eclipse IDE and stored in a central CDO server.

The Camel Models specified in Eclipse IDE can be used to capture the main computing requirements to be associated with the application when deployed on the Cloud. These requirements are expected to be wide ranging and include various perspectives on the application from different stakeholders within the organisation that the user belongs to.

This guide shows the process of requirements capture from the various user perspectives and how this is translated into the Eclipse CAMEL Modelling process.

Note

This guide has been written to support the first PaaSage prototype. It is expected that significant future work will take place in the project to make the process described below both simpler and less open ended.

Pre-Requisites

In order to follow this tutorial, you should have already set up the Eclipse CAMEL modelling environment. Instructions on how to do this can be found in the PaaSage [CAMEL CDO tutorial](#) by Nikolay Nikolov. In addition, some knowledge and familiarity of the Eclipse IDE is required.

Audience

This tutorial is targeted at three types of users:

Application Designer: This type of user is expected to have knowledge of the main execution and deployment characteristics of the application including specific technical requirements, such as platform and data storage and performance.

Business User: This type of user will set the higher level business requirements, such as Cost of application execution, and specific business requirements, such as process data using US hosts.

Systems Admin: This type of user will know the wider technical context from an organisational perspective that the application should execute within. Requirements such as the wider security policies and technical detail can be set by this user type.

The end user which monitors the model and application execution in the Cloud could belong to either one of the user types above depending also on the level of detail and interference that is required. End user choice depends on the organisation's characteristics. It is also possible that users may delegate the setting of requirements

to one of the user types listed above, although requirements from each type of users is needed for the PaaSage platform to provide the optimum Cloud-based application management. The process of requirements capture is split between the perspectives of Business Requirements, Application Requirements and Admin Requirements.

5.2 Camel Modelling Process

5.2.1 Step 1: High Level Requirements Capture

It is expected that an organisation will have a set of clearly defined business objectives associated with the decision to port or deploy an application to the Cloud. These objectives are likely to have formed part of an internal project / business plan for internal approval before the decision to embrace the Cloud via PaaSage was taken.

These business driven objectives will also link to other internal documentation relevant to the security / admin and levels of service for the application in the Cloud. For example a typical objective could be to run the application on an EU Cloud using specially defined Linux virtual machines at a cost lower than 1k per week. This objective spans business (cost), admin (EU only) and application (Linux) specific elements. This scenario will be later used in the CAMEL creation example.

In order to translate these objectives into requirements, we start with the creation of a UML sequence diagram. This step is designed to help capture the main objectives into requirements but also help to describe the main application and its relationship with other entities including other computational elements and business level organisations. It is expected that this task is approved by the main Business, Admin and Application stakeholders linked to the deployment decision.

Resulting from this initial work a high level UML class diagram will be produced. This diagram will represent the key requirements of the cloud-based application deployment including specific rules and constraints linked to the provisioning of the application.

In future developments of PaaSage it is expected that this capture phase will be embedded into a specific PaaSage front end GUI. Here the user will select different capture screens to describe the main business, admin and application requirements. The PaaSage system will then automatically link the sets of requirements into the respective CAMEL model to be exploited for the proper management of the application in multiple-clouds.

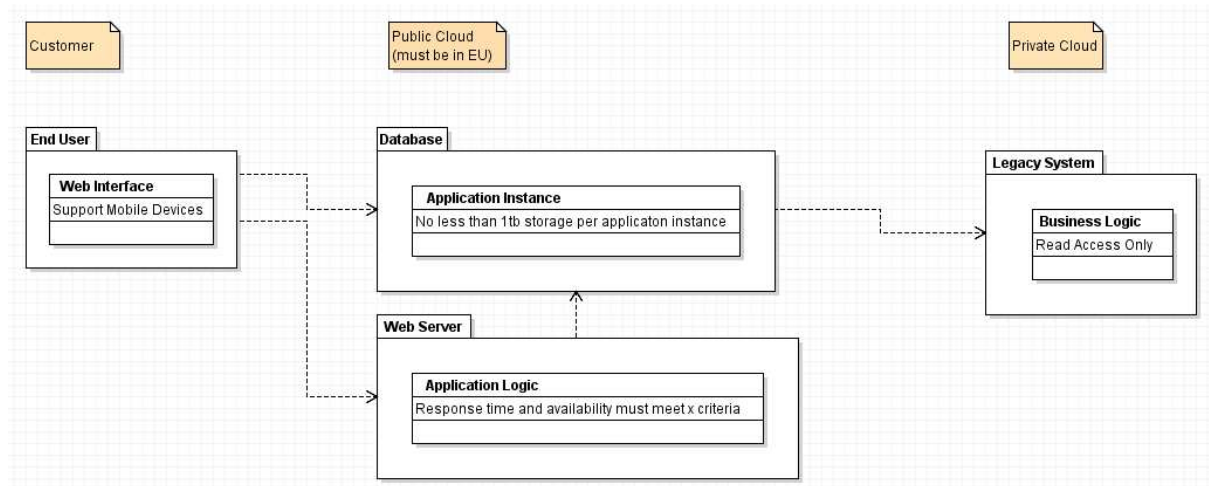


Figure 17: High Level Class Diagram of Deployment

The above diagram is a quite simplistic representation of the desired architecture when the application is deployed to the Cloud. The key points here are to define what components are to be deployed and their main characteristics and interdependencies. Where possible each main component should be broken down into its software requirements and performance characteristics. Boundaries of the Cloud can be used to describe (as illustrated in Figure 17) how some deployments could be hybrid in nature.

5.2.2 Step 2: Detailed requirements capture

Once the high level architecture and relationships between components are defined, in order to ensure that the reasoning PaaSage conducts is suitable to the end users additional detail is added to the model. The additional input includes the prioritization of specific factors such as cost of storage and application execution time to aid automated decision making by PaaSage components.

Users will be required to capture their application workflow in order to model the key points in the application deployment and execution phases. It is expected that the future development of the PaaSage GUI will guide the user to the specific phases and assist them in defining the rules and constraints during this workflow creation process. Once the user driven workflow is complete, the PaaSage components will automatically combine it with the PaaSage lifecycle.

At this stage of the project an interactive workflow based design tool is not yet available to allow users to interact with the possible options available for application deployment and provisioning/execution in the cloud. The project will develop a flexible workflow tool to capture the more detailed requirements which will support the user in modeling their scenario's workflow onto the standard PaaSage lifecycle model identifying constraints / rules at the main PaaSage decision points. **Figure 18** illustrates this process for a generic Cloud deployment.

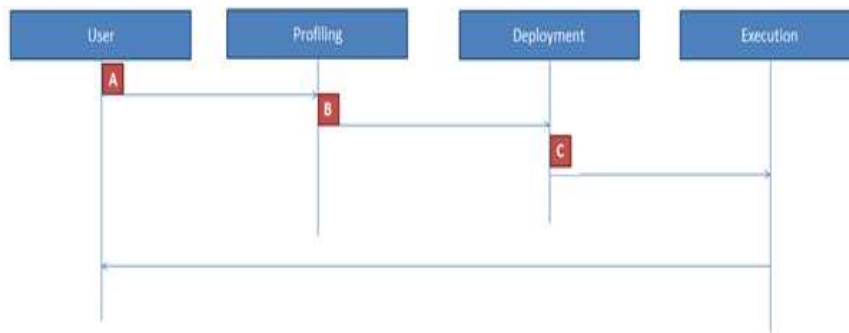


Figure 18: Simple PaaSage Lifecycle

Points A, B and C in the simple PaaSage lifecycle (depicted in **Figure 18**) are typically points at which the user (or alternatively the platform produces automatically on behalf of the user) is required to create specific input, such as functions to aid the management of user defined rules and constraints. Users create or assist in the creation of such an input by ranking importance of specific rules and constraints when defined in CAMEL. It is therefore an important step to have such an input defined before the process of CAMEL creation is started.

Point A is concerned with requirements provided by the end user in terms of priority (e.g., cost is given higher priority than performance).

Point B is where the selection of Cloud providers takes place. Here rules could exist that exclude specific providers or take into account specific historical actions of the provider (stored in the MDDb [D4.1.1]) when making the choice. Utility functions could also be expressed to guide the reasoning process towards discovering the most optimal deployment plan according to the user requirements, constraints and preferences. Such utility functions could be either expressed directly by the user in case he/she is quite experienced in this matter or automatically by the system by considering the user requirements and the priorities posed on them.

Point C is user-defined and specific to the ongoing execution of the Cloud. It could be expressed in the form of scalability rules which indicate those conditions that when met the end-user application should be scaled. Such a scalability rule could be specified by the user, based on extensive testing of his/her application or on live data received during the application execution on the Cloud. Alternatively, the system could generate the scalability rule automatically and suggest it to the user for confirmation based on the previous execution history in the cloud of similar or equivalent applications.

5.2.3 Step 3: CAMEL

Once the high level requirements and more specific input within the PaaSage engagement are defined, the process of CAMEL creation is ready to start. Following on from the pre-requisite step of installing the CAMEL modelling environment, the user is presented with several options when defining a CAMEL root node.

The root nodes in CAMEL are taken from specific DSLs captured in the CAMEL modelling environment (see also Figure 19). More details about each DSL can be found in [D2.1.2].

✚	http://www.paasage.eu/camel/cerif	▶
✚	http://www.paasage.eu/camel/cloudml	▶
✚	http://www.paasage.eu/camel/execution	▶
✚	http://www.paasage.eu/camel/saloon/feature	▶
✚	http://www.paasage.eu/camel/security	▶
✚	http://www.paasage.eu/camel/srl	▶
✚	http://www.paasage.eu/camel/type	▶
✚	http://www.paasage.eu/camel/wsag	▶

Figure 19: The DSLs of CAMEL mapping to the root nodes

DSLs that the root nodes emerge from are listed below with the main areas of information that they cover. It should be noted here that for the capturing of user requirements, not all DSLs are actually needed. However, models for the DSLs that are neglected in requirements capture and stored in CDO Server could be exploited during the requirements modelling as the DSLs are aligned and mapped to each other. For instance, when defining a specific deployment model for an application, the user could define a requirement that a specific cloud provider's VM is used for the deployment of a particular application component which can be linked to the respective cloud provider capabilities description in a model defined via the feature/provider DSL.

- **CERIF:** This is used to map the roles and attributes of users in the application to be deployed in the Cloud. CERIF is actually used for describing organisations, their respective users and user groups, and the corresponding roles that are mapped to these users and groups. Organisations can be cloud providers or particular companies that need to exploit the PaaSage platform for their own needs.
- **CLOUDML:** This is used to model the main architectural nature of the application and its deployment. This includes the application components and the characteristics of the VMs, such as computational resources and operating systems, on which these components will be deployed.
- **PROVIDER/FEATURE:** This is used to capture the Cloud provider capabilities in terms of offered cloud services in a quite expressive way. It is expected that a set of provider models will already be stored in PaaSage's MDDB such that the end-user is not required to specify them but just reference them when e.g. needs to express which cloud services should be mapped to which cloud providers.
- **SECURITY:** This focuses on defining security controls and respective attributes and metrics which can be used for assessing whether the respective security requirements of the user are satisfied. It is expected that the security controls will already be stored in PaaSage's MDDB so the user would just have to express which controls interest him/her and what priorities it set for their satisfaction. Such security control requirements can be used to filter the available cloud provider space based on the providers' security control realizations/capabilities.
- **SRL:** It is used for defining scalability rules which are tied to specific applications and their respective deployments. Each scalability rule is

expressed as a simple or complex condition on one or more QoS metrics whose satisfaction should lead to the execution of the corresponding scalability actions (such as scale-out) by the PaaSage platform. Obviously, apart from widely-known QoS metric templates (already stored in PaaSage's MDDb) which can be instantiated, the end-user has the freedom of defining new QoS metrics and associating them to respective scalability rule conditions.

- **Type:** It is used for defining the values and value types that particular CAMEL constructs/elements can take. For instance, a value type can be bound to a specific QoS metric in order to be able to assess whether the respective metric monitoring values are correct.
- **WSAG:** This actually maps to the WS-Agreement language and thus can be used for defining Service Level Terms as well as Service Level Objectives (SLOs) related to the applications deployment and execution on the Cloud. We should highlight here the connection between WSAG and SRL as SLOs can be expressed as conditions on QoS metrics.

5.2.4 Example: Simple Requirements Capture

In order to demonstrate the process of requirements capture we have created a simple scenario. The main outline of the scenario is explained below:

Business Requirements

Company A has a non-cloud application offered to its customers that is being considered for migration to the cloud. The application consists of a legacy application linked to a web based database and web server presented through a client side web GUI (the UML diagram is expressed earlier in this document in Figure 17).

The Business User defines the core business motivational requirements for using the Cloud. These are that Company A wants to provide greater resource flexibility to the web server and database by provisioning via the Cloud. As a result Company A expects to present a new charging model to its customers and reduce its own costs by selecting the most economical options for deployment. Thus, cost is a significant factor in the setting of the deployment to be optimized.

The Application Designer has knowledge of performance and hosting requirements for the application. Here, the requirements, to support three levels of SLA which are offered Company A via the application to its customers, are integrated. The SLA is linked to application support but also storage capacity and processing power. These factors are expressed as requirements linked to the web server and database to be deployed on the Cloud.

The Systems Admin has knowledge of the security constraints and capabilities within the organisation. For example what type of identity management framework can be supported by clients. Here the requirement is provided that the system should support that company customers (or users representing them) will use their federated identity to gain access to the application. The client side GUI should support mobile devices.

In terms of security, the application handles personal data and therefore appropriate access policies need to be applied linked to national, international and domain specific law. Taking this into account the application must be executed in the EU.

Detailed Requirements

Using the model in Figure 18, detailed requirements are formed at three points. These are expressed in this example as simple functions which determine specific decisions that need to be made at these points of the PaaS Cloud lifecycle.

At Point A, the initial set of detailed requirements is the user defined priorities for the deployment to the Cloud. In this example, the execution must be performed at the cheapest cost possible regardless of processing time.

At Point B, user preferences are provided and linked to Cloud provider selection. In this example, the user does not wish to use the Amazon Cloud and desires to select a Cloud Provider with the highest user reputation mark based on feedback in the MDDb from the Social Network.

At Point C in the example, the user monitors (with the help of the platform) the performance of the application execution. Due to specific changing circumstances (e.g., deterioration in application execution time due to increased load at particular time periods during the day) the user may wish to increase the processing power given to the application to get the application completed quicker. To this end, he/she could feed this in the form of a scalability rule at this stage of the application lifecycle to handle accordingly the equivalent future circumstances that are about to occur.

DSLs

In terms of each DSL, this use case/example should model the following information:

- CERIF: This is used to map the roles and attributes of the end users federated identities to the roles and attributes of the application.
- CLOUDML: This is used to model the main deployment model capturing the three main components above at its core as well as the requirements posed by each of them to the respective VM.
- SECURITY: As the application processes personal data, specific encryption and hosting requirements will have to be enforced to support the deployed application.
- SRL: Scalability rules should be defined and enforced to ensure maintenance of service level dictated by the respective SLA for each customer. For example, a *scale out* action should be executed when the database is reaching capacity or the component VMs run out of computational power.
- WSAG: Service level objectives (SLOs) linked to the 3 tier SLA which are expressed via conditions on metrics that can be monitored and defined in SRL.

Model Creation

In order to set specific elements into the CAMEL, the root node is added relevant to the specific requirement being captured. To demonstrate this, we will use an element from the scenario described above:

Abiding by the security rules, the application should be hosted on an EU Cloud using specially defined Linux virtual machines at a cost lower than 1k euros per week.

From a business perspective, the focus of cost can be captured by defining a specific requirement in WSAG indicating that the respective *Attribute* (defined in SRL) mapped to cost should be restrained with a particular value (1k) and unit (euros). Then, when the user-provided deployment model is fed into the platform, this deployment model will become more concrete by considering the cost constraint into account when mapping the deployment requirements to the respective capabilities of the cloud providers which come with a particular cost.

The application perspective of specific VMs can be modelled using nodes from CloudML. Here the most appropriate way of modelling this would be to create an appropriate root node mapping to a Deployment Model. From this root node, child nodes can be added such as the application components, the VMs and their characteristics, such as required memory and storage, and the mapping of components to VMs via hosting nodes. It should be noted here that the VMs here are described in terms of requirements on their characteristics. However, the user has the freedom to become even more concrete and map particular application components to VMs (or VM types) of specific cloud providers.

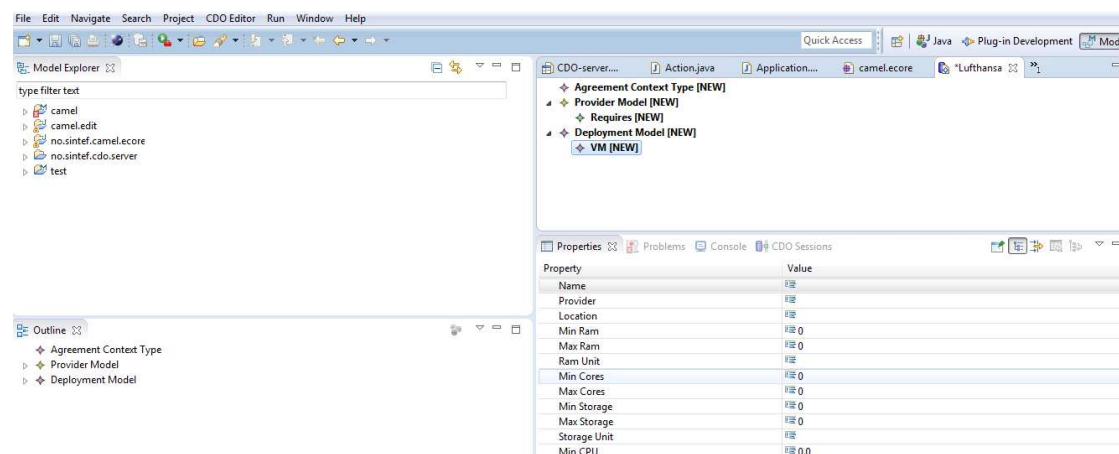


Figure 20: The Eclipse Environment through which the Camel model of the example is specified (and shown at the right top window in the figure)

From a security / administration policy, perspective root nodes from the Security DSL, including Security Requirements that can be linked to specific security controls. Such requirements can be ranked via user-provided properties to allow for a more informed selection of the respective cloud providers.

The process of adding root nodes and drilling down from these nodes to add specific detail is repeated to cover all defined requirements and create the CAMEL model for the application as it is used in the Cloud (see also Figure 20 which shows the Camel model specified for the example).

Functions from the more detailed requirements can be generated from the models by adding specific constraints or rules within the definition of specific nodes. The end result of the process is one or a set of CAMEL models stored in the CDO database. Such models can be added to and modified over time and shared between users and application domains.

5.2.5 Summary

The tutorial above takes a simple view on the creation of a CAMEL model. The process is open ended and the possibility of defining similar constraints using different DSLs is a recognised problem. To address this problem improved integration of DSLs and the production of the GUI-based design tools for the creation of CAMEL is a major focus of the next phase of PaaSage work. Once this improved integration of the DSLs takes place, an extended version of this tutorial will be produced, which will include specific guidelines for each DSL in order to assist the user in defining the different types of requirements for his/her application. The respective example used throughout this tutorial will also be enriched by including particular screenshots and steps which indicate the way particular requirements are formed in specific CAMEL DSLs.








6 Social Network User Guide

The PaaSage Social Network (SN) is a social platform targeting at the creation of a community of users and developers by enabling them 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 modelling recommendations provided by the SN as well as comment and rate such applications models. Soon (end of M30) the SN will be available for users 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.

6.1 Site Sections

The fundamental view of the Social Network Web Site is shown in . In the *top-bar navigation menu* the user can navigate through the key elements of the SN, which are the Models, Components and the Community. The Models are application models described in the Camel Meta-model [D2.1.2], where each model includes a description of one or more Components. The Components are individual units which provide some functionality. 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
	Cart	The user can add to the user's cart any model or components to be used for the design of an application deployment model.
	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.

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 neighbouring information about the *Main Window* and some actions that maybe the user wants to perform, such as filtering the *Main Window* information.

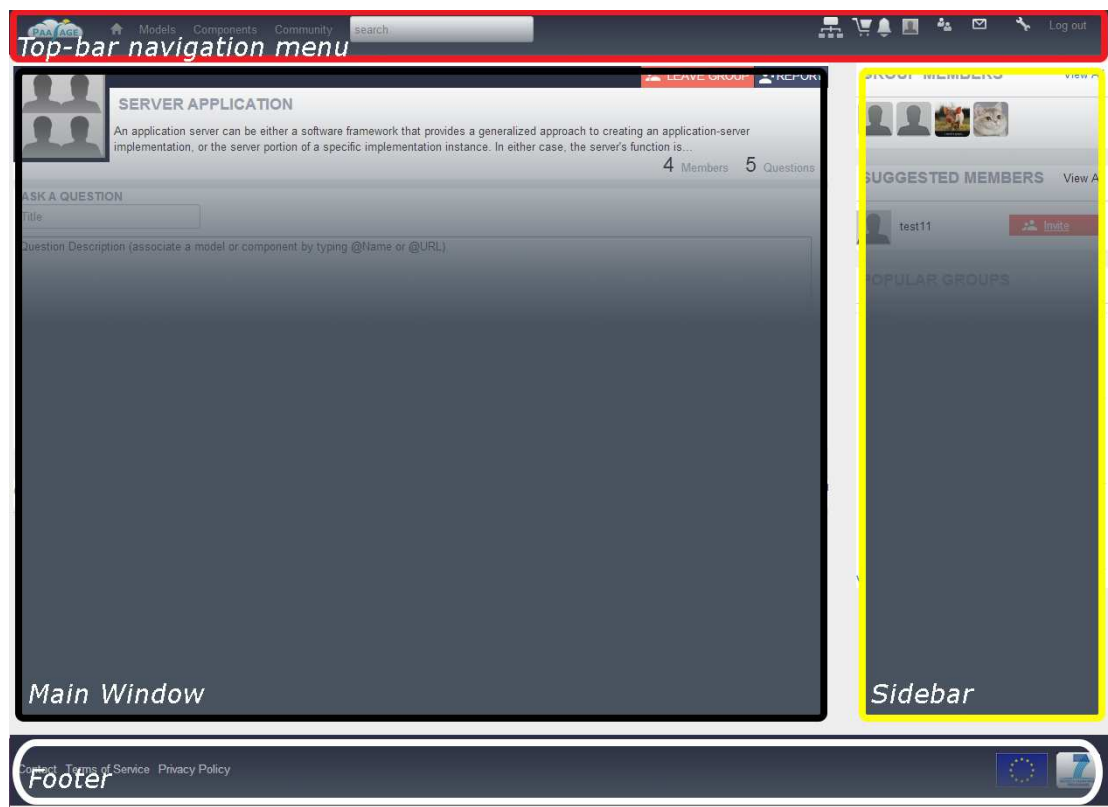


Figure 21: Site Sections: 1) The top-bar section provides the basic navigation options. 2) The Main Window provides the main information that the user desires to see. 3) The sidebar sections exist basically for additional, secondary information and 4) the footer of the site

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

6.2 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 analysed in the following subsection. After the sign-up or log-in process, the user is redirected to the models home page, which is analysed in the last section.

6.3 Profile Configuration

Figure 23 depicts the page of user general settings. The user can navigate to this page either from the settings icon at the right of the top bar or from the profile page. In this page the user can provide additional information as well as determine the privacy level for each information provided.

In the right submenu the user can change the avatar photo on Avatar Settings section or can change the password or other account details by navigating to the *Account Settings*.

The screenshot shows the PAA AGE Social Network (SN) login and registration page. The top navigation bar includes the PAA AGE logo, a home icon, and links for Models, Components, and Community. On the right, there are input fields for 'Username or email' and 'Password', and a 'Log in' button. The main content area is divided into two sections. The left section, titled 'Build application models that can run on multiple Cloud platforms', describes the platform's capabilities and includes a 'Collaborate with a powerful community' section. The right section, titled 'SIGN UP', contains a registration form with fields for 'Display name', 'Email address', 'Username', 'Password', and 'Password (again for verification)', followed by a 'Register' button. Below the main content area, there are two more sections: 'Build and run application models' and 'Join the community'. The footer includes links for 'Contact', 'Terms of Service', and 'Privacy Policy', along with the European Union flag and the PAA AGE logo.

Figure 22: Log-in and Registration Page.

Figure 23: User General Settings

6.4 User Profile

The user's profile page is shown in Figure 24. In the top of the main page the user can see his/her contributions, such as how many models, components, questions and answers 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 his/her top contributions, nominated according to the rating of other users. In the example page shown in Figure 24 the user has asked three interesting questions. 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.

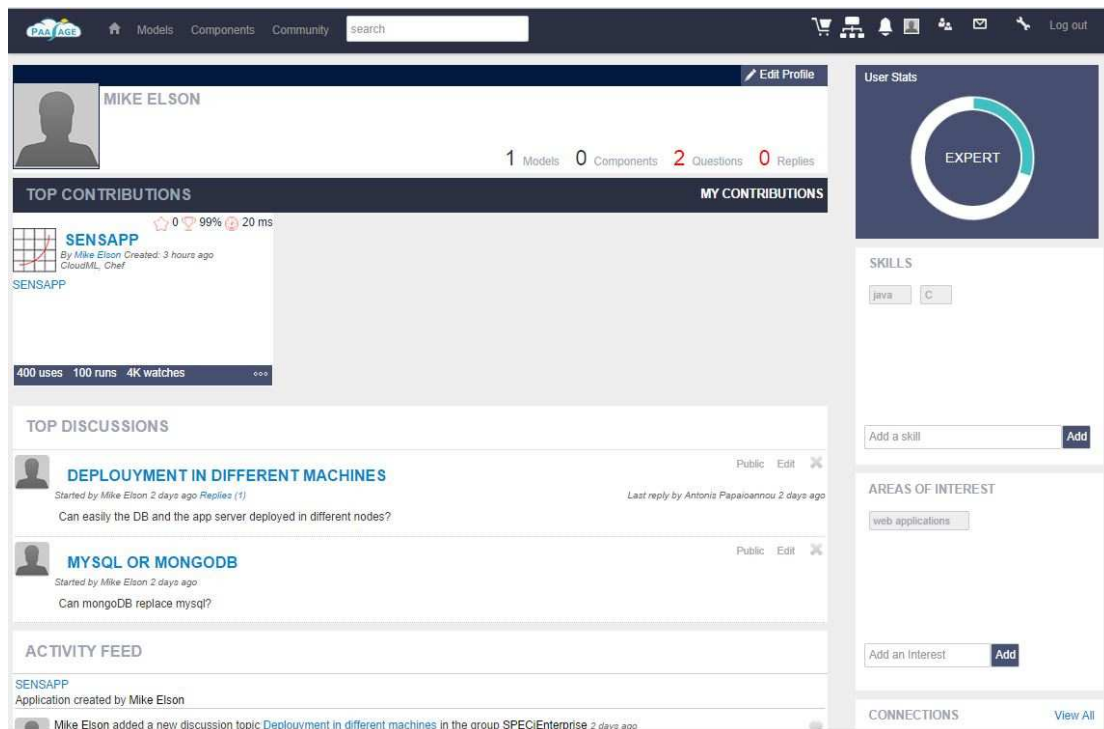


Figure 24: User's profile page.

Figure 25 shows the user's friends page which can be accessed by clicking the *Friends* shortcut or the *View All* link at the bottom part of the side part of the profile page. 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.

6.5 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 components to a question or a reply.

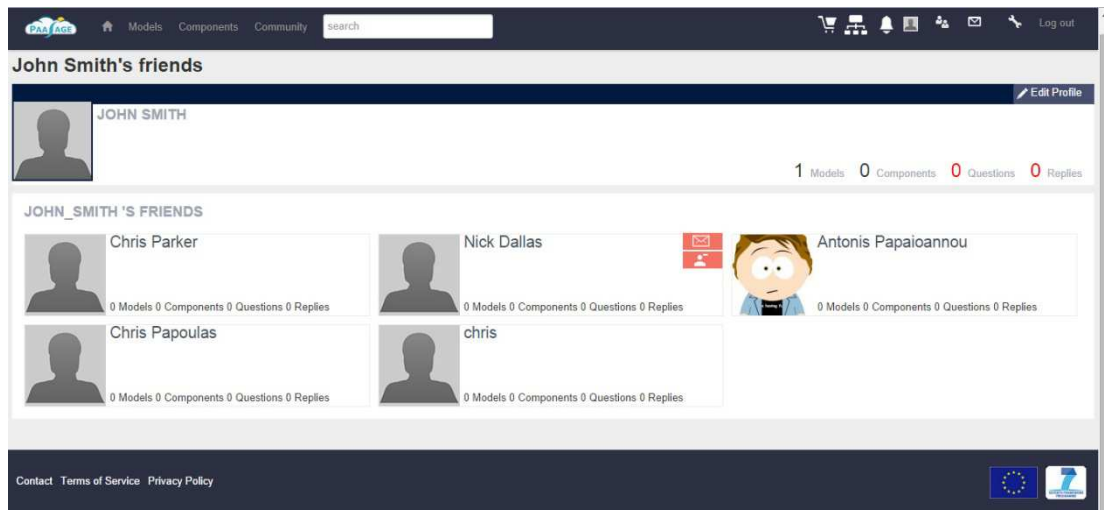


Figure 25: User's Friends

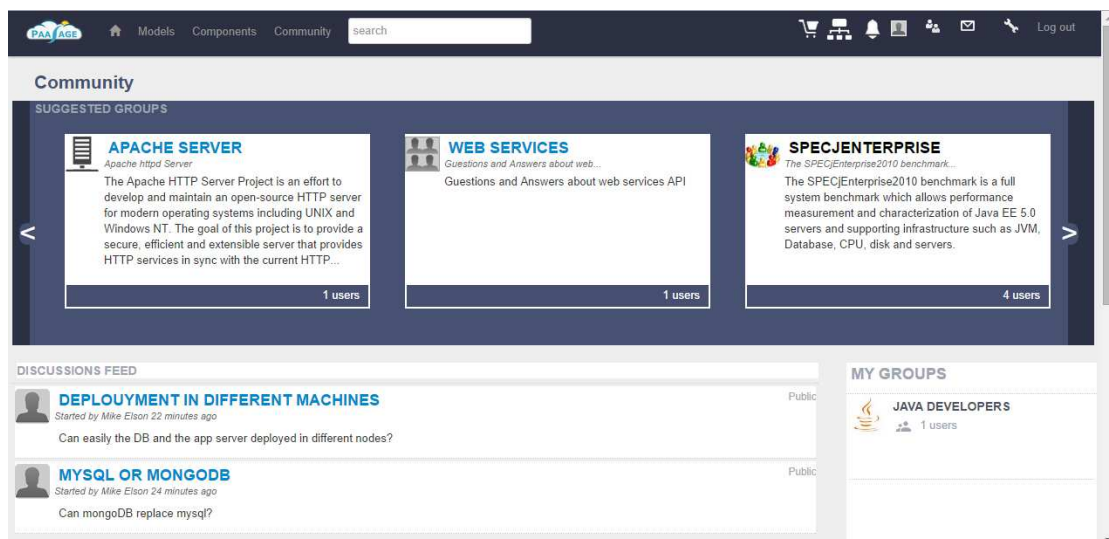


Figure 26: Community Home Page

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.

When a user finds and joins a group, then the outlook of the group is as the page in Figure 27. Then, 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.

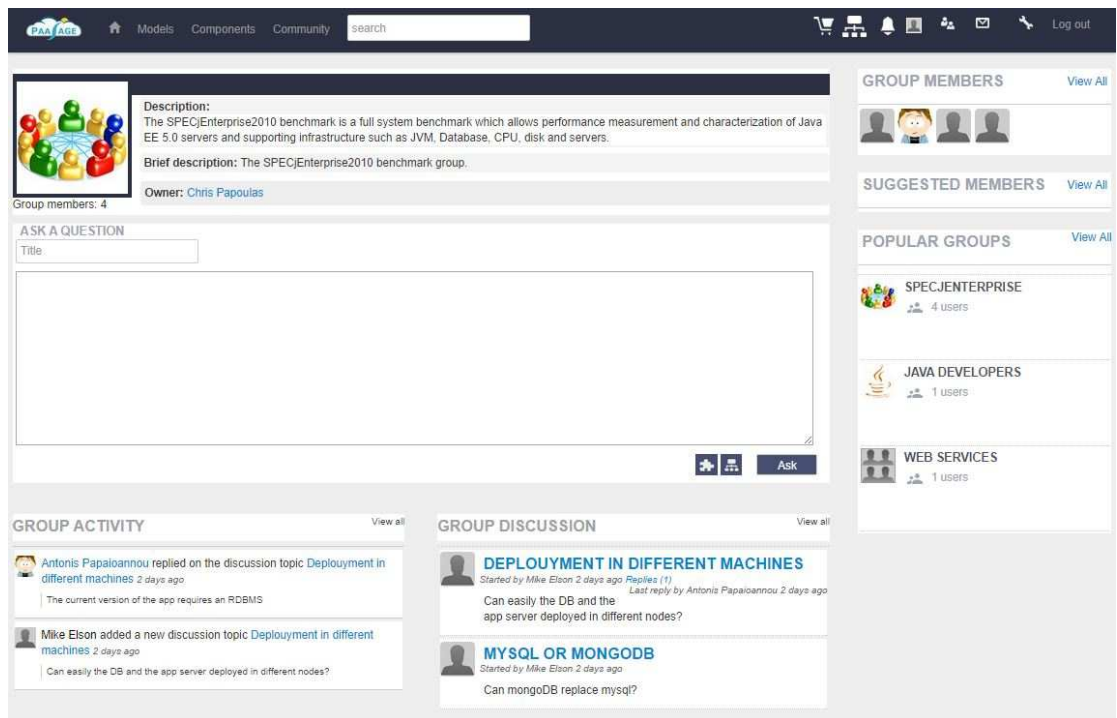


Figure 27: A group's view

6.6 Models

All the applications live in the *Models* section of the SN. The home page of the models is shown in Figure 28. In the top of the main window, the user can navigate through recommended applications according to the tags that he/she has specified.

In the sidebar section, the user can perform the following actions:

- Search for a specific model
- Apply filters such as the deployment cloud, the cost, the response time, the geography etc.
- Add a model to her cart
- Share a model to user's activity feed or
- View more details about a model – full model view.

In full model view, which is the page of , the user can:

- Navigate through the past executions in *Runs* submenu (),
- See the components that the application is composed,
- See or add a review in *Review* submenu () and
- Find similar models.

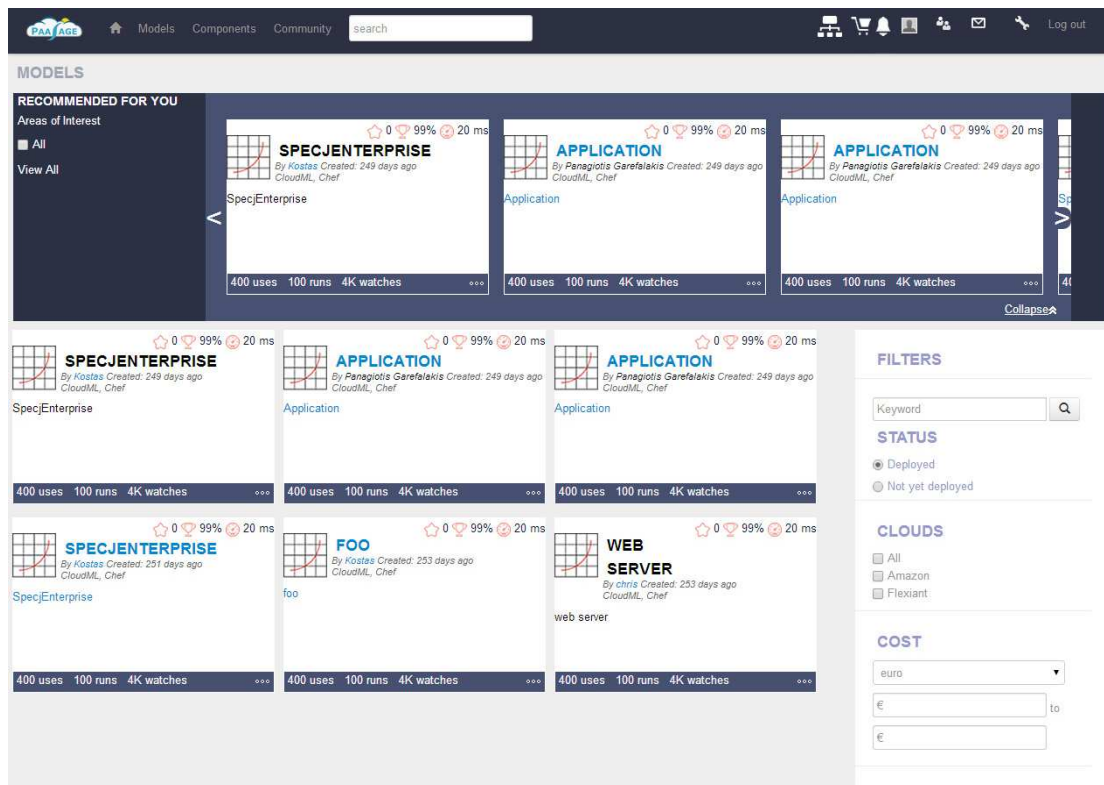


Figure 28: The home page of models.

As mentioned above, in Figure 29 the user can see the past executions of an application model. The user can sort the executions by time, response time, throughput, cost, providers, or status and finally use the configuration of an execution.

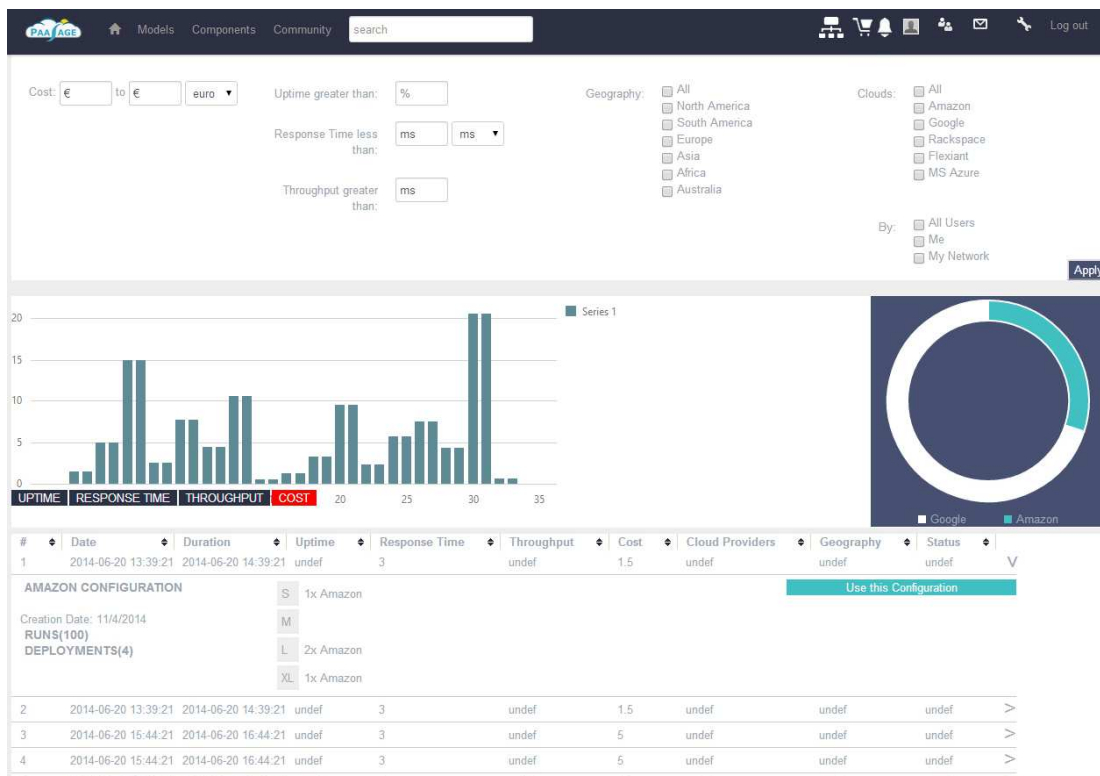


Figure 29: Models: Past Executions Page Section.

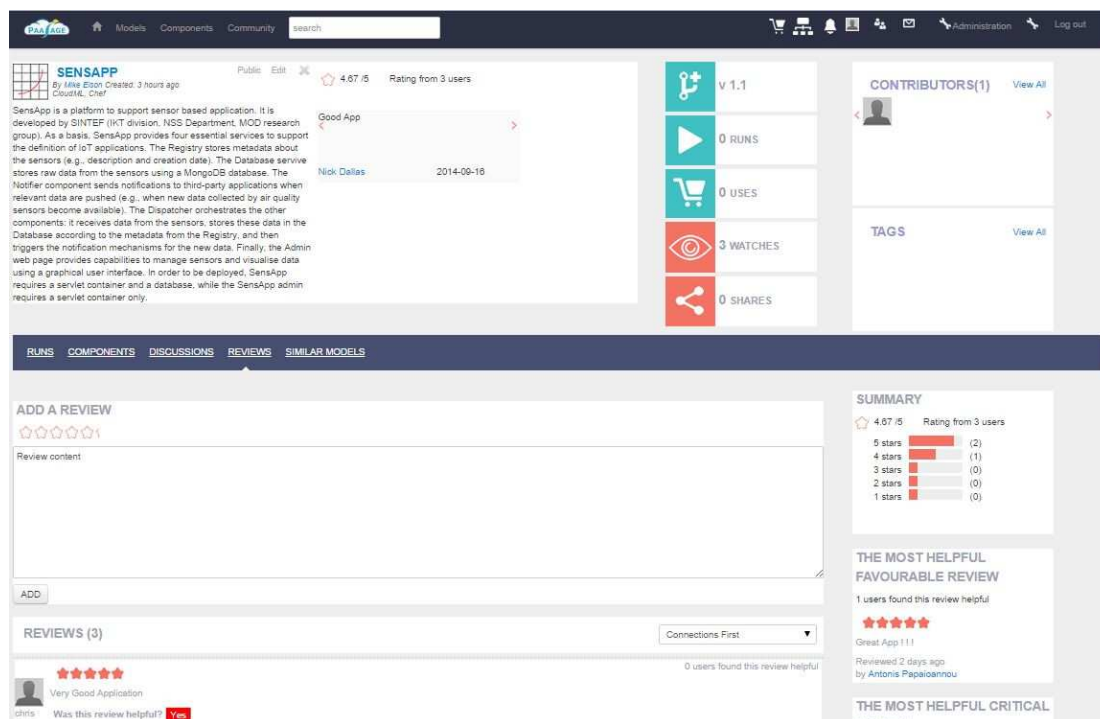


Figure 30: Model's Review Page

6.7 Components

As mentioned above, each application model is composed of one or more components. In PaaSage's Social Network, the Chef Components have been integrated from the Chef Repository to the SN's Elgg Database. The categories of components are:

- Web Servers,
- Utilities,
- Programming Languages,
- Process Management,
- Package Management,
- Operating Systems,
- Networking,
- Monitoring and Trending
- Databases
- Other

Figure 31 shows the component categories of components. We can navigate there through the *Components* link in the top-bar navigation menu. When the user selects a category, he/she can see the list of components in this category as depicted in the page of Figure 32.

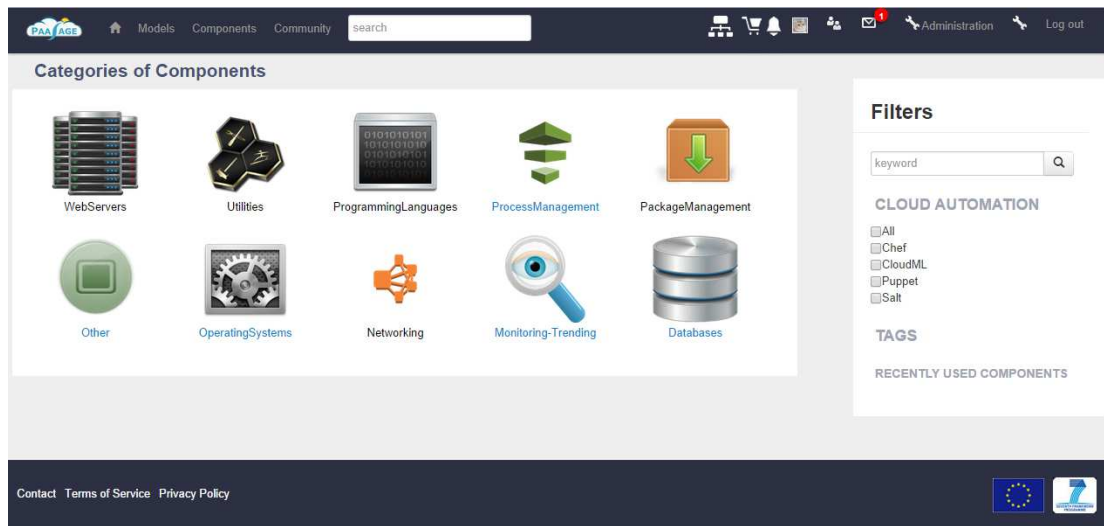


Figure 31: The categories of Chef components

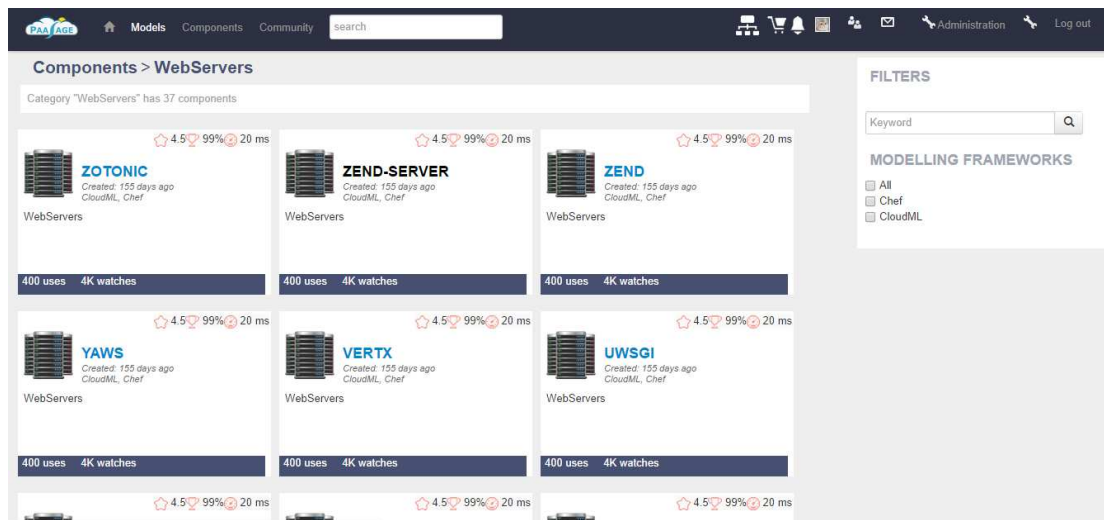


Figure 32: The list of components shown for the Web Servers category

7 Conclusion

This deliverable has provided training material and documentation which can be used as a guide for various types of users in order to perform a variety of tasks, such as the deployment and configuration of the PaaSage platform, the extension of its functionality, the modelling of end-user requirements in Camel and the sharing and exploitation of knowledge incarnated in the PaaSage's Social Network (SN).

While such a material constitutes an initial effort in guiding users how to exploit the PaaSage platform, which is in line with the deployment status of the platform itself, it is expected that the material will evolve as well as new material will be supplied in the next version of this deliverable (D9.3.2 - M45). The existing material could be enhanced as follows:

- It is expected that the PaaSage platform's deployment could be made even more automated. In addition, it is envisaged that the configuration of the platform will become richer even enabling the platform to be deployed in multiple clouds, in accordance with the main objective of the whole project. Thus, the respective documentation will be surely updated, explicating the minimum steps to be performed for deploying the platform as well as the various configuration points that will become available.
- As the PaaSage platform development evolves, existing components will be modified as well as new ones will be developed. Thus, it is expected that in the next version of this deliverable we will see an even more extensive and richer description of even a bigger set of components of the PaaSage platform.
- Similarly to the previous point, the Executionware will be further developed, thus expecting to include additional material pertaining to the modifications as well as the extended functionality that will be exhibited by this PaaSage module.
- Concerning the material in guiding users in expressing their requirements in Camel, it is envisaged that it will be extended towards: (a) showing exactly all the steps needed in order to specify all the requirements of the running example and (b) explicating how the requirements of the various use-cases in PaaSage are expressed in Camel. As Camel is an evolving language/meta-model, we will surely see additional features of the language which will enable to express an even more involved variety of user requirements and this will be anticipated in the forthcoming version of this material.
- The user guide on the PaaSage's Social Network is expected to evolve in providing particular SN exploitation scenarios which indicate how a user is able to perform various simple or complicated tasks, such as registering him/herself or browsing the execution history of existing applications to discover and utilize respective optimal application deployment configurations according to his/her requirements. It is also expected that initial usage information (e.g., statistics) and user feedback from exploiting the SN could be supplied, as soon as the SN launches and becomes available to the public, for assessing whether some of the main goals of the SN are getting close to be realized and providing additional incentives for the users to exploit it.

The new material that we anticipate to foresee in the next and final version of this deliverable is the following:

- Detailed documentation concerning deployment and configuration of not only one but all modules of PaaSage (i.e., MDDB and Upperware).
- Successful use-case stories which include the description of the respective requirements in Camel down to full exploitation of the PaaSage platform in order to achieve the main use-case goals related to multi-cloud application management, such as the optimal deployment of the respective application or the runtime application adaptation based on particular scalability rules.
- Material and user guides for exploiting Camel not only in terms of expressing user requirements but also other types of information, e.g., related to the runtime management of an application, such as the specification of measurement information or the adaptation history of an application. These other types of information will also complement the documentation of the PaaSage components as it will provide useful insight of not only the functionality of a component but also of the way it is expected to manage the respective models or information manipulated by it.
- More visual-based material, such as videos, which will complement the respective description in the supplied user guides, documentations and training materials.

In the end, the final deliverable version will be a collection of structured and both textual and visual material which will enable users to better exploit and utilize the PaaSage platform in many different directions but always according to their requirements and goals. Such a material will also highlight in the best possible way the main benefits of the PaaSage platform and its main differentiation points with respect to the other related commercial or research offerings, thus constituting one extended advertisement point which will eventually cater for a better and more involved exploitation of the platform.

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

[D2.1.1] Alessandro Rossini, Arnor Solberg, Daniel Romero, Jörg Domaschka, Kostas Magoutis, Nicolas Ferry, Tom Kirkham, Maciej Malawski, Bartosz Balis, Dariusz Krol, Achilleas Achilleos, CloudML Guide and Assessment Report (Extended). PaaSage Deliverable D2.1.1e, November 2013. Available at: http://www.paasage.eu/images/documents/paasage_d211_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