LOw Power Heterogeneous Architecture
for NExt Generation of SmaRt Infrastructure and Platforms
in Industrial and Societal Applications

D5.1 - Workload Characterization

Intermediate Version

Co-funded by the Horizon 2020
Framework Programme of the European Union

DELIVERABLE NUMBER	D5.1
DELIVERABLE TITLE	Workload Characterization - Intermediate
RESPONSIBLE AUTHOR	IBM

GRANT AGREEMENT N.	688386
PROJECT REF. NO	H2020- 688386
PROJECT ACRONYM	OPERA
PROJECT FULL NAME	LOw Power Heterogeneous Architecture for Next Generation of SmaRt Infrastructure and Platform in Industrial and Societal Applications
STARTING DATE (DUR.)	01/12/2015
ENDING DATE	30/11/2018
PROJECT WEBSITE	www.operaproject.eu

WORKPACKAGE N. | TITLE	WP5 | Optimized Workload Management on Heterogeneous Architecture
WORKPACKAGE LEADER	IBM
DELIVERABLE N. | TITLE	D5.1 | Workload Characterization - Intermediate
RESPONSIBLE AUTHOR	J.Nider
DATE OF DELIVERY (CONTRACTUAL)	31/05/2017
DATE OF DELIVERY (SUBMITTED)	31/05/2017
VERSION | STATUS	V1 | Final
NATURE	R(Report)
DISSEMINATION LEVEL	PU(Public)
AUTHORS (PARTNER)	NALL, TECH, ISMB, CSI

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	First Draft	28/03/2017	J.Nider - IBM
0.2	Integrated changes from other partners	21/05/2017	J.Nider, A.Scionti, P.Ruiu, L.Scanavino
1.0	Integrated changes from internal reviewers	28/05/2017	R.Chamberlain, R.Peveri

PARTICIPANTS		CONTACT
STMICROELECTRONICS SRL		Giulio Urlini Email: Giulio.urlini@st.com
IBM ISRAEL SCIENCE AND TECHNOLOGY LTD		Joel Nider Email: joeln@il.ibm.com
HEWLETT PACKARD CENTRE DE COMPETENCES (FRANCE)		Gallig Renaud Email: gallig.renaud@hpe.com
NALLATECH LTD		Craig Petrie Email: c.petrie@nallatech.com
ISTITUTO SUPERIORE MARIO BOELLA		Olivier Terzo Email: terzo@ismb.it
TECHNION ISRAEL INSTITUTE OF TECHNOLOGY		Dan Tsafrir Email: dan@cs.technion.ac.il
CSI PIEMONTE		Vittorio Vallero Email: Vittorio.vallero@csi.it
NEAVIA TECHNOLOGIES		Stéphane Gervais Email: s.gervais@lacroix.fr
CERIOS GREEN BV		Frank Verhagen Email: frank.verhagen@certios.nl
TESEO SPA		Stefano Serra Email: s.serra@teseo.clemessy.com
DEPARTEMENT DE L'ISERE		Olivier Latouille Email: olivier.latouille@isere.fr

ACRONYMS LIST

API	Application Programmer Interface
EC2	Amazon Elastic Compute Cloud
ECRAE	Efficient Cloud Resources Allocation Engine
TOSCA	Topology and Orchestration Specification for Cloud Applications
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
RESTful API	Representational State Transfer API
SaaS	Software as a Service
SLA	Service Level Agreement
LAN	Local Area Network

LIST OF FIGURES

Figure 1 - TOSCA Template Model 15
Figure 2 - OpenXchange Architecture Diagram 18
Figure 3 - OwnCloud Architecture Diagram 19
Figure 4 - LibreOffice Architecture Diagram 20

EXECUTIVE SUMMARY

Work Package 5 (WP5) aims at providing a mechanism to efficiently deploy applications on the data centre resources. The "efficiency" expected from the proposed mechanism strictly concerns the energy consumed by the data centre hardware resources and the relative performance of the various tasks composing applications running on the data center. Providing such a mechanism requires to take into account of several elements that influence the runtime execution of such tasks. For instance, a model to evaluate the best mapping of tasks with specific host architectures, a mechanism to select the best host among various with the same architecture where to run the tasks, and a way of optimizing the whole data center workload should be integrated.

The aim of task T5.1 is thus to build an application descriptor that can be used to influence power management and placement or consolidation algorithms. The application descriptor is be based on an existing one, and has been enhanced by adding information on performance and resource requirements, essentially making the application descriptor *energy aware*.

Position of the deliverable in the whole project context

The activities carried out within task T5.1, whose initial results are summarized in this document, are framed in the WP5 ---Optimized Workload Management on Heterogeneous Architecture. Specifically, this deliverable provides the results of the research activity and investigation done by OPERA partners, aiming at developing a software system (ECRAE) to be integrated in the OpenStack orchestration toolchain. The ECRAE system is composed of several components which together are responsible for the mapping of cloud application components on heterogeneous resources.

The deliverable is in connection with the work done in the WP5, specifically with activities in T5.1. This task will accept the results of WP4 as input, which describe a method to measure workload power consumption, and use this information in the characterization of the selected test workloads (specifically the VDI use case). The output from this task is to be used in further tasks (T5.3 and T5.4) to implement the TOSCA descriptor and evaluate its performance as part of WP7.

Description of the deliverable

This document reports the results of the initial design phase of such component. It outlines the work done in researching the requirements of a power-aware descriptor, comparing existing descriptors to determine the most fitting one for modification, and the determining the effort required to integrate the descriptor and its modifications into OpenStack.

List of actions and roles

Activities related to D5.1 involved IBM, NALL, ISMB and CERT. However, also other partners (such as CSI and TECH) provided useful inputs for driving the activities carried out in T5.1.

	LIST OF ACTIONS					
ACTIVITIES LIST AND PARTNERS ROLES	IBM	NALL	ISMB	CERT	TECH	CSI
Requirements Definition	P	P	P	P	I	P
Base Application Descriptor Research	I	I	P	I		P
Integration Definition	I	I	P	I		I
Deliverables and Reporting	P	P	P	P		P

- P = Participating (includes I & R)
- I = Input delivery (Includes R)
- R = review

ISMB: it is the main contributor for the activities carried out in the task T5.1. ISMB started with the analysis and design of the software module used to schedule cloud applications on the data center infrastructure. ISMB also provided the initial implementation of this software module (ECRAE), as well as it studied the integration of a dynamic re-scheduling policy based on an evolutionary algorithm to better optimize the energy efficiency of the data center. Using inputs provided by CERT and TECH partners, a simple but rather effective (power) efficiency model has been integrated in the ECRAE.

CSI: mainly contributed by providing inputs regarding both the integration of the new module developed in the context of T5.1, as well as providing inputs on the exploitation and extension of the TOSCA format to trigger the (static) scheduling phase. CSI provided access to the infrastructure, where OpenStack has been installed and it is used for the purpose of integration (see also WP7 --- VDI use case).

IBM: contributed to this task, by providing the mechanism to migrate Linux containers, also between nodes with different hardware (ISA, configuration, etc.). IBM also actively contributed to the organization and writing of this document.

TECH: provided inputs for the creation of the (power) efficiency model used by ECRAE to select the most suitable server where to execute specific application components (this information is carried out by the TOSCA file).

CERT: similarly to TECH, CERT provided inputs for the creation of the (power) efficiency model used by ECRAE to select the most suitable server where to execute specific application components (this information is carried out by the TOSCA file).

TESEO: reviewed the material provided in this deliverable.

NALL: reviewed the material provided in this deliverable.

TABLE OF CONTENTS

1 INTRODUCTION 9
2 CLOUD SERVICE TYPES 10
2.1 IAAS 10
2.2 PAAS 10
2.3 SAAS 11
3 WORKLOAD CHARACTERIZATION 12
3.1 MICROSERVICES 12
3.2 SPLITTING APPLICATIONS 12
4 APPLICATION DESCRIPTORS 13
4.1 OPENSTACK HOT 13
4.2 USDL 14
4.3 CAMP 14
4.4 CLOUDFORMATION 14
4.5 PUPPET AND CHEF 14
4.6 OTHER TOOLS 15
4.7 TOSCA 15
5 INTEGRATION WITH OPENSTACK 17
6 USE CASE WORKLOADS 18
6.1 OPENXCHANGE 18
6.2 OWNCLOUD 18
6.3 LIBREOFFICE 19
7 CONCLUSIONS 21
8 REFERENCES 22

1 INTRODUCTION

The energy efficiency of a data center is highly dependent on the workload being served. To improve the energy efficiency of a cloud computing data center, it is important to characterize the workloads and to be able to use that information for scheduling. The characterization of a workload can help determine the best compute resources to serve an application, or an application component. This is especially important today when applications are being composed of many smaller components with well-known behaviour, such as in PaaS cloud applications using the microservices architecture. By being aware of the resource consumption requirements of each component, the orchestration software can select the best fit among the compute resources that are available at the time.

Microservices an emerging cloud application architectural style, which considers a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms (e.g. containers such as Docker). Microservices are generally designed to be scalable and resilient, which means they can be deployed on a set of distributed servers, under the assumption that a communication channel (such as TCP/IP) is available to coordinate efforts between the various instances and services. We focus our efforts on characterizing the various microservices that compose the sample applications from the VDI use case. We encode this characterization in an application descriptor that can be used by the orchestration software.

The aim of the task is thus to build an application descriptor that can be used to influence power management and placement or consolidation algorithms. The application descriptor is be based on an existing one, and has been enhanced by adding information on performance and resource requirements, essentially making the application descriptor *energy aware*.

During the first project period, we focused our efforts on selecting the correct application descriptor by doing a study of several possibilities, and comparing them to a set of requirements. We analyzed several software packages from the VDI use case, and determined a subset of the most applicable ones that should be used for testing and demonstration purposes.

The plan for the second project period is to modify the TOSCA descriptor which will then be integrated into an OpenStack installation for testing with the various VDI applications. Then we must apply the characterization to create a descriptor for each of the target applications, which can be used by OpenStack. The results of this effort will be used in task T4.3 for taking the measurements, and to determine efficacy of the solution.

2 CLOUD SERVICE TYPES

The concept of cloud computing originated in network diagrams from the 1990's in which networks beyond the LAN were often represented as clouds to indicate ambiguity of the details. Cloud computing is just that - an ambiguous set of compute resources of which the user has little or no knowledge of the organization. Clouds can be designed to provide different types of services. Customers may choose the service type based on a variety of factors such as their business needs, required speed of deployment, and requirements for customization of applications. We use the following definitions for the types of clouds that interest us in the OPERA project.

2.1 IAAS

Infrastructure as a Service refers to clouds that provide machine abstractions directly to the end user. This is the most basic type of service, as it exposes machines directly. IaaS clouds may be implemented as "bare-metal" in which they expose actual physical machines (such as IBM SoftLayer), but more commonly IaaS refers to virtual machine clouds (such as Amazon EC2). IaaS clouds based on virtualization rely on several layers of software to safely expose the virtual machines to the user. A hypervisor is required on each physical machine to manage the local resources, and divide them among a set of virtual machines. A cloud orchestrator (such as OpenStack) is required to provision a new machine, and for setting up its relevant resources such as shared storage, and communication in the form of virtual LANs.

Since IaaS exposes the virtual machine directly to the end user, the end user has practically full control over the software on the machine including the operating system and drivers all the way up the stack to the applications. This gives the user a large degree of control, but masks a lot of details from the cloud operator as to how efficiently the resources are actually being used. From an economic perspective, this is an efficient model for the cloud operator since the user can be billed for resource allocation, rather than resource utilization. From a resource allocation perspective, the cloud may be operating at a low efficiency level since there are few options available to the cloud operator to improve efficiency without affecting the quality of the service, and potentially violating the service level agreement (SLA).

2.2 PAAS

Platform as a Service refers to a cloud that exposes a set of basic software packages that can be used as building blocks for developing cloud applications. These software packages generally provide the common services an application would require, while removing the burden of installation, configuration, and maintenance from the user. Common services include databases, web servers, and scripting engines (such as PHP or Ruby) to glue all the pieces together. PaaS clouds are designed for users to develop their own applications quickly, and design them to be scalable through the use of microservices.

PaaS clouds give a greater degree of visibility and flexibility to the cloud operator than an IaaS cloud, since the cloud is built on an abstraction of services rather than machines. The cloud operator is able to choose the best method to provide the services, without being tied to a particular set of hardware resources (such as a particular instruction set architecture) or pre-allocating the resources (for example, setting aside 1GB of memory for a virtual machine, regardless of utilization). This fine-grained control is conducive to a more efficient cloud, since the cloud operator now has the ability to see the "big picture", and the options available to reconfigure the cloud according to need.

2.3 SAAS

The third kind of cloud is **Software as a Service**, which refers to a single application which is exposed directly to the end user. SaaS clouds may only run a single application, or they may offer a set of applications - this is highly dependent on the needs and clientele of the cloud operator. One such example is Google Mail (gmail) which is exposed as a software application directly to users, without any visibility as to which services are being consumed internally (databases, web servers, etc). SaaS applications may be implemented on top of PaaS clouds by combining multiple services and exposing an application, but here we make the distinction from the perspective of the cloud operator. If the cloud operator exposes basic services that are available for end users to use to create applications, we define it as a PaaS cloud, whereas if the exposed interface is only to the application itself, it is a SaaS cloud.

SaaS clouds give the greatest degree of flexibility to the cloud operator to be able to reconfigure the infrastructure to respond to changing loads or conditions. Many of the conclusions that can be drawn about PaaS efficiency also hold true for SaaS efficiency, but for the purposes of OPERA, SaaS clouds are less interesting than PaaS clouds. This is due to the fact that PaaS clouds are more general in terms of the applications that run on them, and therefore must be able to handle a wider range of workloads and configurations. However, from the perspective of a business that has its own data center, it is likely that the applications are set up as PaaS (not SaaS) since the number of applications is limited, and for internal use. In such a case, it may not make sense for the local IT team to set up a SaaS cloud if it is not their main business. For the virtual desktop use case, we focus on SaaS applications in an internal data center.

3 WORKLOAD CHARACTERIZATION

3.1 MICROSERVICES

Microservices an emerging cloud application architectural style, which considers a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms (e.g. containers such as Docker). Microservices are generally designed to be scalable and resilient, which means they can be deployed on a set of distributed servers, under the assumption that a communication channel (such as TCP/IP) is available to coordinate efforts between the various instances and services. We focus our efforts on characterizing the various microservices that compose the sample applications from the VDI use case. We encode this characterization in an application descriptor that can be used by the orchestration software.

3.2 SPLITTING APPLICATIONS

Looking at HPC/HTC applications they appear generally as monolithic, so the large computational resources are exploited by launching a large number of threads and processes in parallel. Conversely, cloud applications are generally not monolithic in nature and thus they are composed by different independent modules. The interaction between such modules allows to provide the final service as intended by the cloud application. Microservices are of help in such approach, since different components providing specific services can be associated to their specific microservice. One way of mapping such microservices with real software entities, is the use of Linux containers to provide flexibility and efficiency. Each container is deployed in the data center infrastructure, and it exposes its service through an API.

It remains, however, responsibility of the application developer to provide the description of such components. To this end, the developer must use an “application descriptor” which is a structured form of providing information to the infrastructure manager on how such components must be launched, how they can be controlled (starting, stopping, monitoring the service), how they can be monitored, and how they can interact with other services. Application descriptor can use a sophisticated structure to provide such information. For instance, the TOSCA standard allows to describe applications as a graph, where each node represents a software component that can depend on other components or infrastructural elements.

4 APPLICATION DESCRIPTORS

With the wide adoption of cloud services and the growing complexity of cloud applications, the need for a way to easily describe and automatically manage such applications becomes necessary. The majority of the tools used to manage cloud infrastructures provide mechanisms to allow such description and automation. However, the main drawback of these formats is that they are not standard, thus they do not guarantee the interoperability of the applications across different cloud providers as well as orchestration tools. Some tentative to address this issue has been done by providing only solution tackling the problem at a higher level rather than at the infrastructure (i.e., at a functional level). A more structured and standardized approach is needed, and the TOSCA format described here and referred also in D5.3 and D5.4 goes in that direction.

The following is the set of the criteria that is used to compare the various existing application descriptors in order to select the most applicable starting point for addition of power awareness:

- Ability to describe an application in terms of multiple components
- Ability to describe relationships between multiple components
- Ease of integration with OpenStack
- Interoperability with multiple infrastructures
- Level of standardization

The following is a brief description and comparison of the most widely used formats. The comparison is to demonstrate why TOSCA was selected as the most fitting model to be used as the basis for modification in the OPERA project. It is of worth noting that although the remaining formats have partially overlapping feature sets with the TOSCA model they do not cover all the features provided by TOSCA. Furthermore, none of these formats are standard. For these reasons, within the OPERA project, we decided to adopt the TOSCA model.

4.1 OPENSTACK HOT

HEAT Orchestration Template (HOT) is the default format for defining templates in HEAT (recent version of OpenStack – April 2014 release and later). The format is however specific for the OpenStack HEAT component. It is based on the YAML language and consists of the following main elements:

- A template version
- Description of the template
- Declaration of input parameter groups and order
- Declaration of input parameters
- Definition of template resources (this is mandatory to allow the template doing something useful)
- Declaration of output parameters

Although the format allows also to describe service composition, there is no explicit way to declare relationships among the resources. Such relationships are mostly declared in the “properties” and “depends_on” subsections of the single resources. Furthermore, the granularity at which these relationships can be described is coarser grained than the one provide by TOSCA (i.e., in TOSCA each relationship has its own type). With respect to TOSCA, it is specifically designed targeting the OpenStack infrastructure, thus it does not guarantee interoperability across different cloud infrastructures.

4.2 USDL

The Unified Service Description Language (USDL) was developed in 2008 for describing business, software, or real world services using machine-readable specifications to make them tradable on the Internet. In 2012, a new version named Linked USDL based on Linked Data principles and RDF was proposed. Linked USDL is segmented in 5 modules. The usdl-core module models general information such as the participants involved during provisioning and service options such as customer support. The cost and pricing plans are modelled with usdl-price. The legal terms and conditions under which services may be consumed are modelled with usdl-legal. The module usdl-sla gathers information on the levels of service provided, e.g., availability, response time, etc. Finally, usdl-sec models security features of a service. The main difference with respect to TOSCA is that USDL only describes applications in terms of components, while it completely lacks a way of describing relationships among elements. For this reason, USDL cannot be directly used to orchestrate application deployments, requiring the integration of additional tools.

4.3 CAMP

Cloud Application Management for Platforms (CAMP) is a specification of the OASIS committee designed to ease management of applications across platforms offered as a service (PaaS). In a similar way as the TOSCA model, it allows to describe how an application relates to the underlying platform service. To this end, the specification provides a RESTful generic self-service application and platform management API, which is language, framework, and platform neutral. This API consists of a resource model and a protocol to remotely manipulate these resources (i.e., Platform, Assemblies, Platform Components and Application Components). Compared with the TOSCA model, it offers only limited capabilities for describing applications in terms of components and how these components must be deployed on the underlying infrastructure.

4.4 CLOUDFORMATION

CloudFormation is an Amazon WebServices (AWS) specific product, aimed at orchestrating application deployment within the Amazon cloud infrastructure. CloudFormation is a declarative, JSON-based language allowing to describe a set of related AWS resources (e.g., EC2, Elastic Load Balancer instances, S3 buckets, etc.) called a stack. This stack can then be managed (i.e., created, updated and deleted) as a single entity. Despite its effectiveness, CloudFormation is not designed having interoperability in mind, being designed specifically designed targeting AWS resources.

4.5 PUPPET AND CHEF

Deployment languages (for example, Puppet and Chef), are primarily designed for managing the application configuration. Although there is some similarity between “managing the application configuration” and “orchestrating”, these are quite different at the end. Puppet and Chef use script-based templates (called respectively Manifests and Recipes) to configure the application based on the underlying infrastructure already deployed. This means that, Puppet and Chef cannot manage infrastructural resources, such as VM instances, volumes, floating IPs, etc. From this viewpoint TOSCA allows to describe the applications and the way each of their element must be deployed. To this end, and through “Plans”, TOSCA allows us to interact directly with infrastructural resources, and (when required) to configure such resources.

4.6 OTHER TOOLS

Other tools exist for managing orchestration [1]. Among the other, we consider for our analysis the following ones: Apache Brooklyn, Cloudify, and Apache Stratos. Apache Brooklyn provides a template format based on YAML, which is compliant with the CAMP specification, while support for TOSCA is planned as a future feature. Cloudify provides its own Domain Specific Language (DSL) based on a YAML format for the managing the application deployment. This DSL is strongly aligned with the TOSCA modelling standard, although it does not directly reference the OASIS TOSCA standard. Full TOSCA support is planned as a future feature. Apache Stratos utilises jclouds as a cloud abstraction layer, supporting multiple providers and it is not fully compliant with the TOSCA model.

4.7 TOSCA

TOSCA [2] is a standard created by the OASIS committee to describe application and cloud infrastructural elements in a portable form. The standard provides a way of creating a template that describe all the required elements needed to correctly deploying the application. It originally supports template description as XML files, but a YAML-based version is under validation. A TOSCA template can be automatically translated to the native Heat format (i.e., HOT format) through a dedicated translation tool. OpenStack provides such tools as part of the Heat module installation.

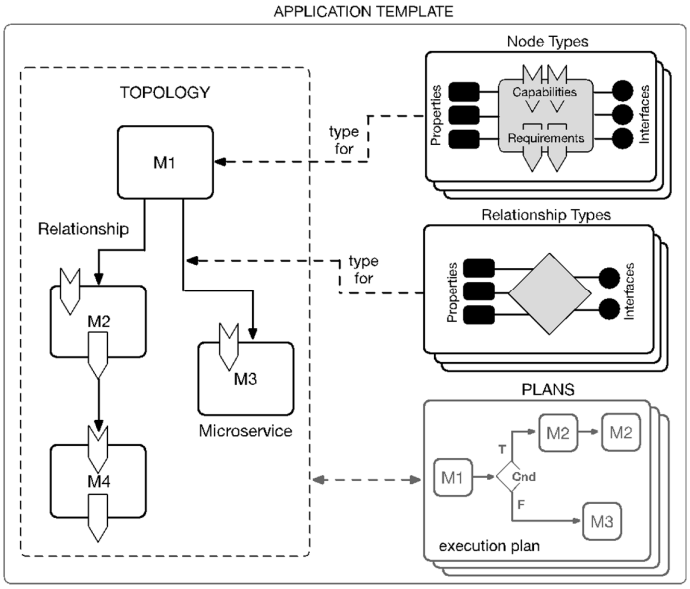

Figure 1 - TOSCA Template Model

Figure 1 shows a generic template that uses a graph to map application components and their relationships. TOSCA provides a description language to define a service template (ST), which in turn includes a topology template (TT), and a set of Plans (P) to orchestrate the execution of the application. The language is essentially based on the creation of a scheme, where components are placed along with their connections with other components. From this perspective, the language resembles the Block Diagram schemes which can be used to describe a generic algorithm. The TT is represented by a set of nodes. Nodes allow to hierarchically describe each component up to the lowest level (i.e., the VM or container that host the component), while edges provide the relationships among the components (each edge has a type describing the specific relationship, e.g., hosted on, etc.). Nodes expose capabilities (essentially features that can be used by other nodes to match with their requirements), interface (a set of functions that can be used to interact with the node, e.g., a create function that allow to create a database, etc.), and requirements (a set of features that must be satisfied in order to correctly work).

Each node has its own type, as well as the relationships among the nodes (also relationships have their types). Actually, each node can be an installable package and/or an image (e.g., a VM image, a container image, etc.). Each node exposes a set of capabilities (C) representing the set of functionalities offered by the node, and a set of requirements (R) representing the specific needs (e.g., a Web Application may require an application server in order to be run). Nodes have also properties and interfaces: the former provide a description of the functionalities offered by the node-service, the latter generally concern with scripts that are used to manage the node operations. Similarly, relationships may expose properties and interfaces. A relationship indicates which nodes can satisfy the requirements of another node, by means of its capability. From this viewpoint a TT defines a mapping between the C of a node and the R of another. A plan specifies how the nodes interact each other in order to provide the whole service represented by the ST. All these elements are packaged together in a single entity called Cloud Service ARchive (CSAR). Requirements for a type can be satisfied also by the underlying hardware infrastructure that can be considered as a specific node type.

A set of scripts can be used to actually perform specific operation for the nodes. TOSCA has a set of standard node types, but customized nodes can be created. A plan (i.e., a workflow) can be also added to describe the correct order of deploying nodes.

5 INTEGRATION WITH OPENSTACK

OpenStack is one of the major cloud infrastructure software stacks, that allows to control compute, network and storage resources of a data center. OpenStack is an open source project, and thanks to its flexibility it has been selected in the OPERA project to support the deployment of cloud application. It is also extensible, since more components can be developed and integrated in the platform.

OpenStack provides several components to manage different aspects of the data center infrastructure. One of the components is HEAT, which is in charge of orchestrating multiple composite cloud applications by using either the native HOT template format or the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.

Integration of the TOSCA model can be easily achieved by adding a transformation layer that becomes responsible for transforming TOSCA XML description into one of the format natively supported by HEAT. There are some projects targeting the implementation of a TOSCA-to-HOT translator, as well as the implementation of a standalone TOSCA parser. Among the others, there are:

- Heat-Translator [3]: a command line tool (released under Apache 2 license) which takes non-HEAT templates (currently it supports TOSCA) as an input and produces a HEAT Orchestration Template (HOT) which can be deployed by HEAT component. There is support for the CSAR format.
- TOSCA-parser [4]: a standalone parser for the TOSCA model (TOSCA YAML or CSAR format), which provides the set of nodes and relationships as in-memory objects. It also creates a graph of TOSCA node templates and their relationship.

The integration with OpenStack through the suggested modifications/extensions of the TOSCA format is necessary to have a fully operative orchestration toolchain. Such toolchain is deeply discussed in deliverable D5.3, as results of the activity of task T5.3 regarding the integration of OpenStack modules with our energy-aware scheduler (see D5.4). Specifically, the information provided in the TOSCA file are used to query a Knowledge Base (KB) managed through a structured database. The structure of this database is well described in deliverable D5.3, while the algorithmic solution used to take decisions on how to allocate application components is provided in deliverable D5.4 as part of the task T5.4.

6 USE CASE WORKLOADS

The VDI use case specifies several software packages that should be used in an office environment. The software packages that can be implemented as SaaS services interest us the most, since these are the cases that provide the most opportunity for optimizations, as discussed in section 2. We have selected 3 of the software packages for deeper investigation in WP5.

6.1 OPENXCHANGE

This section describes the architecture of the Open Xchange application, one of selected software packages that will be used to verify the code developed for OPERA Project. Open Xchange is a web-based open source email, communication and collaboration suite. This suite can be set up as a service according the schema reported in the following figure, where we have four different elements. As shown in Figure 2, the first element hosts both an Apache web server to dispatch incoming connections and serve static resources, and the application layer of Open Xchange based on OSGI platform. The other ones are the basic applications to guarantee the service: a MySQL database, a Postfix SMTP server and a Dovecot IMAP server.

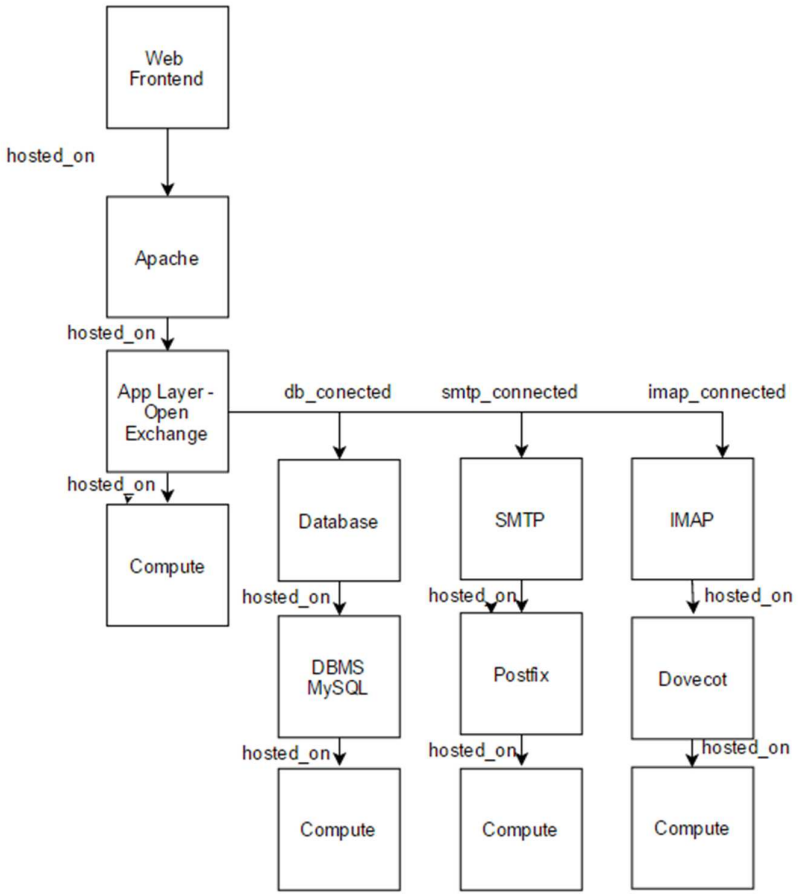

Figure 2 - OpenXchange Architecture Diagram

6.2 OWNCLOUD

In this subsection we provide a description of the OwnCloud application, which is used in OPERA as a vehicle to test the technologies, algorithm and tools developed. The purpose of OwnCloud is to provide a storage service for files and documents. OwnCloud is composed of two main modules in a basic installation: a web server which provides the front-end for managing the application, and a database to store and manage data stored on the platform. Figure 3 shows the topology of the application, with a generic indication for the hosting nodes for the web front-end and the database.

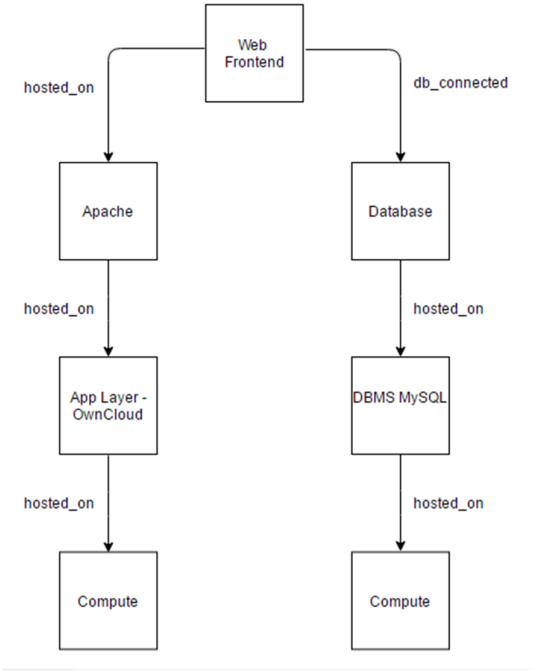

Figure 3 - OwnCloud Architecture Diagram

6.3 LIBREOFFICE

In this paragraph we describe the architecture of the LibreOffice application, the last one of selected software to verify the code developed for OPERA Project. LibreOffice is a suite of Office Automation Applications (Calc, Write, Print, etc.). It is possible to set up LibreOffice as a service, starting from OwnCloud as an additional module. For this reason the schema (reported in the following Figure) is very similar to the previous one, in fact the supplemental element is the Application Layer of LibreOffice, as shown in Figure 4.

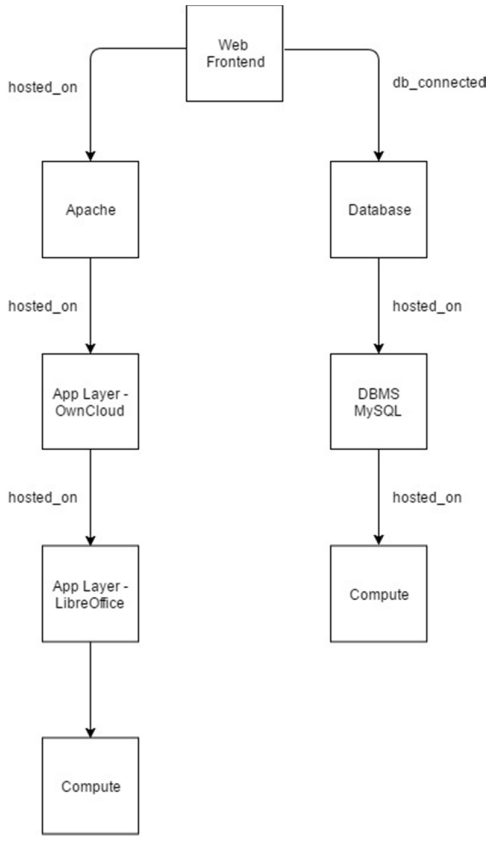

Figure 4 - LibreOffice Architecture Diagram

7 CONCLUSIONS

We have made a comparison study of several competing application descriptors to determine the best one to be used as the basis in OPERA. We have determined the changes that need to be made to the application descriptor in order to make it power-aware. We have investigated how to integrate into OpenStack. We have studied the best way to decompose our sample applications into microservices. These decomposed applications will be described by the enhanced TOSCA descriptor.

8 REFERENCES

[1] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann and J. Wettinger, "Combining Declarative and Imperative Cloud Application Provisioning Based on TOSCA," *2014 IEEE International Conference on Cloud Engineering,* pp. 87-96, 2014.

[2] OASIS, "Topology and Orchestration Specification for Cloud Applications Version 1.0," May 2013. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

[3] "OpenStack HEAT Translator," 2017. [Online]. Available: https://github.com/openstack/heat-translator .

[4] "OpenStack TOSCA Parser," GitHub, [Online]. Available: https://github.com/openstack/tosca-parser . [Accessed 2017].