Enabling Cloud Application Portability

Demetris Antoniades, Nicholas Loulloudes,
Athanasios Foudoulis, Chrystalla Sophokleous
Demetris Trihinas, George Pallis, Marios Dikaiakos
Department of Computer Science
University of Cyprus
Email: [danton,loulloudes.n,athafoud,
stalosof,trihinas,gpallis,mdd]@cs.ucy.ac.cy

Harald Kornmayer
Institut fur Informatik
Duale Hochschule Baden-Wuerttemberg
Mannheim, Germany
Email: harald.kornmayer@dhbw-mannheim.de

***Abstract*—The *Cloud Application Management Framework (CAMF)* enables Cloud application developers to design, deploy and manage their applications through an intuitive *blueprint* design. In this paper we show how Cloud application developers can utilize CAMF in order to have portable applications that can be deployed on different IaaS with minimal effort. Towards this goal, we introduce the *Cloud Application Requirement Language (CARL)*. CARL can be used for defining the application software and hardware requirements, information that is then included into the TOSCA description of the Cloud application, alongside the application blueprint. *CAMF's Information Service* utilizes both these artifacts to provide IaaS specific configurations that fulfill the user's requirements.**

I. Introduction

Cloud computing is in a constant advancing mode nowadays. It is the de facto deployment platform for Web applications (both desktop and mobile) with increasing adoption trends in both corporate and scientific environments. It eliminates the need for owning any computing infrastructure at all, thus cutting-down significantly operational expenses and management efforts for applications of any scale.

However, in the current cloud computing landscape, application developers are usually locked-in to a single cloud provider. Moving an application as a whole, or even partially, from one provider to another is a *challenge* since it entails significant (re)configuration efforts to comply with the technologies of the underlying platform. During this process the developer has to first identify and then provision those resources (machine images, flavor, network interfaces, storage media, etc.) that accommodate best the application's deployment requirements. Moreover, runtime requirements such as software dependencies (system libraries and tools, runtime archives, etc.) need to be satisfied and made available to the newly provisioned resources.

In this paper we demonstrate how one can utilize the Cloud Application Management Framework (CAMF) [8] in order to automate and streamline *application portability* across different Infrastructure as a Service (IaaS) providers with minimal effort. CAMF, uses OASIS TOSCA [2], the Technology and Orchestration Specification for Cloud Applications, and enables application developers to provide a generic blueprint of their application without any consideration in the technological specifications of the target platform. Application requirements (hardware and/or software) are described in the form of simple yet expressive directives using the newly introduced *Cloud Application Requirement Language (CARL)*. The set of such directives is then processed by *CAMF's Information System Service (CAMF-ISS)*, an intelligent online matching service that ultimately results with an IaaS-specific application description for the target platform. In that respect, the cloud-agnostic blueprint describes all the necessary application requirements (virtualized resources, os, libraries, etc.), the application and its components need in order to be deployed, like OS and server version. Additionally, the developer can add custom configuration scripts or application binaries that each of the modules need. Having this information, CAMF, through CAMF-ISS, provides all the matching instances and flavors a specific Cloud provider makes available that match the generic description. The developer can then select the most appropriate ones and deploy her applications.

Moving or trying out a different cloud provider can prove quite beneficial in terms of application cost, performance or both. Through *CAMF-ISS*, CAMF can provide all the matching instances and flavors for the new cloud provider, while transparently transferring all the user customized configurations and binaries [1]. This process limits down the developer actions for moving her application to a new provider to only a few clicks.

Additionally, CAMF comes with a number of integrated modules that enable the cloud developer to monitor and analyze her application performance and cost over any provider, pinpointing in this way the best deployment option. JCatascopia, CAMF's integrated interoperable monitoring system, collects measurements for a number of system and application metrics. *CAMF-ISS* collects all the monitoring information from JCatascopia and enables application topology specific analysis. Using the generic blueprint of the application *CAMF-ISS* shows both generic and per module performance and cost analysis, correlating also the behavior and trend of different modules. In addition, *CAMF-ISS* can be used for comparisons between different application version and deployments, within and among cloud providers, thereby giving the application developer insights towards a more optimized configuration for her application. The contributions of this study are:

1) We enrich the TOSCA cloud application description with the user requirements by introducing CARL, the *Cloud Application Requirement Language*
2) We introduce *CAMF's Information System Service*, an

[1]In spite of its significant importance, the scope of this article is not (live) application migration across providers.

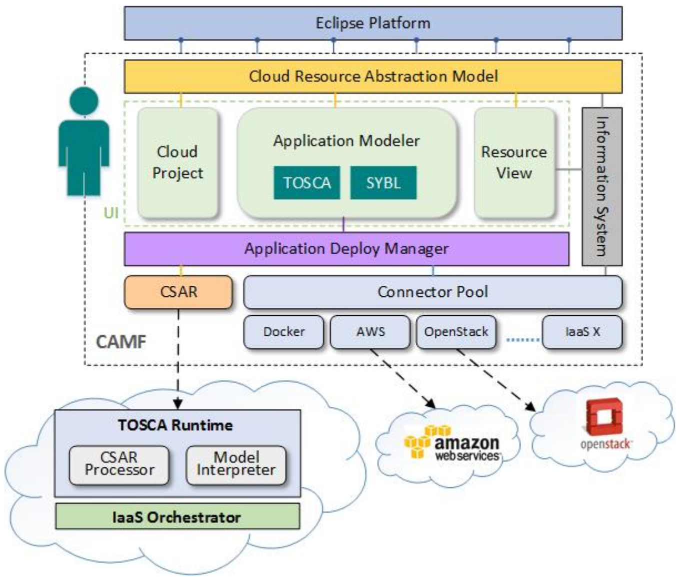


Fig. 1. CAMF's extensible architecture

online system that periodically indexes IaaS hardware and software resources and matches them to the developer requirements, transforming a generic application blueprint to an IaaS specific configuration.

3) Through three heterogeneous applications, we demonstrate how CAMF enables Cloud application portability.

The rest of this article is organized as follows: Section II provides an overview of CAMF's Architecture and discusses the different functional components. Section III discusses CAMF's portability enhancing qualities. In Section IV we introduce and describe three heterogeneous applications through which we demonstrate the portability functionality of CAMF during the challenge. Section V presents the different stages in the application portability process. Finally, Section VI presents the Related Work, and Section VII provides conclusive remarks about our work.

II. CAMF

The Cloud Application Management Framework (CAMF) [2] is a newly established open source technology project under the Eclipse Foundation. CAMF aims to provide extensible graphical tools over the Eclipse Rich Client Platform (RCP) that facilitate lifecycle management operations for Cloud applications, in a vendor neutral manner. To this end, CAMF focuses on three core operations:

Application Description
interoperable representation of cloud application's structure (blueprints) with high-level depiction of service components, management operations, and interrelationships.

Application Deployment
preparation and submission of descriptions to any cloud infrastructure of preference, while enabling seamless and repeatable contextualization workflows.

Application Monitoring
realtime data from the underlying platform and the application itself for fine-grained operational and performance monitoring.

The Eclipse platform provides a strong foundation for CAMF primarily because it retains a dominant position in the worldwide IDE market [3]. Its OSGi plug-in-based software architecture enables tight integration of language-specific IDEs with a plethora of supporting suits providing code testing, analysis and profiling, code management ans so on. Through CAMF, an IT expert can empower her day-to-day development processes with cloud application management operations. Eclipse will not only function as a Web development environment, but also facilitate the preparation, migration, and monitoring of her applications to the cloud. Furthermore, build-in collaboration support can assist when participating in large development teams wherein group members can effortlessly share application description, deployment information artifacts or even complete cloud project with just a few clicks. Notable, all these activities can take place withing a unified and intuitive graphical workspace.

Figure 1 depicts the extensible architecture of CAMF. We briefly describe the most important components of CAMF, in regard to application portability in the next paragraphs. For more information about the complete architecture we refer the interested reader to [15].

A. Application Description

1) Generic description: Through CAMF's graphical environment the application developer's first task it to provide a generic description of her application. These *topology blueprints* specify a different structure and management operations for the application at hand. However, each blueprint is generic in that it does not contain any IaaS-specific information. In this basic form, application descriptions are portable templates, which upon IaaS selection can be enriched with particular metadata that materialize a component or operation(s) for the target IaaS.

2) Requirement description: After describing the generic blueprint of her application, the cloud application developer moves on into describing the application requirements. CAMF introduces CARL, the Cloud Application Requirement Language, that enables the programmer to give abstract descriptions of the application requirements. CAMF exports the generic (topology and custom software) and CARL requirements into a TOSCA [2] specification file. This file is then send to CAMF-ISS, for it to provide IaaS specific configurations, that better match the requirements.

a) CARL: The Cloud Application Requirement Language (CARL) aims at facilitating users to specify the requirements (hardware and software) of their applications in a generic and provider-agnostic manner. It includes a set of simple yet expressive directives that enable a cloud developer to define the requirements of its application at different levels of the virtualization stack. As depicted by the EBNF representation in Grammar 1, CARL's grammar supports three top-level application requirements, namely *System, Operating*

[2]http://www.eclipse.org/camf

[3]http://goo.gl/m0G2u3

```
sys:cpu=[2-4];  sys:mem>=8G;
sys:disk>20G;  sys:net=1000M;

os="Ubuntu";  os:arch="x86_64";

sw="java":ver>1.6;  sw="tomcat":ver>=6.0;
```

Listing 1.   Cloud Application Requirments Example

System and Software. Listing 1 provides an example of requirements specified via CARL.

At the System level, the user can define application requirements in terms of *cpu, storage, memory and network* and assign value constraints through standard arithmetic operators. At the same time the user can further define units of measurement for some constraints, for example Kilo (K), Mega (M), Giga (G) or Tera(T).

⟨*requirements*⟩ ::= ⟨*reqs_list*⟩

⟨*reqs_list*⟩ ::= {⟨*req*⟩}

⟨*req*⟩ ::= ⟨*sys_reqs*⟩ | ⟨*os_reqs*⟩ | ⟨*sw_reqs*⟩

⟨*sw_version*⟩ ::= ⟨*INT*⟩ [('.' ⟨*INT*⟩)][('.' ⟨*INT*⟩)][('.' ⟨*INT*⟩)]

⟨*sys_reqs*⟩ ::= {(⟨*sys_field*⟩ ⟨*DELIMIT*⟩)}

⟨*sys_field*⟩ ::= ⟨*sys*⟩ : ⟨*sys_type*⟩

⟨*sys_type*⟩ ::= ⟨*cpu*⟩ | ⟨*disk*⟩ | ⟨*mem*⟩ | ⟨*net*⟩

⟨*cpu*⟩ ::= `cpu` ⟨*operator*⟩ (value | ⟨*range*⟩)

⟨*disk*⟩ ::= `disk` ⟨*operator*⟩ (value | ⟨*range*⟩) UNITS

⟨*mem*⟩ ::= `mem` ⟨*operator*⟩ (value | ⟨*range*⟩) UNITS

⟨*net*⟩ ::= `net` ⟨*operator*⟩ (value | ⟨*range*⟩) UNITS

⟨*os_reqs*⟩ ::= (⟨*os_field*⟩ ⟨*DELIMIT*⟩)

⟨*os_field*⟩ ::= ((`os` = STRING) | (`os` : ⟨*os_type*⟩))

⟨*os_type*⟩ ::= ⟨*ver*⟩ | ⟨*arch*⟩

⟨*sw_reqs*⟩ ::= (⟨*sw_field*⟩ ⟨*DELIMIT*⟩)

⟨*sw_field*⟩ ::= ⟨*SW*⟩ = ⟨*sw_type*⟩

⟨*sw_type*⟩ ::= ⟨*STRING*⟩ (= ⟨*sw_version*⟩)

⟨*ver*⟩ ::= `ver` ⟨*operator*⟩ ⟨*sw_version*⟩

⟨*arch*⟩ ::= `arch` = ⟨*STRING*⟩

⟨*SYS*⟩ ::= `sys`

⟨*OS*⟩ ::= `os`

⟨*SW*⟩ ::= `sw`

Grammar 1.   CARL grammar in EBNF notation

B. CAMF's Information System Service

CAMF's Information System (Fig. 1), is the component responsible for the communication with CAMF's Information System Service (generic and specific TOSCA file exchange). CAMF-ISS handles both the resources each IaaS makes available to the developer and also keeps up-to date information on how the resources of each application are utilized. The internal architecture of CAMF-ISS is depicted in Figure 2.

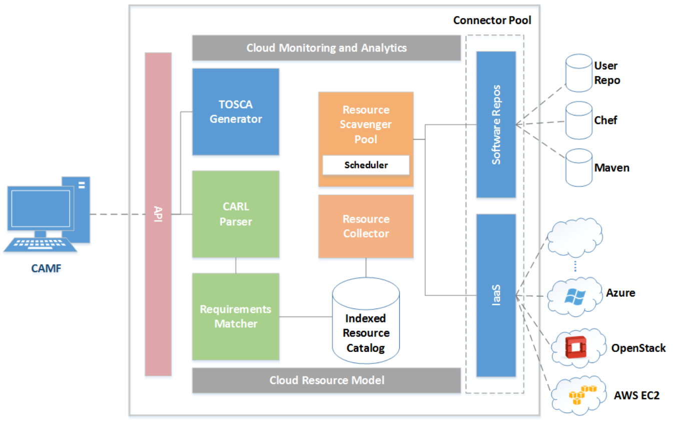


Fig. 2.   CAMF's Information System Service (ISS)

1) Resources: CAMF-ISS communicates via IaaS specific connectors, accessed through a connector pool interface, and retrieves information for both virtualized hardware and any available software resources. All these remote resources, (i.e. virtual images, networks, storage media, key pairs and so on), available by one or more IaaS, are indexed and saved to the Indexed Resource Catalog (IRC) using a Cloud Resource Model representation.

a) Hardware: CAMF-ISS indexes virtualized hardware resources as those are made available in the involved IaaS "marketplaces". The needed resource metadata, as name, type, description or relevant task published by the provider are stored by IRC and periodically refreshed.

b) Software: Collecting software resources from the IaaS provided instances is a more challenging task. Most providers do not directly disclose the already installed resources through the desciptive metadata that usually accompany a VM image. Therefore, the user has to manually check if any resources are available and if necessary download the ones she requests for her application. To provide a more complete view of the available resources to the user and also enable automatic application portability, CAMF-ISS incorporates an *IaaS Resource Scavenger* pool. Each *scavenger* communicates with an IaaS and connects to each available instance in order to retrieve a list of the available software resources (as those are listed in apt and/or yam repositories). The *scavenger* runs in a similar manner as Minersoft [4]. Specifically, it periodically initiates all the newly available instances from each IaaS and harvest the software files available on them. It then provides CAMF-ISS with an enriched software list that includes all the available software components.

2) Analytics: CAMF-ISS is capable of providing, per Cloud application, usage analytics by processing information regarding resource utilization and cost-enriched monitoring data, collected during the lifecycle of an (elastic) cloud service deployment. Moreover, it enables application users to query, explore, compare, and visualize historic data collected from past deployments and previous versions of a cloud service.

a) Resource monitoring: Currently CAMF is fully integrated with JCatascopia [16], an automated, multilayer interoperable monitoring system for elastic cloud environments. JCatascopia follows an agent-based approach, and can be utilized to collect monitoring metrics from multiple layers of the underlying infrastructure, as well as performance metrics from the deployed cloud services. JCatascopia is platform independent and during the metric collection process it takes into consideration the rapid changes that occur due to the enforcement of elasticity actions on the application execution environment and the cloud infrastructure. JCatascopia comes with a standard probe suite that allows the user to monitor basic metrics. CAMF allows application developers to also define and implement custom metrics collectors that adhere to JCatasopia's modular architecture. These probes can be placed anywhere on the application stack, ranging from the virtualization layer, upwards to the application itself, to obtain meaningful metrics that report the runtime health and application performance.

C. Requirement matching

When CAMF's Information System component sends the requirement enhanced TOSCA file, the CARL parser, in CAMF-ISS, extracts the developer requirements and transmits them to the Requirement Matcher. This module runs a keyword based search against the Indexed Resource Catalog, to find the components (software and hardware) that better match the application requirements. When done, the information is passed to the TOSCA Generator to create the different IaaS specific configurations and return them to CAMF. The configurations are now ready for deployment.

D. Application Deployment

The enhanced TOSCA file, including IaaS specific configurations, returned by CAMF-ISS, can now be used for application deployment. At this point, the TOSCA file is translated by the IaaS specific connector to IaaS API calls and deployed to the provider infrastructure.

III. Portability Qualities

To hide any inherent complexity and support cloud portability for any of the three management operations, listed in the previous section, CAMF relies on abstract models for interacting with cloud resources as well as various open source specifications and toolkits.

A. IaaS Cloud Abstractions

Building on top of facilities provided by the *Apache jclouds* toolkit [4], the cloud resource model lays the groundwork towards portability by establishing a common way of interfacing with different cloud vendors. In particular, it defines a common set of abstraction for virtualized resources (server images, network, security, monitor probes, and elasticity policies and actions) and artifacts (executables, libraries, and configurations) that are key to any applications deployment, irrespective of the underlying cloud environment. In addition, it defines a common set of actions on these abstractions (authentication, resource listing, monitor metric listing, configuration, and deployment) that are necessary to complete a management operation. This layer of abstraction is responsible for providing basic interfaces and classes that may be extended by IaaS-specific connectors, enabling CAMF to satisfy application transition in any cloud vendor.

[4]http://jclouds.apache.org

B. Interoperable Application Descriptions

CAMF adopts the Organization for the Advancement of Structured Information Standards (OASIS) Topology and Orchestration Specification for Cloud Applications (TOSCA) specification for blueprinting and packaging cloud applications in a standardized manner. TOSCA provides a vendor-neutral language for describing the topology of cloud applications, along with their management operations. TOSCA adopts a graph-based topology representation, which includes hardware (virtual) and software components involved and their interrelationships. Components implemented by means of *nodes* signify executing entities in a deployment with semantics such as *requirements* against the hosting environment, *capabilities*, and *policies* that govern the execution aspects like resource security or elasticity.

C. Automatic and Streamlined Application Configuration

TOSCA provides the necessary grammar to describe management aspects of an application by means of lifecycle operation or by more complex management plans. For each node, we can explicitly define its lifecycle operations (for example, deploy, configure, or start an instance of the corresponding types). Currently, CAMF supports lifecycle operation and particularly node configuration through native shell scripts. In the near future, we'll incorporate new tools in the framework that allow developers to import third party, or compile their own, open source Chef cookbooks that automate and streamline application configuration processes.

D. Vendor-Neutral Elasticity Specification

Granted that resource autoscaling is widely regarded as one of the key ingredients of future Web applications, CAMF supports the Simple-Yet-Beautiful Language (SYBL) [3], an open source, directive-based, elasticity requirements specification. SYBL enables developers to define fine-grained scaling policies and corresponding enforcement actions, by leveraging monitoring metrics placed at different levels of the application stack. Metrics can be obtained either from native IaaS monitoring systems or from custom metric collectors (probes) provided by the user during deployment. SYBL elasticity policies are mapped to TOSCA node policies and provide two variances: *Constrains* and *Strategies*. The former express the constraints of an application related to cost, performances, and other quality metrics and allows the underlying infrastructure to decide the appropriate scaling action to be taken, if such intelligence is available. The later variance lets developers take full control of elasticity management by specifying explicit scaling actions.

IV. Demo Applications

In this section we describe the three heterogeneous applications that we will use to demonstrate the portability functionality of CAMF during the challenge.

A. Three tier video application

We start with an intuitive three-tier online video streaming service [17]. Such an application combines three basic components every Web application needs. The service is comprised of: (*i*) an HAProxy [5] *load balanced* which distributes requests (i.e. download, upload video) across multiple application servers. (*ii*) An *application server tier*, where each instance is an Apache Tomcat [6] server containing the video streaming web service. Aggregated tier metrics such as the average number of connections and/or request throughput can be used to evaluate the application performance. (*iii*) a Cassandra [7] NoSQL *distributed data storage backend*. Similarly, aggregated metrics such as the average CPU utilization and/or query latency can be used to evaluate the Cassandra ring performance.

B. SCAN

As a second application we use SCAN [14] a framework for running batch scientific workloads in a cloud environment. SCAN enables executing biological applications into elastic cloud environments. In this version of SCAN we run the Genome Analysis Toolkit (GATK) [10], a collection of tools for the analysis of genetic sequencing information, with a particular focus on discovering differences between the genetic samples taken from a particular individual and the "standard" reference genome. SCAN provides one or more work queues and manages a pool of worker machines per queue, dispatching tasks to the workers as they become available. Tasks may come from users, or be generated by a previous task.

C. ScienStore

The third application is a scientific workbench and market place for scientific applications, called "ScienStore", that enables scientist to concentrate on their scientific questions instead of dealing with the details of their IT system. [6] ScienStore is build as an OSGi component, enabling the delivery of software on the scientist Cloud resources.

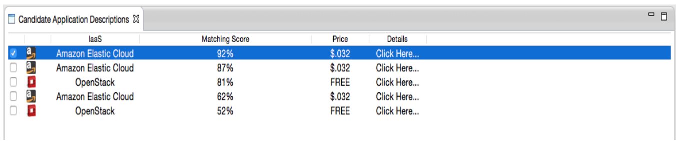
V. Portability Process

This section describes the process a Cloud application developer follows to describe and deploy her application using CAMF. We showcase the process using the ScienStore application, since it's simpler and more intuitive. However, the process is identical for any cloud application. The challenge demo will also showcase the process for the other two applications.

A. Generic Application Description

As described in the previous section the ScienStore application is composed by one basic component. To better illustrate the use of CAMF, we split this component into to two different components based on their resource demands. The first component can be used for memory intense calculations while the second one can be used for CPU demanding calculations. The first step for the application developer is to graphically describe these components using CAMF. For each component the developer includes any artifacts that should be included

[5]http://haproxy.lwt.eu
[6]http://tomcat.apache.org
[7]http://cassandra.apache.org



		IaaS	Matching Score	Price	Details
✓		Amazon Elastic Cloud	92%	$.032	Click Here...
☐		Amazon Elastic Cloud	87%	$.032	Click Here...
☐		OpenStack	81%	FREE	Click Here...
☐		Amazon Elastic Cloud	62%	$.032	Click Here...
☐		OpenStack	52%	FREE	Click Here...

Fig. 4. ScienStore proposed configurations ready for application development

with her applications. These artifacts include any custom source code, configuration scripts, executable files and data files the developer wants to include in her project. In the case of ScienStore we include the OSGI bundle that implements the scientific application offered by the soter.

B. Requirement specification

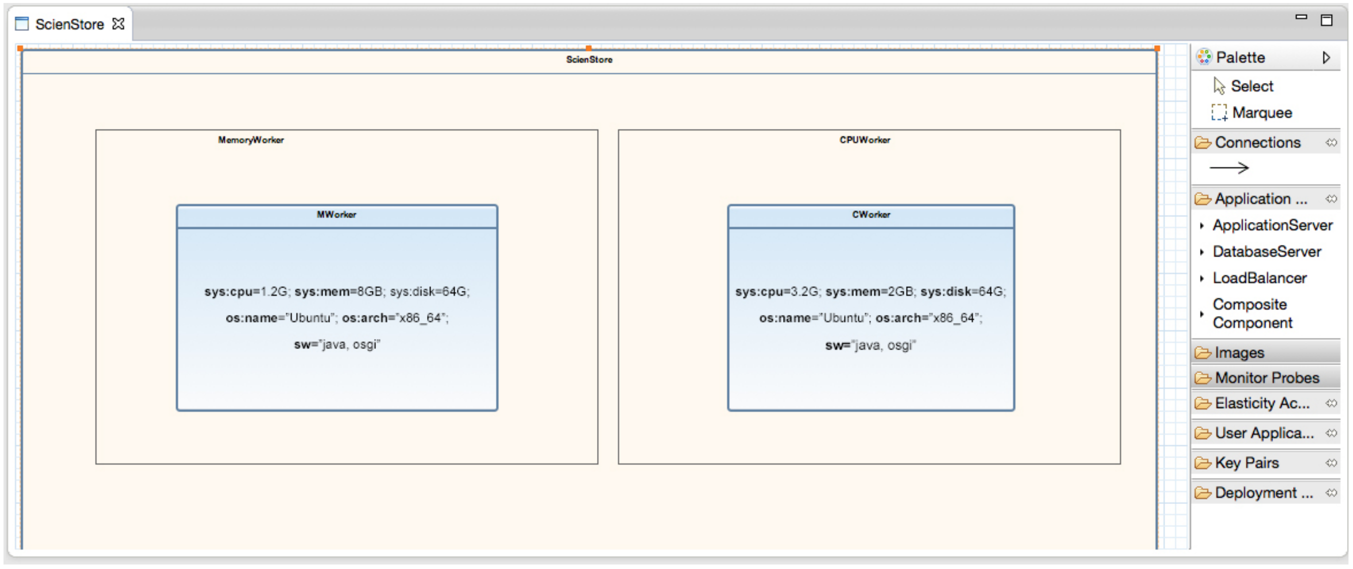
The second step is the description of the software and hardware requirements for the application to be deployed. Through an intuitive graphical environment, the cloud developer can provide the requirements for each specific component. Figure 3 shows how the generic description of our sample application looks like. The component on the left, represents the memory intense component. For this, the application developer requires 8GB of memory. The right component is configured for more CPU intensive calculations. For this reason the developer requires a 3.2GHz CPU. All other requirements are the same and include OS selection and installed software requirements.

C. Provider Selection

When the developer has completed the above steps, the generic configuration and requirements is send to CAMF-ISS, as a TOSCA description, through CAMF's Information System component. After the matching procedure, CAMF-ISS, returns the IaaS specific descriptions that are presented to the application developer through a table, as shown in Figure 4. The table includes a matching score, denoting the amount of requirement with an exact match. Also it provides an estimate of cost per hour of usage for the components used (inferred by the IaaS pricing scheme), and the option to get more details on the configuration. The selected deployment is ready to be deployed to the IaaS, an one-click step performed through CAMF.

VI. Related Work

Portability, along interoperability, are considered two of the major challenges in cloud computing today, topped only by trust and security issues. Petcu [11] gives a thorough review of portability and interoperability challenges. Petcu et al. proposed mOSAIC an open-source platform based on a stack of existing Cloud technologies that allows the development of Cloud application on top of Private of Public Cloud resources [12]. The PaaS Semantic Interoperability Framework (PSIF), aims at modeling semantic interoperability conflicts that may occur during the migration of an application on a cloud platform [9]. CSAL provides a storage abstraction layer to enable applications to both utilize the highly-available and scalable storage services provided by cloud vendors and be portable across platforms [5]. Ranabahu et al [13] present an abstraction-driven approach, where cloud service consumers use abstract languages to specify their programs. A software infrastructure transforms the user program specifications to



sys:cpu=1.2G; sys:mem=8GB; sys:disk=64G;
os:name="Ubuntu"; os:arch="x86_64";
sw="java, osgi"

sys:cpu=3.2G; sys:mem=2GB; sys:disk=64G;
os:name="Ubuntu"; os:arch="x86_64";
sw="java, osgi"

Fig. 3. ScienStore generic application description including application requirements

the required, provider specific software components. Kostoska et al. proposes a PaaS platform solution that acts as an intermediate between SaaS applications and services and the IaaS [7]. Their solution relies on TOSCA for the application description and installation requirements. CAMF's approach uses TOSCA [1], the de facto standard for cloud portability, to describe the application and resources. Additionally, our approach provides detailed indexing of the available hardware and software resources of the Cloud provider, and through CARL, the requirement specification language, enables developers to describe and move their applications in different IaaS, with the minimal effort required.

VII. Conclusions

Through three heterogeneous applications we demonstrate how CAMF, a cloud application monitoring framework, enables Cloud application portability. We introduce CARL, a requirement description language that enables Cloud application developers to generically, non-IaaS specific, describe the hardware and software requirements of their applications. CAMF's Information System Service uses these generic descriptions to provide, ready to deploy, IaaS specific configurations.

VIII. Acknowledgments

This work was supported by an AWS in Education Research Grant award and the European Commission in terms of the CELAR 317790 FP7 project (FP7-ICT-2011-8).

References

[1] T. Binz, G. Breiter, F. Leyman, and T. Spatzier. Portable cloud services using tosca. *IEEE Internet Computing*, (3):80–85, 2012.

[2] T. T. Committee et al. Topology and orchestration specification for cloud applications (tosca)–committee specification 01. Technical report, Technischer Bericht, OASIS, 2013. URL http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.pdf. Abgerufen am 2013-04-26.

[3] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar. Sybl: An extensible language for controlling elasticity in cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 112–119. IEEE, 2013.

[4] M. D. Dikaiakos, A. Katsifodimos, and G. Pallis. Minersoft: Software retrieval in grid and cloud computing infrastructures. *ACM Transactions on Internet Technology*, 12(1), 2012.

[5] Z. Hill and M. Humphrey. Csal: A cloud storage abstraction layer to enable portable cloud applications. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 504–511. IEEE, 2010.

[6] H. Kornmayer. Towards the scientific workbench and sciencestore for astroparticle physics. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 540–544. IEEE, 2012.

[7] M. Kostoska, M. Gusev, and S. Ristov. A new cloud services portability platform. *Procedia Engineering*, 69:1268–1275, 2014.

[8] N. Loulloudes, C. Sofokleous, D. Trihinas, M. Dikaiakos, and G. Pallis. Enabling interoperable cloud application management through an open source ecosystem. *Internet Computing, IEEE*, 19(3):54–59, May 2015.

[9] N. Loutas, E. Kamateri, and K. Tarabanis. A semantic interoperability framework for cloud platform as a service. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 280–287. IEEE, 2011.

[10] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, et al. The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome research*, 20(9):1297–1303, 2010.

[11] D. Petcu. Portability and interoperability between clouds: Challenges and case study. In *Proceedings of the 4th European Conference on Towards a Service-based Internet*, ServiceWave'11, pages 62–74, Berlin, Heidelberg, 2011. Springer-Verlag.

[12] D. Petcu, G. Macariu, S. Panica, and C. Crciun. Portable cloud applications-from theory to practice. *Future Gener. Comput. Syst.*, 29(6):1417–1430, Aug. 2013.

[13] A. Ranabahu, E. M. Maximilien, A. Sheth, and K. Thirunarayan. Application portability in cloud computing: An abstraction driven perspective. 2013.

[14] C. Smowton, C. Miller, W. Xing, A. Balla, D. Antoniades, G. Pallis, and M. D. Dikaiakos. Analysing Cancer Genomics in the Elastic Cloud. In *Workshop on Clusters, Clouds and Grids for Life Sciences In conjunction with CCGrid 2015 - 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015.

[15] C. Sofokleous, N. Loulloudes, D. Trihinas, G. Pallis, and M. Dikaiakos. c-Eclipse: An Open-Source Management Framework for Cloud Applications. EuroPar 2014, 2014.

[16] D. Trihinas, G. Pallis, and M. D. Dikaiakos. JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud . In *14th IEEE/ACM In-*

ternational Symposium on Cluster, Cloud and Grid Computing, CCGRID 2014, 2014.

[17] D. Trihinas, C. Sofokleous, N. Loulloudes, A. Foudoulis, G. Pallis, and M. D. Dikaiakos. Managing and Monitoring Elastic Cloud Applications [Demo Paper]. In *14th International Conference on Web Engineering*, ICWE 2014, 2014. Demo Paper.