c-Eclipse: An Open-Source Management Framework for Cloud Applications

Chrystalla Sofokleous, Nicholas Loulloudes, Demetris Trihinas,
George Pallis, and Marios D. Dikaiakos

Department of Computer Science, University of Cyprus, Nicosia, CY1678, Cyprus
{stalosof,loulloudes.n,trihinas,gpallis,mdd}@cs.ucy.ac.cy

Abstract. Cloud application portability and optimal resource allocation are of great importance in the realm of Cloud infrastructure provisioning. c-Eclipse is an open-source Cloud Application Management Framework through which users are able to define the description, deployment and management phases of their Cloud applications in a clean and intuitive graphical manner. It is built on top of the well-established Eclipse platform and it adheres to two highly desirable features of Cloud applications: portability and elasticity. In particular, c-Eclipse implements the open, non-proprietary OASIS TOSCA specification for describing the provision, deployment and re-contextualization of applications across different Cloud infrastructures, thereby ensuring application portability. Furthermore, c-Eclipse enables Cloud users to specify elasticity policies that describe how the deployed virtualized resources must be elastically adapted at runtime to match the needs of a dynamic application-workload. In this paper, we introduce the architecture and implementation of c-Eclipse, and describe its key characteristics via a use-case scenario that involves a user creating a description of a 3-tier Cloud application, enriching it with appropriate elasticity policies, submitting it for deployment to two different Cloud providers and, finally, monitoring its execution.

1 Introduction

Application deployment and management in Infrastructure as a Service (IaaS) Clouds can be a complex and time consuming endeavor, typically requiring manual effort on the users' behalf and relying on vendor-specific, proprietary tools. Existing IaaS tools do not provide users with vendor-neutral mechanisms for describing application configuration, deployment, runtime application-scaling preferences, and elasticity policies. Consequently, the migration of applications between different IaaS providers requires significant re-configuration and re-deployment effort and time, leading to vendor lock-in. With the growing number of IaaS-provider service offerings and the increasing complexity of applications deployed on Clouds, the selection of the most appropriate provider to host an application becomes challenging. While seeking to identify the deployments that suit best their needs, IaaS clients need to overcome vendor lock-in in order to test and/or deploy their applications on multiple IaaS providers. Therefore, it becomes evident that there is a need for *application management tools* that facilitate the description of applications in a vendor neutral manner, enabling easy application deployment, management, and migration across different providers, preventing vendor lock-in.

F. Silva et al. (Eds.): Euro-Par 2014, LNCS 8632, pp. 38–49, 2014.

This article presents **c-Eclipse**, a generic Application Management Framework that:

- is *open-source* and has been implemented on top of the reliable Eclipse platform[1];
- offers graphical tools to *facilitate the description* of an application's structure and its lifecycle management operations;
- *adopts the TOSCA [1] open specification* for blueprinting Cloud applications and consequently packaging them in portable archives that can be processed by any compliant IaaS-provider;
- adopts a language that enables the description of *elasticity policies* for such Cloud applications;
- provides *tools for elasticity policy specification* at different levels of an application's structure.

To this end, c-Eclipse can be promoted by Cloud vendors as an enabling tool for configuring, deploying and managing Cloudified applications on their infrastructure. This is beneficial both for vendors and users; The former can integrate c-Eclipse to their Cloud architectures to attract a wider customer base to use their services via its GUI; the latter are able to describe - the often complex - deployment and management lifecycle of their applications with minimal effort and in a portable way, thus avoiding vendor lock-in.

The rest of the paper is structured as follows: Section 2 presents the related work in Cloud Application Managements platforms. Section 3 gives an overview of the c-Eclipse framework, its architecture, the application description language used and the UI. The c-Eclipse approach for describing elasticity policies for applications is discussed in Section 4. Finally, Section 5 presents a use-case scenario with a 3-tier application described via c-Eclipse and deployed on two separate Cloud infrastructures.

2 Related Work

Many application management frameworks have been developed lately to support Cloud Computing. Some of these frameworks are proprietary, locking their users to specific providers, while others are generic enough allowing management of applications on different infrastructures.

Proprietary: Amazon CloudFormation enables the creation and provisioning of EC2 infrastructure deployments. It uses JSON template files to describe the collection of EC2 resources that compose a deployment. Furthermore, by leveraging Amazon Auto Scaling it enables the specification of policies for automatically scaling the number of EC2 instances in a deployment. Oracle Virtual Assembly Builder (OVAB) [2] simplifies the provisioning of multi-tier applications by capturing the application components into self-contained VM appliances. OVAB can instantiate the appliances on Oracle's Exalogic Elastic Cloud Infrastructure and scale the deployed applications horizontally after a scale command is sent via the command line interface. VMware vCloud Application Director [3] is a provisioning solution that provides the necessary tooling for simplifying the process of designing, customizing and deploying applications on any

[1] `https://www.eclipse.org/`

VMware based Cloud infrastructure. From the well established aforementioned tools, only CloudFormation enables the specification of elasticity policies for automatic scaling, while all of them lock their users to specific IaaS providers.

Generic: Juju [4] is a tool for designing, configuring and deploying applications on a limited number of Cloud platforms. It makes use of shareable and reusable *charms* that encapsulate the configuration, deployment, connectivity and scaling information for an application. Charms are usually Linux oriented, thus limiting the portability of Juju applications. Also, Juju does not allow the specification of elasticity policies. The Agility Platform by ServiceMesh [5] enables the automatic deployment of applications on any Cloud provider, and the dynamic management of their lifecycle by defining auto-scaling rules for adding/removing VMs. Although ServiceMesh allows deployment of applications on different Cloud environments, it comes with a significant financial cost. Wrangler [6] provides a system for automatic deployment and monitoring of distributed applications with complex dependencies on different Cloud infrastructures, through a dedicated XML language. Users can describe a deployment; characteristics of the virtual resources, VM images, authentication credentials; and send it to a coordinating web service.

None of the aforementioned platforms adopts open Cloud standards for describing applications. In an effort to promote Cloud application portability, Winery [7] supports modeling of TOSCA applications via an HTML5-based environment. TOSCA elements are created via the Web-based GUI, which also allows users with prior knowledge of the TOSCA standard to define new types for the TOSCA elements, or configure the existing ones. Furthermore, Winery does not provide a straightforward way of specifying elasticity policies for applications. c-Eclipse on the other hand provides an intuitive GUI that hides all the complex details of the TOSCA standard, enabling thus users to exploit the full potential of the tool. In addition, c-Eclipse enriches the TOSCA specification with *Policy Types* for elasticity, and allows its users to specify the desired elasticity policies for their applications. Finally, Winery relies on BPEL to model applications' management plans, while c-Eclipse makes use of the TOSCA *Lifecycle Interface*. Another platform that uses TOSCA to automate the deployment and scaling of applications over any Cloud technology, is Cloudify [8]. It supports the creation of TOSCA application blueprints via an open-source CLI, however requiring users to master YAML and Python languages. This procedure gets easier when using the full-featured web interface, available only in a payware edition.

3 c-Eclipse Overview

This section presents the c-Eclipse framework focusing on the features that make it attractive to Cloud application developers. Furthermore, it provides a brief overview of the open Cloud application description specification adopted by c-Eclipse. It continues with the description of its architecture together with the necessary requirements when it comes to integration with Cloud vendors. Finally, the c-Eclipse UI is introduced.

3.1 c-Eclipse Features

The c-Eclipse Cloud application management framework incorporates the following characteristics:

- **Ease of Use:** It provides an intuitive and user-friendly GUI that minimizes any complexity regarding the process of Cloud application management, therefore serving as a low-entry barrier to Cloud technologies for new end-users. Not neglecting experienced users, GUI-driven operations can be manually fine-tuned, effectively allowing full workflow control when needed.
- **Elasticity Policies Specification:** It enables the specification of applications' elasticity policies, so that applications can benefit from the dynamic nature of Cloud environments.
- **Monitoring Interface:**It provides interfaces for integration with existing monitoring systems, so that its users can monitor the performance of their deployed applications and their resources thereof.
- **Cloud Vendor Neutral:** Through the adoption of the TOSCA open specification, allows its users to describe applications in a very generic way, so that they can be deployed across different Cloud infrastructures.
- **Platform Independence:** It runs on any OS supported by Eclipse.

3.2 TOSCA Specification for Cloud Applications

TOSCA provides a language to describe the structure of applications, together with their management operations. The structure of an application defines the components an application consists of and the relationships between them. Application components are described in TOSCA by means of *Nodes* (i.e. an application component can be a Tomcat application server in a 3-tier environment). Each Node can have certain semantics that are defined by the properties of the corresponding *Node Type*. Such semantics include the *Requirements* a Node has against its hosting environment, the *Capabilities* it offers and the *Policies* that govern its execution, such as security or elasticity policies. Similarly, TOSCA *Relationships* are used to represent the relations in an application's structure, and have their own semantics defined by their *Relationship Type*. The management aspects of an application are described in TOSCA either by means of lifecycle operations (via the *Lifecycle Interface*) or by more complex *Management Plans*. The *Lifecycle Interface* defines five operations (install, configure, start, stop, uninstall) for describing the management of applications' lifecycle. On the other hand, there is no TOSCA specific way to describe Management Plans. Instead, plans can be specified in any existing process modeling language, such as BPMN, and referenced through TOSCA. Both Lifecycle Operations and Managements Plans require some content to be realized, such as virtual machine images, configuration files etc. These contents are collectively referred to as *Artifacts*.

TOSCA application descriptions can be processed in an imperative or declarative manner. In case of imperative processing [9] the management behaviour of the described application has to be explicitly defined by the user by means of Management Plans. In declarative processing, the management behaviour of the application can be inferred by the semantics of Nodes' and Relationships' Types (i.e. operations specified in the *Lifecycle Interface* of a Type). The latter imposes extra overhead to TOSCA type architects who need to precisely define the semantics of each type, and for the implementers of the TOSCA processing environments who must correctly interpreter the

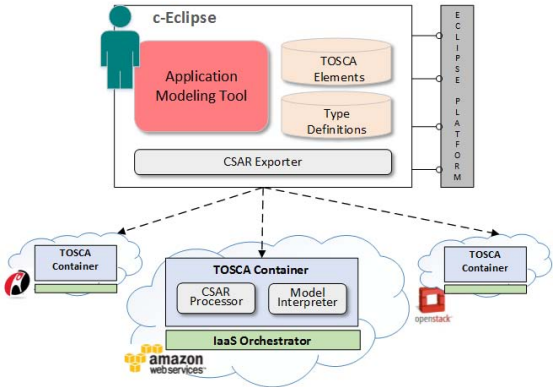

Fig. 1. c-Eclipse Architecture

types' semantics to infer an application's management plans. Consequently, declarative processing makes modeling of Cloud applications easier from the user perspective, since they don't have the extra overhead of defining Management Plans. For this reason, c-Eclipse adheres to the declarative processing approach.

3.3 c-Eclipse Framework Architecture

c-Eclipse is built on top of the Eclipse Platform and follows its OSGi plug-in based software architecture. Its main component is the **Application Modeling Tool**, which facilitates the creation of TOSCA application descriptions. The elements specified in TOSCA and the c-Eclipse specific type definitions for Nodes and Relationships, are stored in the c-Eclipse file system (**TOSCA Elements** and **Type Definitions**), so that they can be accessed by the Modeling Tool. The Application Modeling Tool associates the TOSCA elements and the defined Node and Relationship Types with visual elements that can be used to graphically model an application. The graphical description is translated on the fly into TOSCA, using the semantics of each element in the description. In order to provide such functionalities, c-Eclipse utilizes Graphiti[2], an Eclipse-based graphics framework that enables rapid development of state-of-the-art diagram editors for domain models. Graphiti is based on the Eclipse Modeling Framework (EMF) and offers graphical representations and editing possibilities for EMF objects. To this extend, the Application Modeling Tool transforms TOSCA elements into EMF objects and uses the Graphiti infrastructure to build the graphical editor through which users can schematically describe their applications The TOSCA description along with any artifacts for materializing and managing the described application are packaged into a single archive file (CSAR) by the **CSAR Exporter**. Fig. 1 depicts the high level architecture and the major components of c-Eclipse.

The exported CSAR is passed from c-Eclipse to a TOSCA processing environment operated by a Cloud provider. This environment, referred to as a **TOSCA Container**,

[2] `https://www.eclipse.org/graphiti/`

must be able to process CSAR files and understand the semantics of the contained application description, so that it can deploy and manage the application throughout its lifecycle. Each application modeling tool can define its own types, for various TOSCA elements, with different properties and interfaces. Thus, in order for the TOSCA Container to process a TOSCA description in a declarative manner, which implies deriving based on the type definition of each element the order in which the specified management operations must be executed, the type definitions utilized in a description must also be known to the TOSCA Container. Consequently, a CSAR archive file must contain the following so as to be portable and processable by any TOSCA Container: (1) The XML file specifying the TOSCA-based application description, (2) The definitions of the Node, Relationship and other elements' types that are used in the TOSCA description, and (3) the artifacts that realize an application's management operations and that are referenced in the TOSCA description.

A TOSCA Container might include various components that can be used to process CSARs. Each vendor can decide what components to support and how to provide them within his Cloud architecture. A Container that supports declarative processing of CSARs must implement at least two components: **CSAR Processor** and **Model Interpreter** (Fig. 1). The CSAR Processor receives the CSAR from the **TOSCA Container** and is responsible for the extraction and deployment of the artifacts. Once the artifacts are ready to be used by the TOSCA Container, the Model Interpreter navigates the application's structure and distinguishes the artifacts realizing the management operations of each Node, such as installing/uninstalling instances. Other components that can be implemented by the container, are a **Definition Manager** component in charge of storing the type definitions and making them available to the Model Interpreter and an **Artifact Manager** component for storing the artifacts in appropriate stores.

According to the specification, Cloud providers that wish to become TOSCA-compliant should provide a Container as part of their Cloud architecture. The Container must communicate with an **IaaS Orchestrator** to invoke the necessary IaaS-specific API calls that satisfy the respective TOSCA description. An alternative way of integrating TOSCA modeling tools, such as c-Eclipse, with Cloud providers is to implement a TOSCA Container at the tools' side, with interfaces to multiple Cloud infrastructures. To this extend, Cloud providers should offer the required APIs, so that they can be accessed by the Containers. However, this endeavour entails in-depth knowledge of several complex APIs (sometimes lacking sufficient documentation) and extensive development skills to produce a fully working Container at the tools' side. This was observed and confirmed at first hand, while working towards the evaluation of c-Eclipse in a real scenario (see Section 5), where we implemented simple yet functional TOSCA Containers for two Cloud vendors. Among other, developing a TOSCA Container for a particular IaaS requires to provision for the exchange of authentication tokens, perform validity checks for CSARs, correct deployment/configuration of virtualized instances given defined Node Types, as well as, user requirements and constraints.

Finally, c-Eclipse provides the necessary interfaces so as to be integrated with existing monitoring systems, enabling thus its users to acquire and record the performance of their deployed applications from a single working environment. Currently, it is fully integrated with the JCatascopia [10] monitoring system.

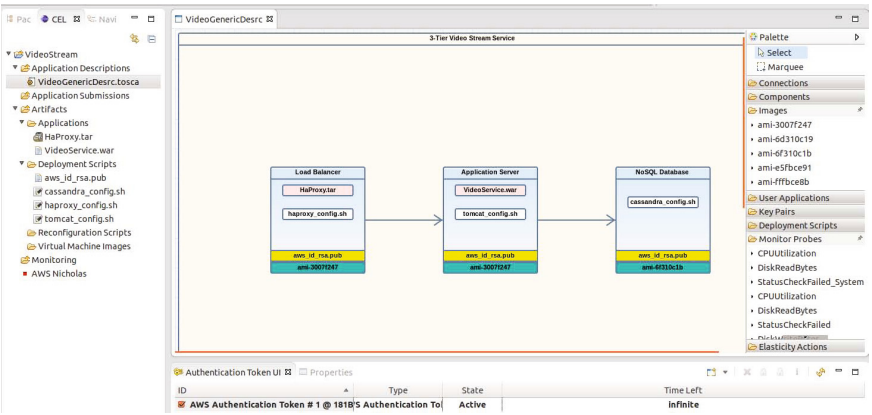

Fig. 2. c-Eclipse UI - (Left) Cloud Project View, (Center) Canvas, (Right) Palette, (Below) Authentication View

3.4 c-Eclipse User Interface

Like any other Eclipse project, c-Eclipse organizes all the files related to an application in a structured hierarchy, as depicted in Fig. 2. A Cloud project, in the Cloud Project View, acts as a placeholder for a single Cloud application and consists of four folders: (i) the *Application Descriptions* folder containing TOSCA descriptions of applications, (ii) the *Application Submissions* folder containing details about application deployments (i.e. Cloud provider, deployment status, total cost etc.), (iii) the *Artifacts* folder with the actual files for the artifacts referenced in the application description, and finally (iv) the *Monitoring* folder including any monitoring data collected by the integrated monitoring system during application's deployments.

Application developers can use the Modeling Tool to describe a Cloud application graphically. The most important part of the tool is the Palette, which includes most of the elements required for creating application descriptions. These are the application components (one component element in the Palette for each distinct Node Type), Relationships (one relationship/connection element in the Palette for each distinct Relationship type), artifacts and monitoring metrics. By simply dragging and dropping pictorial elements from the Palette onto the Canvas of the tool, developers can create a graphical representation of an application. Throughout the application description process, the Modeling tool translates on-the-fly the graphical description into TOSCA and error-proofs the generated TOSCA to assure adherence to the specification, prompting warnings if necessary.

Apart from the default semantics that each Palette element has, additional information can be provided for each element contained in the description, by using the *Properties View* of the tool. For example, the view can be used for uploading custom images for application components, specifying elasticity policies for the whole application and/or for components separately, writing deployment scripts etc. Fig.3 presents a tab in the properties view for specifying elasticity constraints and strategies for a specific application component.

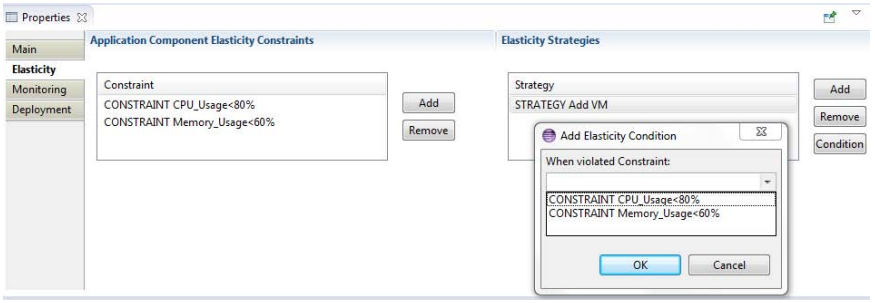

Fig. 3. c-Eclipse Properties View (Elasticity constraints and strategies tab)

Users with expertise in writing XML and with deep knowledge of the TOSCA specification, can manually create or edit an application's TOSCA XML description. Any changes in the XML will be automatically reflected to the corresponding graphical description. This way c-Eclipse attracts broader audience, from entry level to more advanced users.

4 Elasticity Specification in c-Eclipse

Apart from enabling portable automated application deployment and management, c-Eclipse facilitates the specification of applications' elasticity policies so that they can scale at runtime based on user defined policies. Since the TOSCA language does not directly specify how to define elasticity policies for Cloud applications, c-Eclipse exploits the TOSCA *Policy* element to achieve elasticity specification without interfering with applications' portability. TOSCA defines policies as the means by which we can express non-functional behaviour or quality-of-services for an application.

We use two types of elasticity-oriented TOSCA policies in accordance with the SYBL [11] language for elasticity requirements specification: *Elasticity Constraint* and *Elasticity Strategy*. The Elasticity Constraint type is used to express the constraints of an application, related to cost, performance and other application-quality metrics. Here the application user does not specify the exact actions to be enforced when a constraint is violated. Instead, the appropriate actions are determined by the underlying intelligent elastic *Resource Provisioning System* [12]. The Elasticity Strategy type, is used to express specific strategies that should be enforced by the execution environment when specific constraints are violated.

The purpose of defining two distinct TOSCA Policy Types of elasticity is twofold. Cloud users can:

- Specify elasticity constraints and strategies for their applications at different levels of detail, based on their expertise.
- Fully exploit the capabilities of the underlying Resource Provisioning System. In case the underlying system is smart enough to take scaling decisions on its own, the user specifies only the elasticity constraints and relies on the system to decide how to fullfil them.

The purpose of specifying elasticity policies in c-Eclipse is to give its users more control over their deployments. Elasticity policies are translated into SYBL, and injected into the TOSCA description. If the IaaS resource provisioning system supports dynamic scaling of applications, then the specified elasticity policies are translated, (by the TOSCA Container) to provider specific elasticity rules. Otherwise, the defined elasticity policies will be ignored.

5 Use-Case

This section aims at demonstrating the portability and elasticity support capabilities of the c-Eclipse Cloud Application Management Framework. To do so, we present the description, deployment and management phases of an exemplary Cloud application on two environments: (i) Amazon's EC2 infrastructure and (ii) Nephelae[3], our own OpenStack-compliant Cloud research infrastructure.

Before starting the demonstration we needed to implement our TOSCA Containers, as described in Section 3.3, and deploy them on a single virtual instance both on Amazon EC2 and Nephelae. Our simple container for AWS is composed by ≈ 450 lines of Code (LOC), implementing 24 needed functions. Similarly, the OpenStack container needed ≈ 600 LOC and same number of functions. In order to instrument the application's deployment we also needed a monitoring system to be deployed on both infrastructures. In contrast to EC2[4], Nephelae does not include a native resource and application monitoring solution. Therefore, we instantiate the JCatascopia system for providing the monitoring metrics that will be utilized during the specification of elasticity policies. Finally, we assume each tier instance runs on a Linux-based OS.

Use-Case Scenario: We consider a 3-tier Web application that provides video streaming services to online users. The tiers comprising the application are as follows: (i) a *Load Balancer* which serves as an entry point and distributes incoming user requests across multiple application servers, (ii) the *Application Server* itself, which is materialized through an Apache Tomcat server with the necessary video streaming Web application deployed, and (iii) a *Cassandra*[5] *NoSQL distributed data storage back-end* from where the necessary video content is retrieved.

Application Description Phase: In this first step, the application developer initiates the description process by creating a Cloud project, which will be unique for the above Web application. The necessary folder structure (see Section 3.4) is automatically created, establishing placeholders for individual components required throughout the application management lifecycle. At the same time, the developer is prompted to enter service endpoints and authentication credentials[6] for one or more candidate Cloud provider(s), where the application might eventually be submitted for deployment. The *Authentication Token View* gives an overview of credential details (Fig. 2).

[3] `http://linc.ucy.ac.cy/Nephelae/`

[4] AWS provide the CloudWatch solution for monitoring applications and Cloud resources.

[5] `http://cassandra.apache.org/`

[6] Credentials are managed in a secure manner using the native Eclipse password manager.

The next step involves creating the application description itself through a guided wizard and subsequently invoking the Modeling Tool, where the respective application structure will be defined. During this phase, the user designs a coarse-grained blueprint of the application structure, avoiding reference to vendor-specific details. This way, the description is portable across different providers. Consequently, at this stage the Palette contains only those generic components that will later-on act as containers for vendor-specific information. Such structural parts include: the application components and the relationships.

For the use-case scenario at hand, the coarse-grained application blueprint is comprised of 3 different application components (Fig. 4). The Load Balancer component is populated with an HA Proxy[7] tarball (orange color box) and a Bash script (white color box) for the respective configuration. Similarly, the Application Server component is populated with the Web application ARchive (WAR) that provides the video streaming functionality and a Bash script for minimal Tomcat configurations. The NoSQL database component is populated with a Bash script for contextualization purposes, such as seed node IP address, listening ports, etc. Additionally, each Component is enriched with a common RSA keypair[8] for shell-access purposes (yellow color box). Finally, the necessary inter-dependencies in the application's structure are specified via the two Relationships shown in Fig. 4. Generic application descriptions are stored in the Application Descriptions folder, and can be used later as customizable templates which can be enriched with vendor-specific information at the deployment phase.

Application Deployment Phase: Once the application developer completes the generic design, it is time to engage in a more fine-grained topology description by providing vendor-specific information. To do so, the user has to invoke the application deployment phase through a context menu action on the description file. This phase is again a wizard driven process requiring the user to select the target Cloud provider where the application deployment will eventually take place. The Palette and Properties Views are now populated with vendor-specific information retrieved by interrogating the IaaS API. In addition to the standard information advertised by the provider such as compute resources availability, volumes and networking configurations, the Palette provides monitoring metrics available by the monitoring systems on EC2 and Nephelae.

To minimize the information displayed and swiftly identify any required component, the Palette includes standard searching and filtering mechanisms. Given that each tier instance of the video streaming service will run on a Linux-based OS, the developer sets the necessary filters to expose available base images that include a 64-bit Ubuntu 12.04 server. For the Application and Database components, the filters are adjusted to search for available Ubuntu-based images that include Apache Tomcat and Cassandra NoSQL, respectively. When suitable images are returned, a simple drag-n-drop operation of their pictorial representations from the Palette to the respective application components (green color box), results to their inclusion within the generated TOSCA description. In the case that matching images are not retrieved, c-Eclipse provides the necessary fields through which the developer can pass specific scripts (or artifact filenames) that will be executed upon contextualization.

[7] `http://haproxy.1wt.eu/`

[8] Only the public key material of the RSA keypair is included within the TOSCA description.

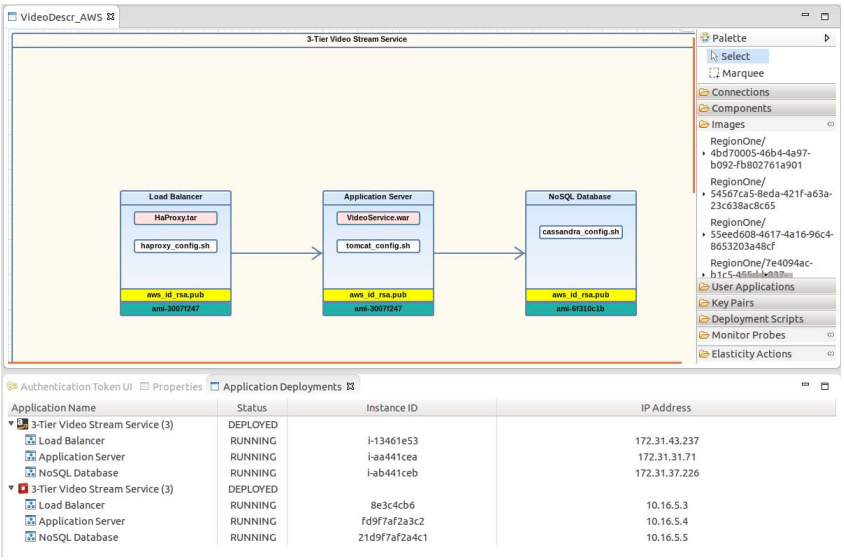

Fig. 4. Application Deployment on Amazon EC2 and Nephelae

What remains to do before inhibiting the actual application deployment process, is for the developer to specify the elasticity-oriented policies. This includes selecting one or more available monitoring metrics from the Palette and assigning them to the components whose resources need to be elastically adapted on runtime. For the video service, it was decided to scale-up only the Application and Database components by adding a new virtualized instance when the CPU utilization threshold exceeded 80% (see Fig. 3). To achieve this, each component was assigned to a CPU probe that reports utilization to the underlying IaaS orchestrator in frequent time intervals.

The customized description, is stored under the Application Submissions folder attributed with the name of the Cloud provider. Upon the completion of the fine-grained description, the application can be submitted to the target Cloud infrastructure for deployment. With a context-menu action, the CSAR Exporter creates the CSAR containing the description with the artifacts, and hands it to the TOSCA Container at the selected IaaS provider.

Application Management Phase: Finally, through, the c-Eclipse *Deployment View*, the application developer can instantly obtain the deployment status without leaving the Eclipse environment. As depicted in the lower part of Fig. 4, a snapshot of the deployments on EC2 and Nephelae is provided, along with provider-specific properties such as component IP addresses, instance IDs, running times etc. A background polling mechanisms refreshes the view and provides the latest information from each IaaS.

6 Conclusion and Future Work

In this paper we present c-Eclipse; an open-source, vendor neutral, Cloud Application Management Framework built on top of Eclipse. c-Eclipse aims at facilitating the

deployment and management of Cloud applications, promoting portability of applications across infrastructures, and supporting application elasticity. It adopts an open Cloud standard, and provides a unified environment for describing the structure, deployment and management operations of applications. It then exports the applications' descriptions into portable archives that can be processed by different providers. The functionality of c-Eclipse is presented via a use-case scenario with a 3-tier application being described and deployed on private and public Cloud infrastructures. Though still a prototype, c-Eclipse is currently used in the CELAR Project to deploy elastic Cloud applications. As future work, we will extend c-Eclipse to support existing application configuration management tools, such as Chef (`http://getchef.com`), to automatically provision and configure applications on new node instances, without requiring the user to write custom deployment scripts. c-Eclipse is available on GitHub at `http://github.com/CELAR`.

Acknowledgments. This work was partially supported by the European Commission in terms of the CELAR 317790 FP7 project (FP7-ICT-2011-8) and by the European Regional Development Fund and the Republic of Cyprus through the Research Promotion Foundation ("Infrastructure Upgrade /0609/09" project). The authors thank Andreas Papadopoulos, Georgiana Copil and Demetris Antoniades for their fruitful insights.

References

1. OASIS: TOSCA Version 1.0, `http://goo.gl/ApNP3C`
2. Oracle Virtual Assebly Builder, `http://goo.gl/Eetq0V`
3. VMware vCloud Application Director, `http://goo.gl/j7LyU7`
4. Ubuntu Juju, `https://juju.ubuntu.com/`
5. ServiceMesh Agility Platform, `http://www.servicemesh.com`
6. Juve, G., Deelman, E.: Automating Application Deployment in Infrastructure Clouds. In: Proceedings of the 2011 IEEE 3rd International Conference on Cloud Computing Technology and Science, pp. 658–665. IEEE Computer Society (2011)
7. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery: A Modeling Tool for TOSCA-Based Cloud Applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 700–704. Springer, Heidelberg (2013)
8. GigaSpaces Cloudify, `http://goo.gl/rYGceK`
9. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA - A Runtime for TOSCA-Based Cloud Applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 692–695. Springer, Heidelberg (2013)
10. Trihinas, D., Pallis, G., Dikaiakos, M.D.: JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (2014)
11. Copil, G., Moldovan, D., Truong, H.L., Dustdar, S.: SYBL: An Extensible Language for Controlling Elasticity in Cloud Applications. In: 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 112–119 (2013)
12. CELAR EU FP7 Project, `http://celarcloud.eu/`