1.1 Getting Started

1.1.1 Open CAMF Perspective

The CAMF Perspective contains the Views and Editors utilized by CAMF.

Open the CAMF Perspective: Window -> Open Perspective -> Other -> CAMF. (Figure 1)

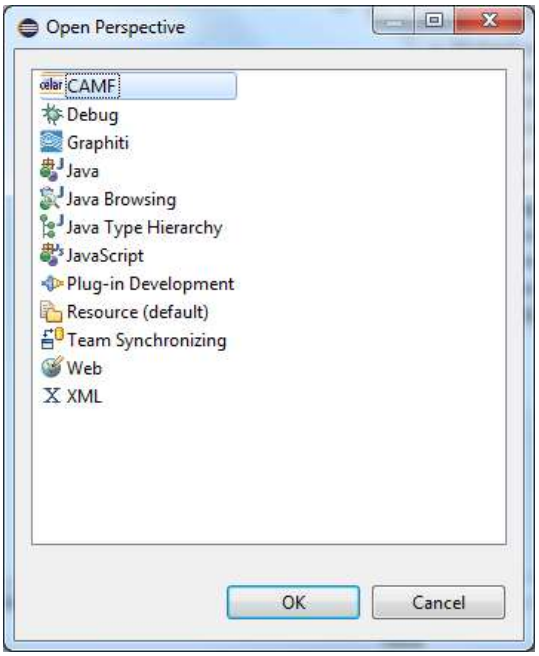

Figure 1

You can close the *Cloud Information Tool* view, since we are not going to need it for the current tutorial.

1.1.2 Create a new Cloud Project

Right Click in the Cloud Project View -> New -> Other -> Cloud Application Management Framework -> Cloud Project. (Figure 2)

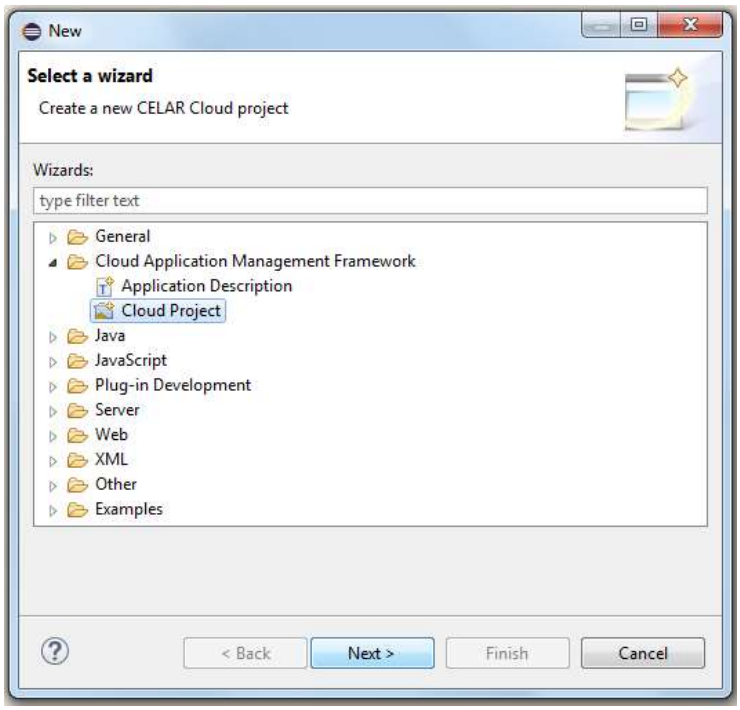

Figure 2

Next -> Give a name for the newly created project -> Next -> Select a Cloud Provider to associate the project with (Figure 3) -> Finish (if you have not already added providers to your workspace you can do so by clicking on the Edit Cloud Providers button and following the instructions in section 1.1.3).

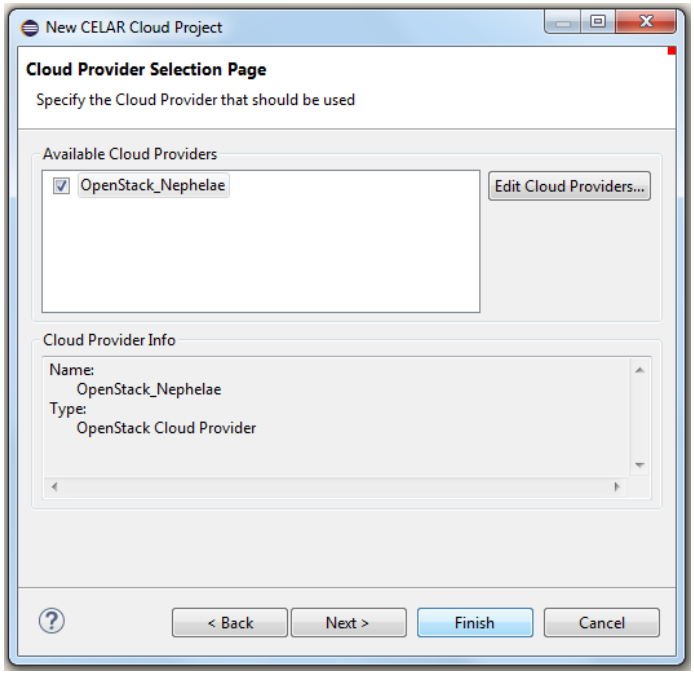

Figure 3

The new project is created and the project's folder structure is shown in the Cloud Project View (Figure 4). The created folders will be used as follows:

- Application Descriptions: holding topology blueprints defined using TOSCA. Each blueprint can be unique by specifying a different structure and management operations for the application at hand.
- Application Deployments: holding IaaS-flavored blueprints along with important operational records (past and current) regarding different deployment. Amongst other, these can include date/time, the target deployment IaaS, version of the application description, operational costs, etc.
- Artifacts: holding concrete software implementations required for the successful deployment and correct operation of the application. These include, and not limited to: executable and/or 3rd party libraries, custom virtual machine images, Chef cookbooks, O/S-specific configuration scripts, etc. The Cloud project structure, allows for artifacts to be referenced by multiple descriptions thus avoiding unwanted duplication.
- Monitoring: holding monitoring metric collectors prepared by the developer. These probes can be placed anywhere on the application stack ranging from the virtualization layer upwards to the application itself, in order to obtain meaningful metrics that report the runtime health and performance of the application.

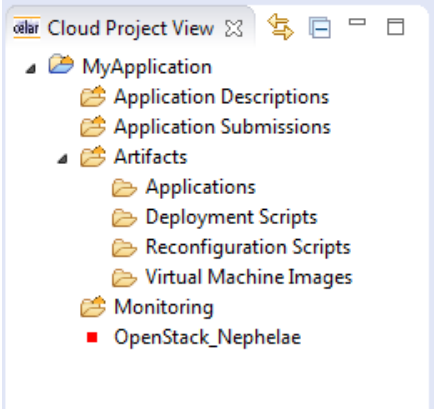

Figure 4

1.1.3 Manage Cloud Providers

1.1.3.1 Add Cloud Provider

If the Cloud Providers wizard is not open go to Window -> Preferences -> Cloud Application Management Framework -> Cloud Providers

Add -> OpenStack Cloud Provider (Figure 5) -> Next: Give a name for the OpenStack compliant provider to which you have already been registered and have acquired the necessary credentials (username and password). Fill in the Access Id (username) and Endpoint (URI) and click Finish -> OK. Here you can specify as many providers as you want. The selection of a provider to submit an application for deployment is done at a later stage.

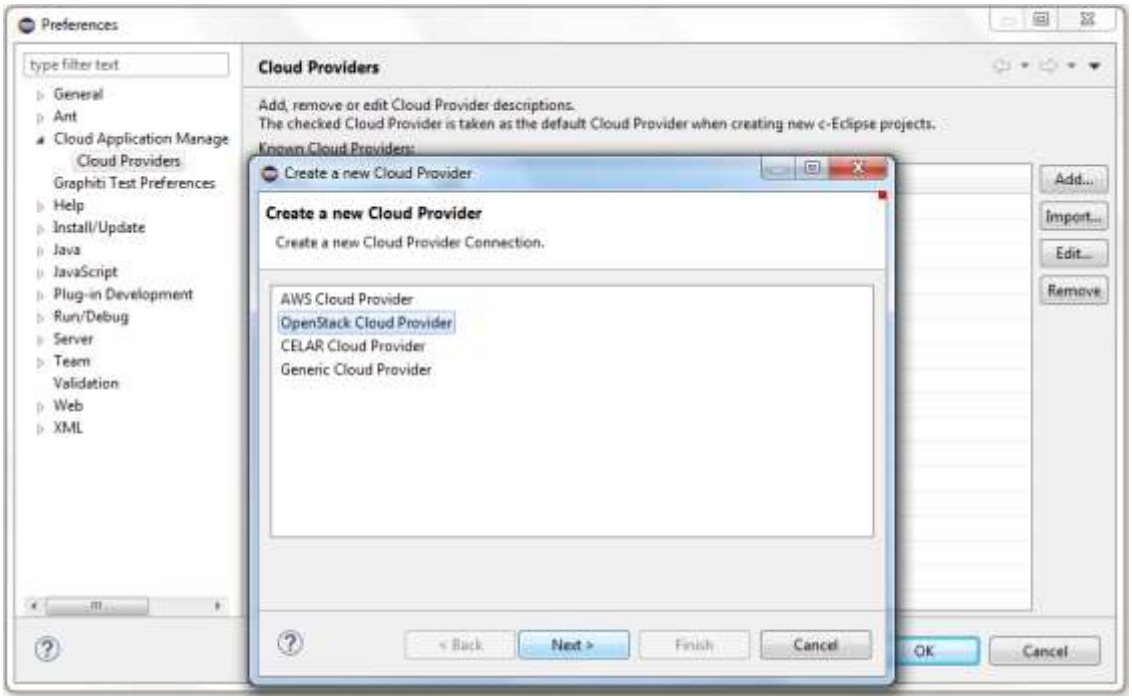

Figure 5

1.1.3.2 Edit Cloud Provider

If the Cloud Providers wizard is not open go to Window -> Preferences -> Cloud Application Management Framework -> Cloud Providers

Select the provider to be edited -> Edit -> OpenStack Cloud Provider -> Next: You can update the Access Id (username) and/or Endpoint (URI) (Figure 6). Click Finish -> OK.

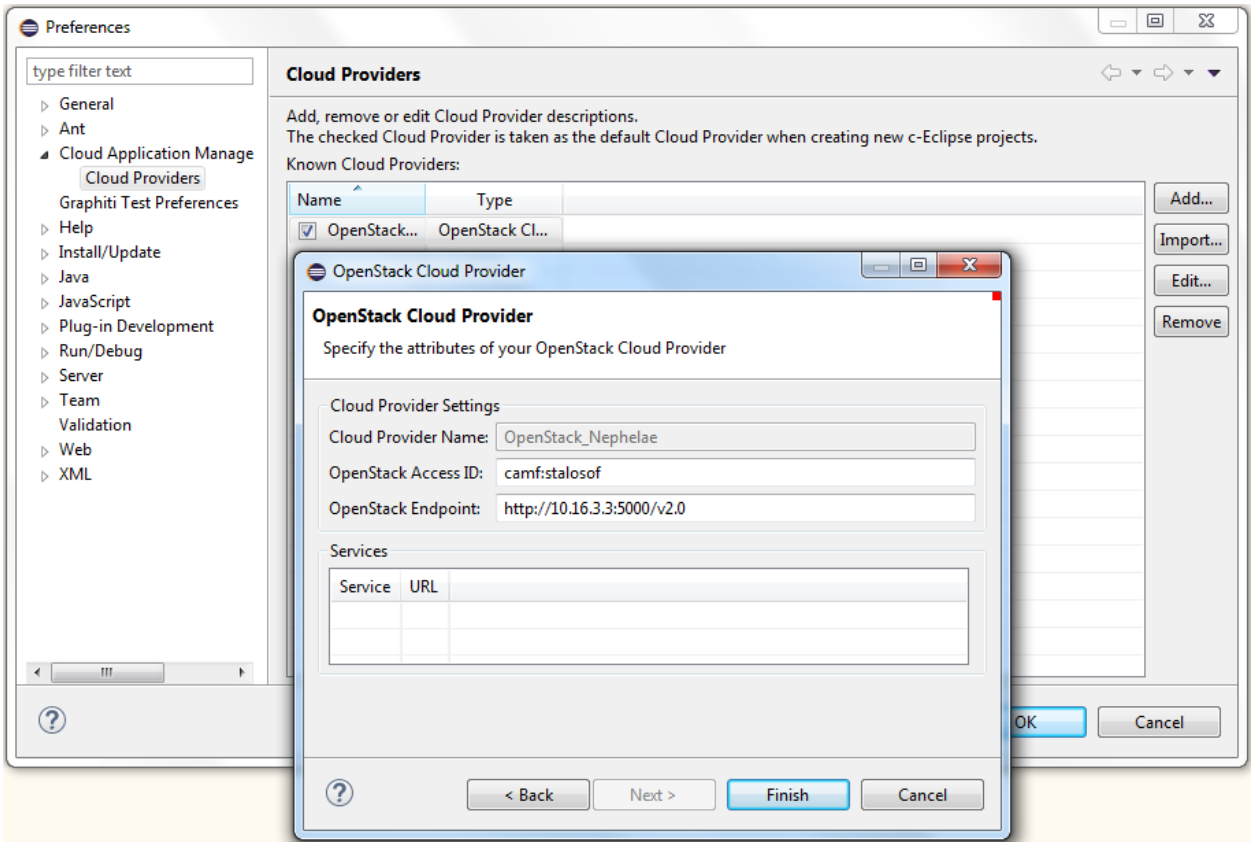

Figure 6

1.1.4 Cloud Application Description

1.1.4.1 Create application description file

Right Click on a Project in the Cloud Project View -> New -> Other -> Cloud Application Management Framework -> Application Description -> Next -> Give a name for the newly created description -> Finish.

The application description file is created under the Application Descriptions folder with extension .tosca (Figure 7).

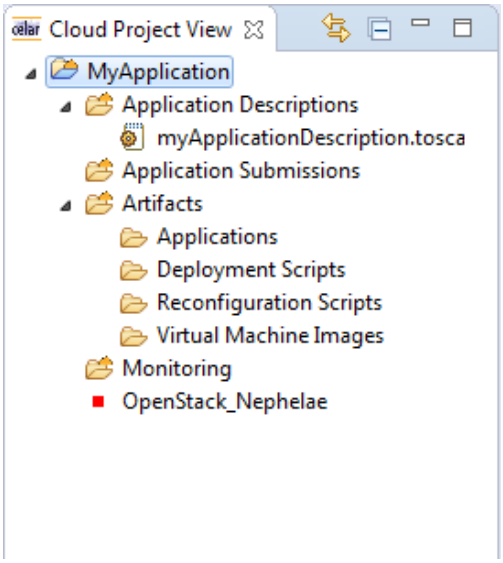

Figure 7

1.1.4.2 Fetch Cloud provider info

When creating the first application description in a project, the framework tries to fetch the following data from the specified OpenStack Cloud provider:

- *Images:* Virtual machine images that are made available through the OpenStack Image Service
- *Networks:* Virtual networks to enable compute servers to interact with each other and with the public network
- *Key Pairs: User-created keys (i.e. rsa keys) that can be used to access the instances once they have been launched*
- *Flavors*: A flavor represents a set of virtual resources. Flavors define how many virtual CPUs an instance has and the amount of RAM and size of its ephemeral disks. OpenStack provides a number of predefined flavors that you can edit or add to. Users must select from the set of available flavors defined on their cloud.

A pop up window asks if you want to fetch this data (Figure 8).

Figure 8

Click Yes and add the missing credentials (password) for the selected provider -> Finish (Figure 9).

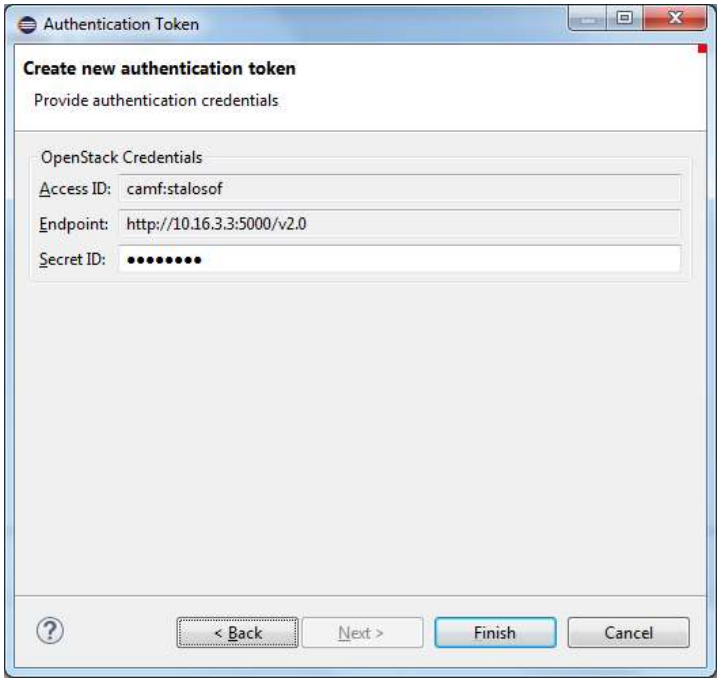

Figure 9

1.1.4.3 Check authentication details

Once the user has provided her full credentials (username and password) for the specified endpoint and has been authenticated by the respective Cloud provider, her authentication details are shown in the *Authentication Token UI* view (Figure 10).

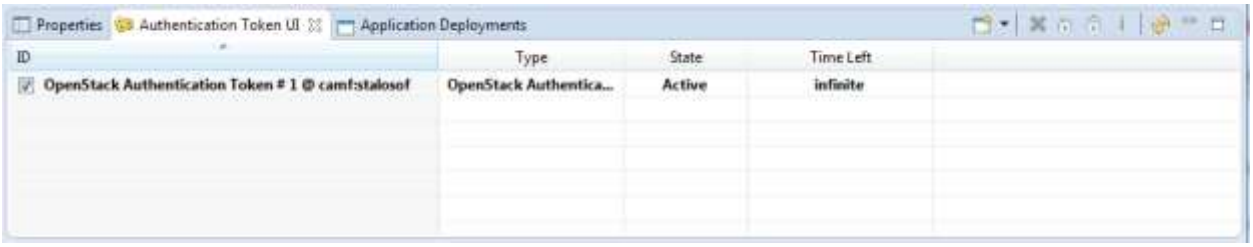

Figure 10

1.1.4.4 Describe Cloud application structure/contextualization

Right click on the application description file -> Open With -> Tosca Diagram Editor. You can also open the xml TOSCA file at any time, by selecting Open With -> Text Editor.

All the information required for the description can be found in the *Palette* (Figure 11) and/or in the *Properties* view (Figure 12). The Palette includes some generic elements (application components and relationships), as well as Cloud provider specific elements previously fetched (images, networks and key pairs). Elements from the Palette can be simply dragged and dropped onto the center *Canvas* of the tool.

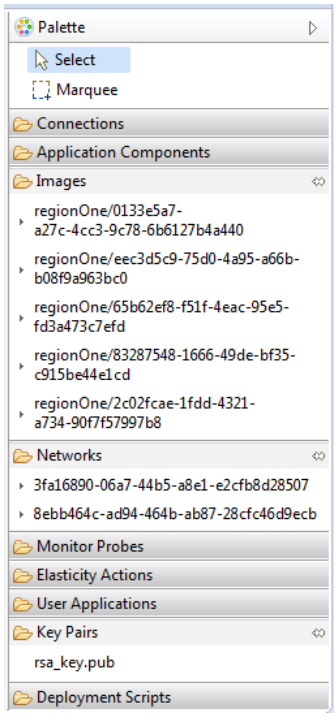

Figure 11

The rest of the information (flavors) can be found in the *Properties* view (Figure 12).

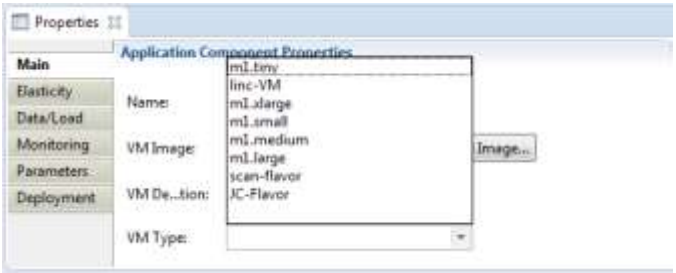

Figure 12

Users can also use image and configuration files stored in their local file system, simply by importing them into the respective folders of a Cloud Project (Virtual Machine Images, Deployment Scripts folders). Once the files are imported into the Cloud Project, they are also added in the Palette.

Right Click -> Import -> General -> File System -> Next -> Browse -> Select a directory to import from. (Figure 13)

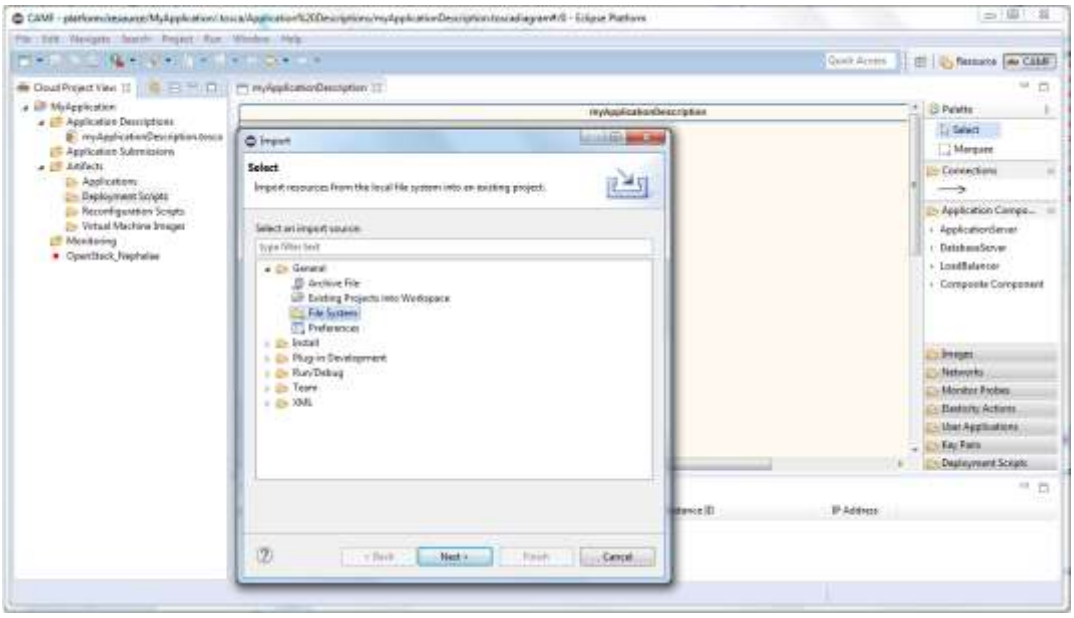

Figure 13

Firstly decide the number of application components to put in the description. Each application component corresponds to a virtual machine instance. Then, for each application component the image (green box), flavor and network(pink box) must be specified. Additional attributes can be specified, such as key pair (yellow box), deployment/configuration scripts (beige box), and initial number of instances (Figure 14).

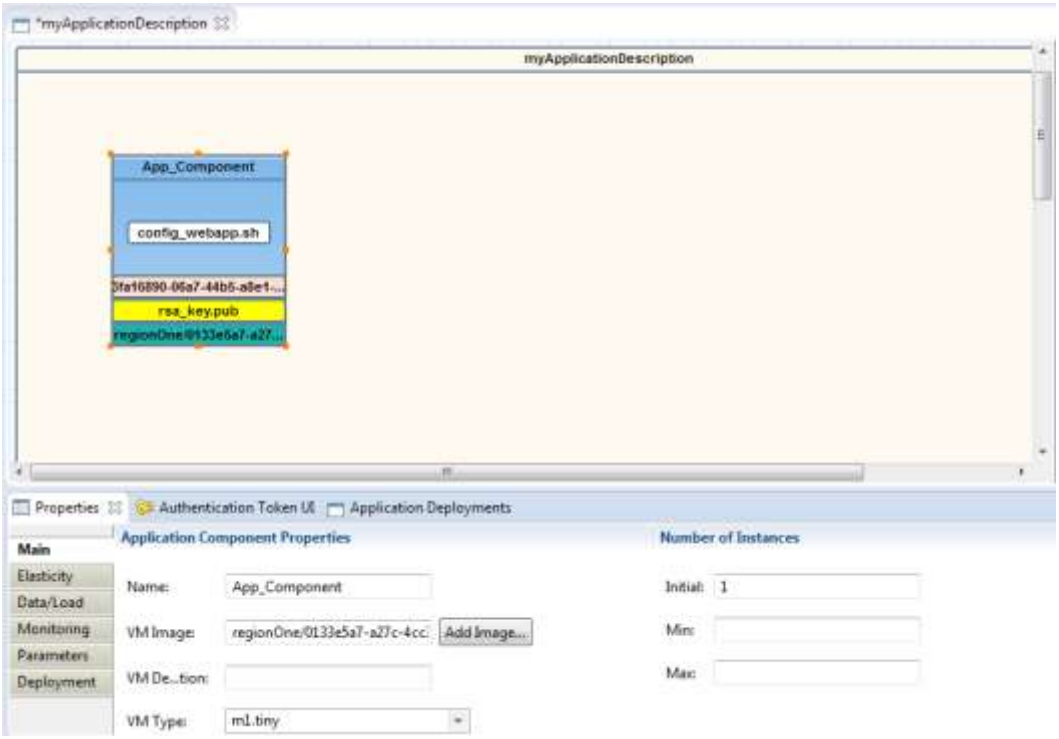

Figure 14

1.1.4.5 Specify application's elasticity requirements

You can specify elasticity policies for your application so that it can scale at runtime based on the defined policies.

There are two different types of elasticity policies:

- *Elasticity Constraint:* 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[1].
- *Elasticity Strategy:* Used to express specific strategies that should be enforced by the execution environment when specific constraints are violated.

The policy presented in Figure 15 describes an elasticity constraint where the application provider wants the CPU usage of a database cluster to be less than 80%. In Figure 15 the application provider uses an elasticity strategy to specify that when the CPU usage of the cluster exceeds 80%, a new database VM should be added to the cluster.

Figure 15

In order for the user to specify elasticity policies related to a monitoring metric, a corresponding monitoring probe (i.e. CPU Usage probe) from the Palette (Figure 16) must be assigned to the respective component by dragging the probe from the Palette and dropping it to the component.

[1] http://www.celarcloud.eu/

Figure 16

1.1.5 Submit application description to Cloud provider

Once the application description is completed Right Click on the .tosca file under the Application Descriptions folder -> Application Submission (Figure 17).

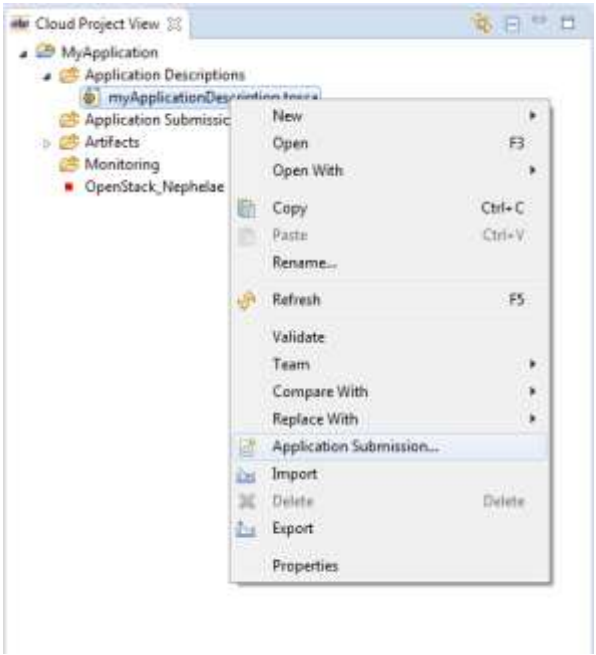

Figure 17

Give a name for the submission file (Figure 18).

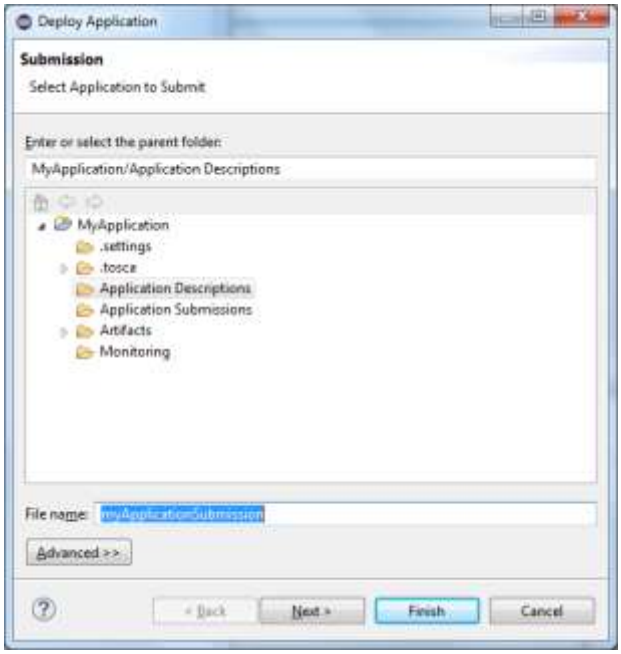

Figure 18

Select the provider to submit the file to (Figure 19).

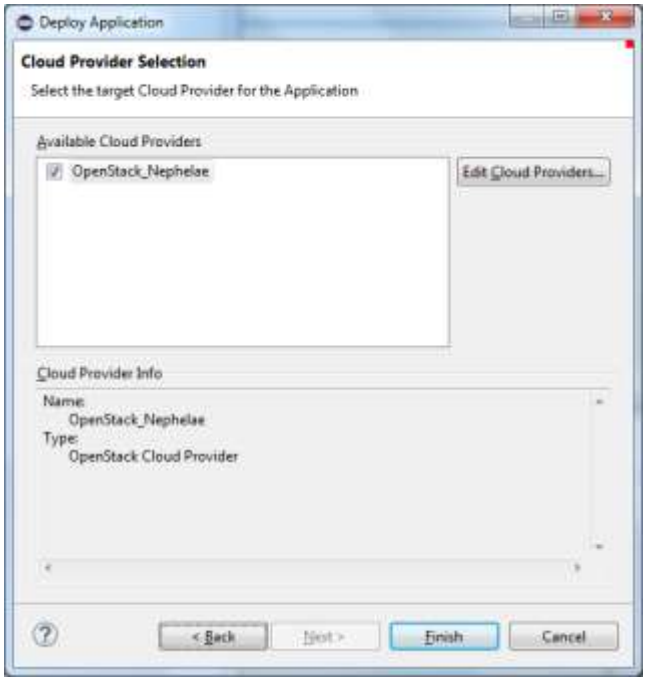

Figure 19

The submission file is created under the Application Submission folder (Figure 20).

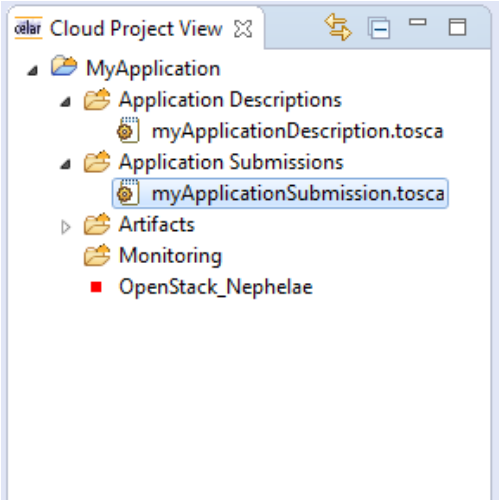

Figure 20

1.1.6 Deploy application to Cloud provider

1.1.6.1 Create deployment file

What remains is to send the deployment request to Cloud provider of your choice. Right Click on the .tosca file under the Application Submissions folder -> Application Deployment (Figure 21).

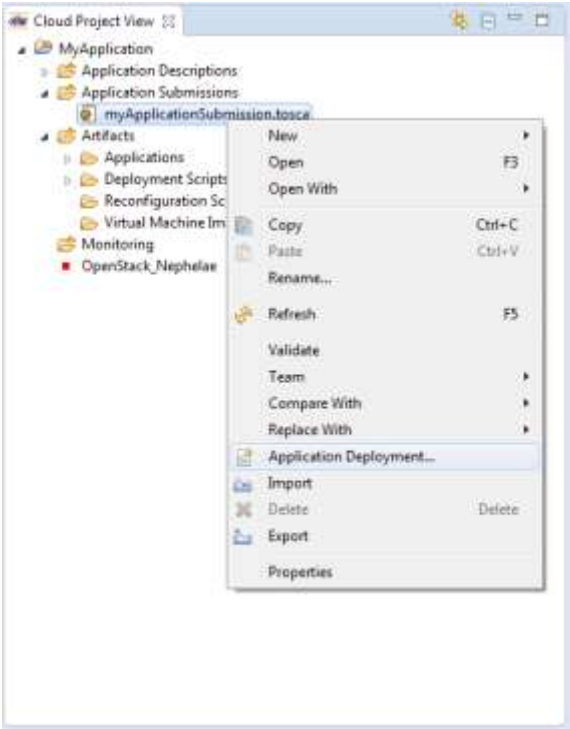

Figure 21

Select the Cloud provider over which you wish to deploy your application -> Finish (Figure 22).

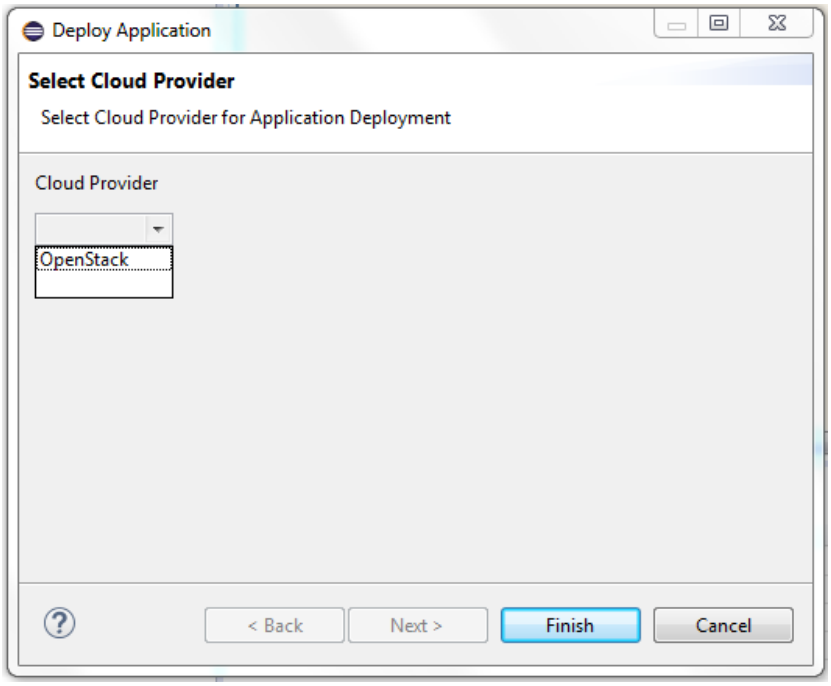

Figure 22

1.1.6.2 Watch deployment status

After deployment, the application developer can interact with the *Deployment* view (Figure 23), so as to instantly retrieve the applications' operational status without leaving CAMF's workspace. The *Deployment* view provides a snapshot of all application deployments grouped per target IaaS. Each deployment is accompanied with provider-specific properties such as IP addresses of each component, instance IDs, uptime and cost information, if available.

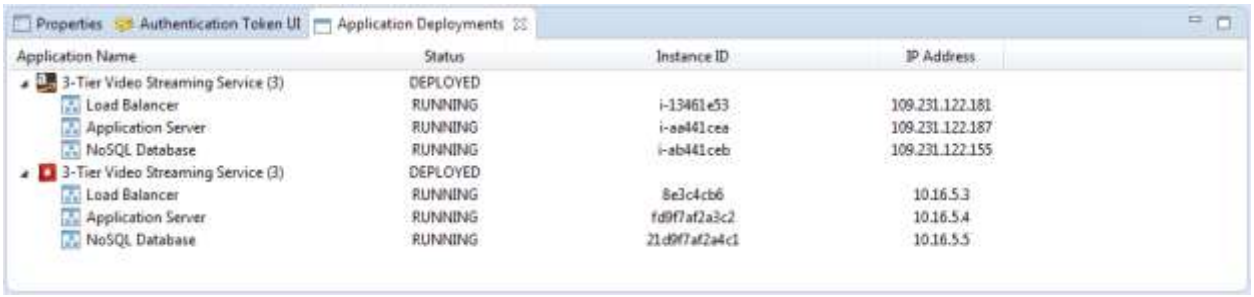

Figure 23

1.1.7 Monitoring

CAMF enables integration with different monitoring systems, enabling its users to collect performance metrics regarding any deployed application without being required to deal with external software environments.

Currently, CAMF is fully integrated with JCatascopia[2], an automated, multi-layer interoperable monitoring system for elastic Cloud environments. Besides the standard probe suite available in JCatascopia, CAMF allows application developers to define and implement custom metric collectors that adhere to JCatascopia's modular architecture. These probes can be placed anywhere on the application stack ranging from the virtualization layer upwards to the application itself, in order to obtain meaningful metrics that report the runtime health and performance of the application.

1.1.7.1 Create JCatascopia Monitoring Probe

Users can write custom java monitoring probes in CAMF.

Click on the Monitoring tab in the Properties view -> Create (Figure 24).

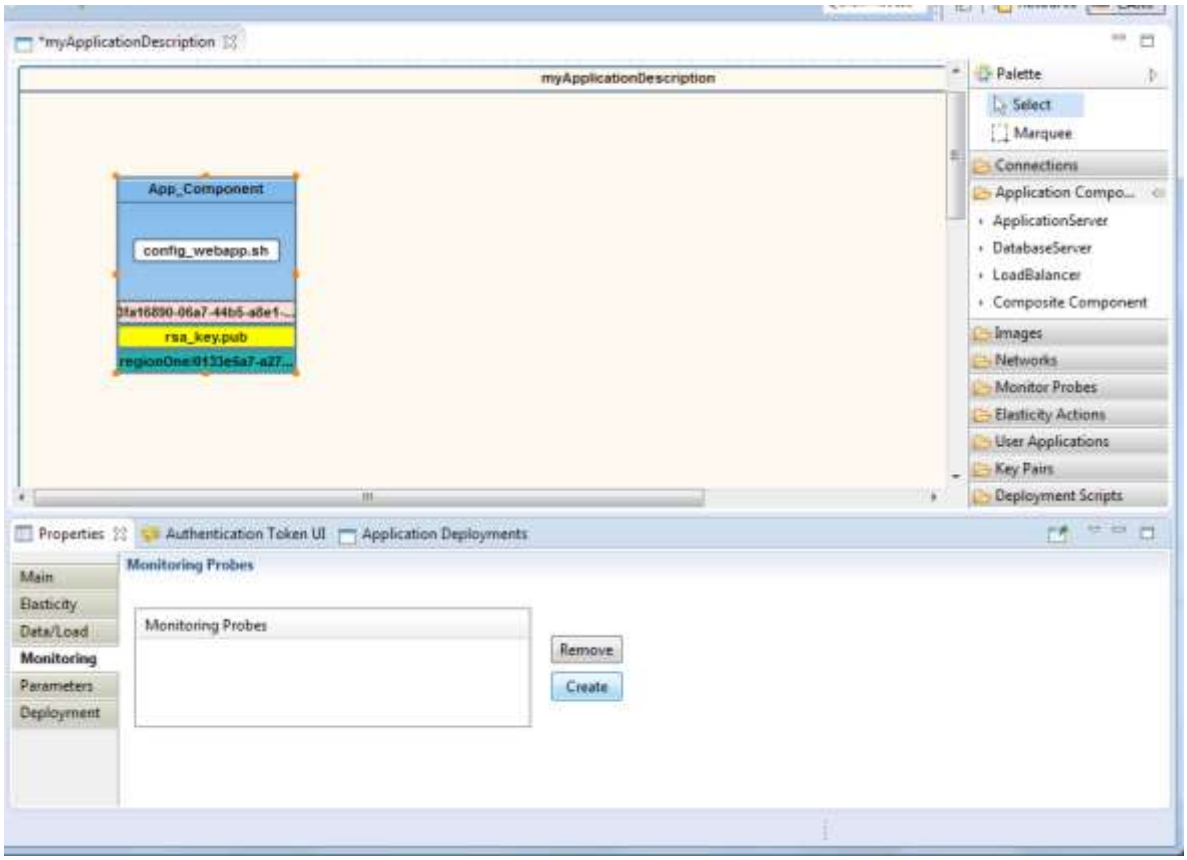

Figure 24

A java class that extends the JCatascopia Probe interface is automatically generated (Figure 25), guiding the user on how to implement the required methods. The created probes are exported as .jar files and are packaged in the CSAR file together with the application deployment file. At deployment time, the exported .jar files are installed on the specified VMs, just like the rest of the JCatascopia probes.

[2] http://linc.ucy.ac.cy/CELAR/jcatascopia/

myApplicationDescription *probe.java

```java
import java.util.HashMap;
import eu.celarcloud.jcatascopia.probepack.Probe;
import eu.celarcloud.jcatascopia.probepack.ProbeMetric;
import eu.celarcloud.jcatascopia.probepack.ProbePropertyType;

public class probe extends Probe {

    private static int DEFAULT_SAMPLING_PERIOD = 20;
    private static String DEFAULT_PROBE_NAME = "probe";

    public probe(String name, int freq) {
        super(name, freq);
        /* define metrics that will be collected as ProbeProperties.
         * addProbeProperty (int id, String name, ProbePropertyType type,String units, String description)
         * e.g. addProbeProperty(0,"cpuUsage",ProbePropertyType.DOUBLE,"%","Current system cpu usage");
         */
    }

    public probe() {
        this(DEFAULT_PROBE_NAME, DEFAULT_SAMPLING_PERIOD);
    }

    @Override
    public ProbeMetric collect() {
        HashMap<Integer, Object> values = new HashMap<Integer, Object>();/* add to HashMap the values for each defined metric e.g. values.put(0,71.32) */
        return new ProbeMetric(values);
    }

    @Override
    public String getDescription() {
        return "Probe description";
    }

    public static void main(String[] args) {
        probe p = new probe();
        p.activate();
    }
}
```

Figure 25