Assignment 2- Virtual Resource Management in OpenNebula
Team-9
Shreyas S(12476188), Petar G(12495255), Bogdan E(12497924)

Assignment 2.1

Experience with ONE API: Having worked previously on Loopback in Period 3 for Developing Services for the cloud at VU, the tutorials were intuitive enough and mostly easy to follow. Creating and managing VMs is much more straightforward compared to the complicated IBM toolchains used. Using a CLI offers a better level of understanding about developing the application. Only suggestion that we could point out would be to address at least the most common troubleshooting in the tutorials so that students do not have to waste time figuring out some nitty gritty or bug that has already been dealt with.

The account setup and login to DAS was done first. On logging in, open nebula was already set up but it used its default one location path instead of the one in the shared folder. So, the correct one location was prepended to the original path environment variable to get the expected *which onevm* output. Then we followed the steps in the tutorials to create a VM. The first time, the VM remained in Pending state which was fixed by changing the port number in wiki.cs.vu.nl tutorial or by using the template given in the demo file by the TA (as mentioned in discussions). We ended up using the same test template for the VM with our own startup script. Once the VM was running, we logged in via *ssh -Y* into the DAS and try to run the firefox command that would draw the display in DAS server and relay the keyboard and mouse inputs. However, we encountered the issue of MacOS not forwarding display and additional parameters to the final SSH which gave a GDK_BACKEND not found error on the headnode when we tried to run Firefox. So, we installed XQuartz and used Xterm to login via SSH to the headnode and XTerm successfully displayed the firefox.

The url shortener developed for assignment 1 was developed in Java. We had used IntelliJ IDE to develop and run the application which abstracted lot of underlying building process that could not be replicated via command line tools for Ubuntu. So, we refactored the code in eclipse into a Maven build for easier deployment and generated a war file. The war file was optimised for Apache Tomcat 9.0 via options in Eclipse. Then, the war file was uploaded to cloud so that it could be downloaded during VM contextualization while the startup script is being executed. We converted startup script that we wrote to base64 and included it in the VM template via variable *STARTUP_SCRIPT_BASE64*. The startup script without encoding has also been enclosed in the tar file. When the *onetemplate create test9.one* and *onetemplate instantiate $ID* commands are run, the startup script installs the required JDK, Tomcat 9, downloads the war file, places the war file in the Tomcat web-apps folder and also starts the server at port 9090. The base-url for the deployed app is http://10.100.0.66:9090/url-shortener/shortener which returns a list of shortened urls or ids. The base-url/shortened-url(eg: http://10.100.0.66:9090/url-shortener/shortener/1) redirects to the original page. In our example, we have already set up shortened-url 1 for google, 2 for youtube which can be viewed with the said firefox display. On startup, Tomcat automatically picks up the war file in the web-apps directory and exposes the API on the endpoint specified by the web.xml file. We tested and demonstrated the deployment of the url-shortener in the lab class with cURL all of which work as expected. (Note: *test16.one* is the actual template used for contextualization. setup.sh is the actual script that is used as startup script. onevmshow.txt is the output of the command *onevm show $VM_ID*)