# Assignment 2.1 - VM Contextualization

Katerina Chinnappan, Rohit Shaw, Kjell Zijlemaker

## Experience with ONE API

For some of us OpenNebula/VM Contextualization was completely new so it was a good skill to acquire. We had prior experience with DAS-4, so that environment was comfortable to work in. One of the challenging parts was to think of a way to extract the assigned IP to the VM and update the service files, allowing us to run the service at VM startup without having to update the IP every time. With some of us having worked with ONE API previously, the idea of making the VM persistent came into play and proved beneficial for our approach of having the REST service run with a front-end on a web browser.

## VM Contextualization

### Template Details

We have our `VM` template specified in the `vm_template` file. When instantiating the VM, it deploys the `REST.py` service on startup. This is done by specifying in the template variable `FILES` and pointing it to `file.sh` which executes at startup. We have thought about pre-assigning our VM an IP address to avoid auto-assignment, however we ran into a problem when the specified IP in the template would be already occupied or the network would be full. We came up with a workaround; have the VM be auto-assigned an IP, extract the IP, save it to a file and update our `REST.py` and `index.html` files inside the VM.

We had to instantiate the VM using `persistent` option in order to ensure that the VM is run from a fixed disk-image and all of the files and settings are saved in the VM at every log-out.

### file.sh

This shell script is executed when instantiating the VM. The assigned IP was extracted from the environmental variable `$hostname -I` and saved into a local variable in the shell script. The extracted IP was then saved to a local `.txt` file inside the VM. Then we `cd` into the VM directory containing all files for the `REST` service and start the `update_html.py` script which updates the host name in `index.html`. We have a `sleep` for 5 seconds defined in our shell script before starting the `REST` service in order to avoid concurrency. `REST.py` is then executed which is equivalent to deploying the `REST` service at startup.

### LAMP Server

In order to run the `URL shortener` program with a front-end on a web browser, it had to be deployed on a local server. This was done through an `Apache LAMP Server` which was set up inside the VM. The server was configured and a virtual host, pointing to the files of the REST service, was set up. The service could then be accessed through FireFox with the command `firefox -no-remote http://`*`VM_IP`*`/url.com/public_html`, without having to log into the VM.

Step-by-step instructions for deploying and using the *REST* service can be found in the `README` file included in the `.tar` file.