

UPPSALA UNIVERSITY



APPLIED CLOUD COMPUTING 1TD265 AUTUMN 2020

Assignment 2

– Automating the creation of cloud services

Copyright authors:

Henrik Schulze

1st November 2020

ABSTRACT

This computer assignment has two major tasks.

1. In Task 1, which is mandatory, you will familiarize yourself with the OpenStack command line clients (CLIs) and Python APIs for interacting with the Interface as a Software layer. You will also get introduced to the orchestration engine in OpenStack, "heat".
 - (a) In Task 1 – Part 1, you will study the effect of virtualization by doing a simple performance comparison between the virtual machine and the physical host.
 - (b) In Task 1 – Part 2, you will learn the basics of how to automatically configure instances and how to automate and orchestrate deployment of single resources as well as more complex stacks.
2. Task 2 is optional and consists of three parts.
 - (a) In Part 1 of Task 2, you will learn the basics of the OpenStack Python APIs that can be used to interact directly with the IaaS services Keystone (Identity), Glance (Image) and Nova (compute).
 - (b) In Part 2 of Task 2, you will write half a page mini-essay where you highlight some of the key issues that can hinder cloud interoperability and also suggest possible solutions to those issues.
 - (c) In Part 3 of Task 2, you will design and explain a highly available cloud native application architecture for CSaaS (CowSay-as-a-Service).

1 Task-1. OpenStack command line clients (CLIs), Python APIs, and orchestration

1.1 Part-1. Performance comparison between the VM and the physical host

Access your virtual machine and execute the following command:

```
time echo "scale=5000; a(1)*4" | bc -l
```

The command will calculate the value of PI up to 5000 digits. It is a CPU intensive task. Record the time taken to execute the command. The time on the physical machine is UPPMAX: 20.856 seconds.

Question:

Explain your findings in part 1. Are virtual machines slower than the physical machine? If yes, explain the reason. Are there alternatives to virtual machines? How do you compare them with virtual machines? Keep the answer fairly short, limit it to a few sentences, max $\frac{1}{4}$ of a page.

Answer.

I log in to my virtual machine¹ [1]:

```
$ ssh-keygen -R 130.238.29.11 && ssh ubuntu@130.238.29.11 (1)
```

My virtual machine is a little more than half a second slower – taking about 21.56 seconds on average, so it is about 3% slower. The reason it is slower is that there is always an overhead when running a virtual machine compared to running a physical or *bare-metal* machine. Therefore, the obvious alternative to running a virtual machine is of course to run a physical machine. “The typical experience [...] on a bare metal\ Type 1 Hypervisor is around 1-5% of CPU overhead and 5-10% Memory overhead” [2].

A natural question then is:

why would you ever prefer a virtual machine to a physical, given that it always performs worse?

The answer is that there can be a number of benefits running virtual machines, as follows [3].

- a) Multiple OS environments can exist simultaneously on the same machine, isolated from each other.
- b) Virtual machine can offer an instruction set architecture that differs from a real computer.
- c) Easy maintenance, application provisioning, availability and convenient recovery.

¹ If you ever encounter *** System restart required *** then just ignore it.

1.2 Part-2. Basic contextualization and orchestration [34]

1.2.1 Task-0. Setting the environment for API access

Install Openstack libraries on your test virtual machine.

1. Goto <https://docs.openstack.org/install-guide/environment-packages-ubuntu.html>, <https://docs.openstack.org/python-openstackclient/latest/> and download the client tools and API for OpenStack.

Answer.

I will deal with this later. *Stay tuned!*

2. Log in to <https://east-1.cloud.snic.se/> and choose project UPPMAX. In the openstack dashboard, download the Runtime Configuration (RC) file:
Project > API Access > Download OpenStack RC File > OpenStack RC File (Identity API v3)

Answer.

I download it to my laptop, and then check and update the script. (See below.)

3. Confirm that your RC file has the following environment variables:

```
export OS_USER_DOMAIN_NAME="snic"  
export OS_IDENTITY_API_VERSION="3"  
export OS_PROJECT_DOMAIN_NAME="snic"
```

Answer.

The row saying `export OS_PROJECT_DOMAIN_NAME="snic"` was missing so I added it. I have also noted that the script says `export OS_IDENTITY_API_VERSION=3` (without quotation marks). Don't know if it matters. Next I upload it to my student home page, so that I can download it from there when I need it in my virtual machine.

4. Set the environment variables by sourcing the RC-file:

```
$ source <project\_name>\_openrc.sh
```

Answer.

I log in to my virtual machine by running command (1). Since I need a way to get the shell script to my virtual machine, I copy it, using a (Windows Subsystem for) Linux terminal:

```
$ scp openrc.sh ubuntu@130.238.29.11:.. (2)
```

Then I run the script:

```
$ source ~/openrc.sh (3)
```

The problem now is that I get the message:

Please enter your OpenStack Password for project UPPMAX 2020/1-2 as user s16996.

So what password is that? It turns out I can set it at <https://cloud.snic.se/>, in the left pane:

Services > Set your API password.

At this point I can confirm that the password is missing: `echo $OS_PASSWORD_INPUT`

(Here is what to do in order to get a fresh copy of `~/.bashrc`:

```
curl -H 'Cache-Control:no-cache' http://user.it.uu.se/~hesc0353/acc/.bashrc -o ~/.bashrc
```

The command above makes sure that you don't get a cached version of the file [4].)

5. Run the following commands and explain the output:

```
$ openstack server list (4)
```

```
$ openstack image list (5)
```

Answer.

Which of the commands I run first does not matter. The output when running command (5) is:

Command 'openstack' not found, but can be installed with:

```
sudo snap install openstackclients # version train, or
```

```
sudo apt install python-openstackclient
```

```
sudo apt install python3-openstackclient
```

so I install the `openstackclient` by running:

```
$ sudo apt -y install python3-openstackclient # takes about 2 minutes (6)
```

If this results in an error message ending with:

```
E: Unable to locate package python3-openstackclient,
```

then I update the advanced package tool (APT), the main command-line package manager for Debian and its derivatives [5]:

```
$ sudo apt update # takes about ten seconds (7)
```

and then I re-run command (6). Running command (6) corresponds to the very first task in this section: "download the client tools [...] for OpenStack". – where I ask the reader to *stay tuned*².

Of course, I then re-run command (5). If this gives me an error message:

```
The request you have made requires authentication. (HTTP 401) (Request-ID: [...])
```

```
– or – the error message: Missing value auth-url required for auth plugin password,
```

```
– or – the error message: KeyError: 'OS_PASSWORD', then I re-run the authentication script, command (3).
```

Then, of course, I re-run command (5) once again.

² The `openstackclient` includes a lot of sub-packages, among others the `novaclient` and the `keystoneclient`, which are the only two packages necessary for running contextualization. (See section 1.2.2.)

Questions on Task-0. Setting the environment for API access.

1. What version of the API are we using?

Answer.

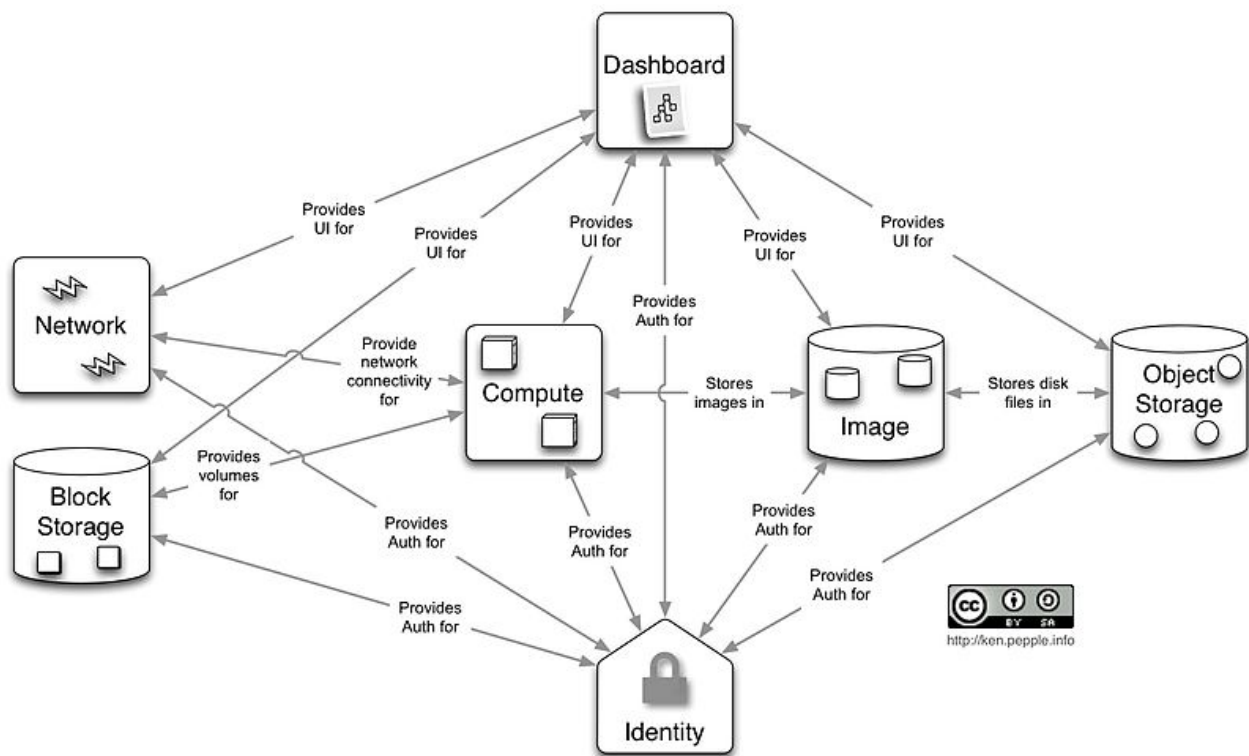
I am using Identity API v3.12 (Stein) [8].

2. Explain how openstack API services communicate with other OpenStack services.

Answer.

Through OpenStack APIs, services can communicate with each other. A service may call another service with similar characteristics, as long as the form of communication is respected [9].

The reference I have found includes an informative schematic diagram on a conceptual level how the services interact with each other [10].



3. Can we use the EC2 and S3 APIs to communicate with OpenStack?

Answer.

Yes, OpenStack can communicate with the APIs of both EC2 and S3 [11] [12].

1.2.2 Task-2. Single-machine contextualization

NOTE: The code assumes Ubuntu virtual machines.

The code for this task is in the folder `automation/contextualization`.

First make sure that there are no dirty versions of the files in the folder `/tech-tr/`:

```
$ rm -fr ~/tech-tr/ (8)
```

Now, to get the code into the virtual machine, download the Git repository:

```
$ git clone https://github.com/SNICScienceCloud/technical-training/ ~/tech-tr/ (9)
```

The following link provides an introduction to Cloud-init:

<https://help.ubuntu.com/community/Cloud-init>

In this task, from your primary virtual machine, you will prepare a single instance on a secondary virtual machine to install packages and start a web-service "cowsay" on that instance. The configuration is set by using the Cloud-init package. Run the code.

Answer.

I start out by naïvely running the code. First I list the contents:

```
$ ls -l ~/tech-tr/automation/contextualization/cloudinit/add-userdata/
```

The output is something like:

```
-rw-rw-r- 1 ubuntu ubuntu 745 <Month, Day, Time> cloud-cfg.txt
-rw-rw-r- 1 ubuntu ubuntu 2103 <Month, Day, Time> ssc-instance-userdata.py
```

Now I blindfoldedly run the code, optimistically hoping for the best:

```
$ python3 ~/tech-tr/automation/contextualization/cloudinit/add-userdata/ssc-instance-userdata.py
```

The output (almost) is:

```
File "~/tech-tr/automation/contextualization/cloudinit/add-userdata/ssc-instance-userdata.py", line 29
  print "user authorization completed."
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("user authorization completed.")?
```

The reason for the syntax error is that the code is written for Python 2, but I am using Python 3. Every string following the `print` command needs to be surrounded by parentheses – in the case of line 29, the code should be changed to:

```
print("user authorization completed.")
```

and so on for all other similar occurrences.

However, I prefer not to make any changes in this directory.

Instead I create a new folder and *copy* the two files to it:

```
$ mkdir ~/contextualization (10)
```

```
$ cp ~/tech-tr/automation/contextualization/cloudinit/add-userdata/* ~/contextualization/  
$ ls -l ~/contextualization/
```

The same kind of syntax error as on line 29 also occurs on lines 51, 54, 58, 63 of `ssc-instance-userdata.py`. Also, on line 52, I change `vm1` to `ctxHenk`, and add an extra argument `, key_name="HenkeKey-200922WSL"` to the call of `nova.servers.create()`.

It turns out that the file `cloud-cfg.txt` also has compatibility issues with Python 3. Thus, on lines 7, 8, 31, I change `python` to `python3`, and on line 30 I change `pip` to `pip3`. Also, on line 22 I replace `cowsay` by `/usr/games/cowsay` in order to get the full path included when calling the service.

I could do these changes directly in the virtual machine, for example by opening the file(s) in the nano text editor:

```
$ nano ~/contextualization/ssc-instance-userdata.py
```

However, I prefer to make the necessary changes on my local machine, and then copy the files over to the virtual machine. Thus, from the folder where my modified versions of the files `cloud-cfg.txt` and `ssc-instance-userdata.py` reside, using a (Windows Subsystem for) Linux terminal, *from my laptop* I run:

```
$ scp cloud-cfg.txt ubuntu@130.238.29.11:contextualization/. (11)
```

```
$ scp ssc-instance-userdata.py ubuntu@130.238.29.11:contextualization/. (12)
```

Note: for commands (11) and (12) to through, it may be necessary to first run command (??), just as I previously did when logging in to the virtual machine.

The configuration file `cloud-cfg.txt` will prepare the secondary instance at boot time. Read and try to understand `cloud-cfg.txt` in the folder `contextualization/cloudinit/add-userdata/`.

The file `ssc-instance-user-data.py` in the same folder is used to start the instance. Open the file, try to understand it. The python file is not complete, make the necessary changes, run the file and see if your instance is created.

Answer.

Having copied the two files `cloud-cfg.txt` and `ssc-instance-userdata.py` from my personal computer by running commands (11) and (12), I now run:

```
$ python3 ~/contextualization/ssc-instance-userdata.py (13)
```

Once again I try re-running command (13). If I get a python error message of about 30 lines, where the last line starts with:

```
keystoneauth1.exceptions.http.Unauthorized: The request you have made [...]  
– or – the last line says:  
KeyError: 'OS_AUTH_URL'
```

then I re-run the authentication script, command (3). I get an error message:

Traceback (most recent call last):

```
File "/home/ubuntu/contextualization/ssc-instance-userdata.py", line 31, in <module>
image = nova.glance.find_image(image_name)
File "/home/ubuntu/.local/lib/python3.6/site-packages/novaclient/v2/images.py", line 58, in find_image
raise exceptions.NotFound(404, msg)
novaclient.exceptions.NotFound: No Image matching None. (HTTP 404)
```

This suggests that something is wrong on line 31 in `ssc-instance-userdata.py`. Line 31 reads `image = nova.glance.find_image(image_name)`, which is fine really. But the `image_name` is set on line 15: `image_name = None` which clearly causes the error message saying No Image matching None. By inspecting the openstack dashboard at Project > Compute > Instances > henke, I can tell that the **Image Name** is `Ubuntu 18.04`. I therefore replace line 15 in `ssc-instance-userdata.py` with `image_name = "Ubuntu 18.04"` and then re-run commands (12) and (13).

This time the error message is No Flavor matching `{'name': 'ACHT18.normal'}`. (HTTP 404). From the openstack dashboard at Project > Compute > Instances > henke, I can tell that the **Flavor Name** is `ssc.xsmall`, so I replace `"ACHT18.normal"` on line 11 of `ssc-instance-userdata.py` with `"ssc.xsmall"`, and re-run commands (12) and (13).

I now get an error message saying `private-net` not defined, so again in `ssc-instance-userdata.py` I replace line 12 with `private_net = "UPPMAX 2020/1-2 Internal IPv4 Network"`, where I got the name from the openstack dashboard, Project > Network > Networks, column Name. While I am at it, I also set the `floating_ip_pool_name` on line 13 to `"Public External IPv4 Network"`³.

This time I get the message: `cloud-cfg.txt` is not in current working directory.

I therefore replace line 2 in `ssc-instance-userdata.py` with:

```
import datetime,os,pathlib,sys,time
```

I replace line 43 in `ssc-instance-userdata.py` with:

```
cfg_file_path = pathlib.Path(__file__).parent.absolute() / 'cloud-cfg.txt'
```

I replace line 47 in `ssc-instance-userdata.py` with:

```
sys.exit("The file '" + str(cfg_file_path) + "' is missing.")
```

And I replace line 50 in `ssc-instance-userdata.py` with:

```
print(str(datetime.datetime.now())[0:19])
```

Finally, when I now re-run commands (12) and (13) I can see in the SNIC openstack dashboard that the virtual machine `ctxHenk` has been created. It is a good idea to take a look at the log of the machine in the dashboard. After 20 seconds it will likely start to show rows containing `cloud-init`. See the figure on the next page.

³ The assignment of `floating_ip_pool_name` does not seem to have any effect though – the virtual machine `ctxHenk` must still be manually associated with a floating IP in the openstack dashboard.

```

[ 20.465151] cloud-init[679]: Cloud-init v. 19.4-33-gbb4131a2-0ubuntu1-18.04.1 running 'init' at Sun, 25 Oct 2020 16:02:22 +0000. Up 20.27 seconds.
[ 20.484363] cloud-init[679]: ci-info: *****Net device info*****
[ 20.492399] cloud-init[679]: ci-info: +-----+
[ 20.498185] cloud-init[679]: ci-info: | Device | Up | Address | Mask | Scope | Hw-Address |
[ 20.503536] cloud-init[679]: ci-info: +-----+
[ 20.508911] cloud-init[679]: ci-info: | ens3 | True | 192.168.2.113 | 255.255.255.0 | global | fa:16:3e:a9:e6:f2 |
[ 20.513858] cloud-init[679]: ci-info: | ens3 | True | fe80::f816:3eff:fea9:e6f2/64 | . | link | fa:16:3e:a9:e6:f2 |
[ 20.520077] cloud-init[679]: ci-info: | lo | True | 127.0.0.1 | 255.0.0.0 | host | . |
[ 20.525245] cloud-init[679]: ci-info: | lo | True | ::1/128 | . | host | . |
[ 20.530714] cloud-init[679]: ci-info: +-----+
[ 20.535680] cloud-init[679]: ci-info: *****Route IPv4 info*****
[ 20.540802] cloud-init[679]: ci-info: +-----+
[ 20.544923] cloud-init[679]: ci-info: | Route | Destination | Gateway | Genmask | Interface | Flags |
[ 20.549110] cloud-init[679]: ci-info: +-----+
[ 20.558305] cloud-init[679]: ci-info: | 0 | 0.0.0.0 | 192.168.2.1 | 0.0.0.0 | ens3 | UG |
[ 20.563024] cloud-init[679]: ci-info: | 1 | 169.254.169.254 | 192.168.2.1 | 255.255.255.255 | ens3 | UGH |
[ 20.567697] cloud-init[679]: ci-info: | 2 | 192.168.2.0 | 0.0.0.0 | 255.255.255.0 | ens3 | U |
[ 20.572841] cloud-init[679]: ci-info: +-----+
[ 20.578134] cloud-init[679]: ci-info: *****Route IPv6 info*****
[ 20.582577] cloud-init[679]: ci-info: +-----+
[ 20.592111] cloud-init[679]: ci-info: | Route | Destination | Gateway | Interface | Flags |
[ 20.597362] cloud-init[679]: ci-info: +-----+
[ 20.603935] cloud-init[679]: ci-info: | 1 | fe80::/64 | :: | ens3 | U |
[ 20.612408] cloud-init[679]: ci-info: | 3 | local | :: | ens3 | U |
[ 20.620141] cloud-init[679]: ci-info: | 4 | ff00::/8 | :: | ens3 | U |
[ 20.628088] cloud-init[679]: ci-info: +-----+
[ 24.961219] cloud-init[679]: Generating public/private rsa key pair.
[ 24.968425] cloud-init[679]: Your identification has been saved in /etc/ssh/ssh_host_rsa_key.
[ 24.976098] cloud-init[679]: Your public key has been saved in /etc/ssh/ssh_host_rsa_key.pub.
[ 24.980093] cloud-init[679]: The key fingerprint is:
[ 24.984094] cloud-init[679]: SHA256:hYL08s5M3E]+0/n+82Zfx9BYVu2zF0qYqUQ1BHroMLU root@vmhenke
[ 24.992115] cloud-init[679]: The key's randomart image is:
[ 24.996129] cloud-init[679]: +---[RSA 2048]-----+
[ 25.000136] cloud-init[679]: | o= . . o. o |
[ 25.004095] cloud-init[679]: | B+o+ +. o .o|
[ 25.008111] cloud-init[679]: | *+=E..o . .+|
[ 25.016136] cloud-init[679]: | o++o . B. |
[ 25.018282] cloud-init[679]: | +.S = + + |
[ 25.024398] cloud-init[679]: | o . + . o |
[ 25.026660] cloud-init[679]: | o . o |
[ 25.032385] cloud-init[679]: | . . o .o |
[ 25.036086] cloud-init[679]: | . . = . . |
[ 25.040084] cloud-init[679]: +----[SHA256]-----+
[ 25.044103] cloud-init[679]: Generating public/private dsa key pair.
[ 25.052139] cloud-init[679]: Your identification has been saved in /etc/ssh/ssh_host_dsa_key.
[ 25.056118] cloud-init[679]: Your public key has been saved in /etc/ssh/ssh_host_dsa_key.pub.

```

I associate with this new machine an available floating IP. – In the openstack dashboard at Project > Compute > Instances, under the rightmost column Actions, in the drop-down menu I choose Associate Floating IP.

Test that the server is working by executing the following curl command:

```
$ curl -i http://<your_public_ip>:5000/cowsay/api/v1.0/saysomething
```

If you are using Windows, use for example Git Bash [13].

Answer.⁴

The first thing to note is that *it takes time for the service to become available*. For me it mostly took between six and seven minutes. As I understand it, such a start-up time is quite normal. [14] After those minutes have passed, a good idea is to enter the openstack dashboard and check the log. To do this, I go to Project > Compute > Instances and click the name of the newly created machine, and then on the Log tab. A good sign is if the last two lines are similar to:

```
[ 277.500087] cloud-init[1041]: * Debugger is active!  
[ 277.510836] cloud-init[1041]: * Debugger PIN: 247-780-808
```

where I disregard any lines that are responses to client requests.

On the other hand, if the last few lines have contents like:

```
Ubuntu 18.04.4 LTS ctxhenk ttyS0  
ctxhenk login:
```

– or –

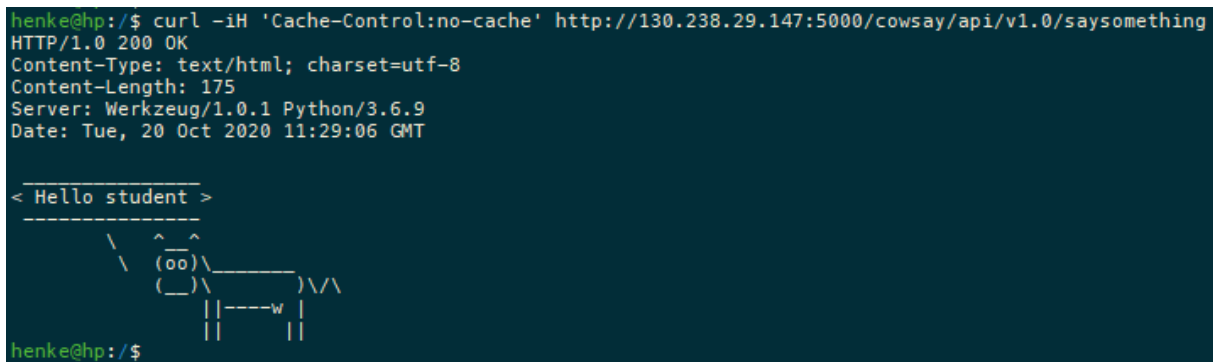
```
[ 28.392075] cloud-init[1024]: Cloud-init v. 19.4-33-gbb4131a2-0ubuntu1~18.04.1
```

then consider this to be a rather strong indication that something is wrong in the cloud configure script. It deserves to be pointed out that the cloud configure script *must* start with the 13 characters `#cloud-config` – nothing else.

If the log is looking promising, in a (windows subsystem for) linux terminal, I type:

```
$ curl -iH 'Cache-Control:no-cache' http://130.238.29.147:5000/cowsay/api/v1.0/saysomething (14)
```

which displays



```
henke@hp:/$ curl -iH 'Cache-Control:no-cache' http://130.238.29.147:5000/cowsay/api/v1.0/saysomething  
HTTP/1.0 200 OK  
Content-Type: text/html; charset=utf-8  
Content-Length: 175  
Server: Werkzeug/1.0.1 Python/3.6.9  
Date: Tue, 20 Oct 2020 11:29:06 GMT  
  
< Hello student >  
-----  
      \   ^__^  
       (oo)\_____  
        (__)\       )\/\  
           ||----w |  
           ||     ||  
henke@hp:/$
```

⁴ At some point it might become necessary to start a completely new *primary* virtual machine and do everything all over *from scratch*. The commands I need to run in order to get back on track again with single-machine contextualization are (1), (2), (10), (11), (12), (7), (6), (3), and (13) – in this given order.

Questions on Task-2. Single-machine contextualization.

1. Explain the output.

Answer.

The output is the result of a python program that creates an instance from an Ubuntu image. By instructions in a Cloud-init configuration script, the cowsay package is installed, and the service is started.

2. What is contextualization?

Answer.

According to Oxford Advanced Learner's Dictionary, *contextualization* is *the process of considering something in relation to the situation in which it happens or exists* [15]. In cloud computing, the term *contextualization* was coined in 2008 in a paper by Keahey and Freeman. Their aim was to *provide a generic, lightweight, and automated mechanism that will quickly deploy fully configured images and adapt them to their deployment context* [16]. What it means to put a newly created virtual machine *in context* includes for example to add on a network and some application software, or to assign a public IP [17].

3. What language is used to prepare the Cloud-init configurations?

Answer.

I have used *Cloud Config Data* for the configurations above [18].

4. What are some alternatives to the Cloud-init package?

Answer.

Some alternatives to Cloud-init are Ansible, Puppet, Rudder, SaltStack, and Jenkins, among others [19].

5. Can we run Cloud-init scripts without booting an instance?

Answer.

Yes, it is possible to run cloud-init manually without booting [20].

6. What limitation can you anticipate with the Cloud-init package?

Answer.

Not quite sure what is asked for here, but to the best of my understanding, there seem to be some concerns about Cloud-init when it comes to authentication and security [21].

1.2.3 Task-4. Orchestration using Heat

In this task you will create a cluster of two machines using the Heat engine. Heat is OpenStack's native orchestration engine and will let you automate the deployment of complex resources/stacks. The cluster will be completely customized by having its own network, security group, router settings and resources. Complete the following steps to provide the required information in the template file:

1. Enter your personal key name by replacing the key section's default value.

Answer.

First list the contents of the folder:

```
$ ls -l ~/tech-tr/automation/orchestration/heat-template/
```

The output is something like:

```
-rw-rw-r- 1 ubuntu ubuntu 3950 <Month, Day, Time> ssc-test-stack.yml
```

As before, I prefer not to make any changes in this directory, but instead create a new folder:

```
$ mkdir ~/orchestration
```

Again, I prefer to make the necessary changes on my local machine, and then copy the file over to the virtual machine.

I therefore open `ssc-test-stack.yml` in a text editor.

Under `parameters > image`, I look at line 19, and then into the openstack dashboard, at `Project > Compute > Images` I click on the image named `Ubuntu 18.04` (on its name), which shows that the ID of the image is `0b7f5fb5-a25c-48b6-8578-06dbfa160723`. Thus, on line 19 of `ssc-test-stack.yml` I replace the existing value with the one from the dashboard.

On line 25 I replace `ACHT18.normal` with `ssc.xsmall`.

On line 31 I replace `<-MY_KEY_NAME->` with `HenkeKey-200922WSL`, which I found in the dashboard at `Project > Compute > Key Pairs`.

On line 37 I leave the default value as `Public External IPv4 Network`.

Now, from the folder where my modified version of the file `ssc-test-stack.yml` resides, using a (Windows Subsystem for) Linux terminal, *from my laptop* I run:

```
$ scp ssc-test-stack.yml ubuntu@130.238.29.11:orchestration/. (15)
```

In the terminal of my virtual machine I check that the contents of the file look OK:

```
$ cat ~/orchestration/ssc-test-stack.yml (16)
```

-
2. In your primary virtual machine, generate an SSH key-pair: `$ ssh-keygen -t rsa`.

3. Replace `<ADD-CLUSTER's-PUBLIC-KEY>` with the public part of the generated key pair. Replace it for both of the instances.

Answer.

I follow these two instructions. After running `$ ssh-keygen -t rsa`, I can retrieve the value of the public key by displaying the contents of the `id_rsa.pub` file in the terminal:

```
$ cat ~/.ssh/id_rsa.pub
```

Then I make the requested changes on lines 103 and 123 and re-run the command (15) – and then command (16) to check that the changes went through.

4. Run the command:

```
openstack stack create stack_with_init_script -f 'yaml' -t ssc-test-stack.yml
```

Answer.

I run:

```
openstack stack create stackHenke -f 'yaml' -t ~/orchestration/ssc-test-stack.yml (17)
```

As a result. I get an error message:

```
openstack: 'stack create [...]' is not an openstack command. See 'openstack -help'.
```

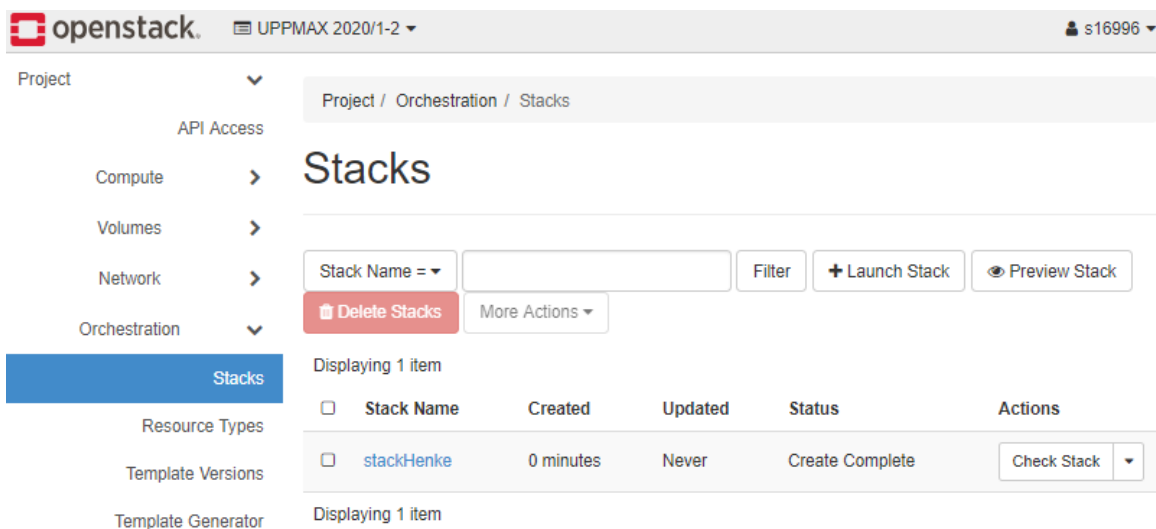
The reason for this error message is that the openstack heatclient has not yet been installed. So I do that now:

```
$ sudo -y apt install python3-heatclient # takes about ten seconds  
and then I re-run command (17).
```

5. Open the SSC dashboard, click on the Orchestration and check the status of your stack.

Answer.

It works fine, as can be seen in the screenshot below.



Questions on Task-4. Orchestration using Heat.

1. What language is used with the Heat service to define the templates?

Answer.

The language used is **YAML**.

YAML is a human friendly data serialization standard for all programming languages [22].

2. What are the advantages of using templates rather than the APIs?

Answer.

Templates allow creation of instances, floating IPs, volumes, security groups, users, as well as some more advanced functionality such as instance high availability, instance autoscaling, and nested stacks [23]. Templates can also specify the relationships between for example a volume and a server. This enables the orchestration tool to call out to the Cloud APIs to create infrastructure in the correct order to completely launch an application [24]. Another advantage with a template is that – being a declarative script – it is just an easily managed text file [17] that allows versioning.

3. Explain the different sections in the template.

Answer.

A Heat template has three major sections.

parameters:

These are tidbits of information – like a specific image ID, or a particular network ID – that are passed to the Heat template by the user. This allows users to create more generic templates that could potentially use different resources [25].

resources:

Resources are the specific objects that Heat will create and/or modify as part of its operation [25].

outputs:

The output is information that is passed to the user, either via the OpenStack Dashboard or via the `heat stack-list` and `heat stack-show` commands [25].

1.2.4 Task-5. Introduction to Linux Containers

This task will introduce Linux containers. There are different technologies available but in this Lab, we will focus on Docker containers. Your task is to build and run CSaaS service using Docker containers.

Step-1: Install Docker on your virtual machine.

0 - Switch to the root user:

```
$ sudo bash
```

Answer.

I decline this advice. The reason is that it may backlash in the configuration script when contextualizing a container. Compared to the instructions I will consequently replace all occurrences of # with \$ sudo.

1 - First, add the GNU Privacy Guard (GPG) key for the official Docker repository to the system:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

2 - Add the Docker repository to APT sources:

```
$ sudo add-apt-repository "deb [arch=amd64] \
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" [26]
```

3 - Update the package database with the Docker packages from the newly added repo:

```
$ sudo apt update
```

4 - Install Docker:

```
$ sudo apt install -y docker-ce
```

5 - (Optional) Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

```
$ systemctl status docker
```

Answer.

Yes, I did all of the above. – Thanks for the unusually clear and detailed instructions.

(Sadly as usual though, it turns out that once again the instructions are incomplete. Read on!)

Below is the start of the output from the command \$ systemctl status docker – so far so good.

```
docker.service - Docker Application Container Engine
  Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
  Active: active (running) since Mon 2020-10-12 17:14:40 UTC; 19h ago
    Docs: https://docs.docker.com
 Main PID: 26128 (dockerd)
   Tasks: 15
  CGroup: /system.slice/docker.service
          └─26128 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
          └─31488 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 5000 [...]
```

For more information visit:

<http://docker.com>

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04>

Step - 2: Use the Dockerfile in the repository (the container folder) to build a container.

0 - In case the Git repository is not on my virtual machine, I need to download it by running commands (8) and (9) in the beginning of section 1.2.2.

Answer.

Before I move on, I need to consider `~/tech-tr/automation/container/Dockerfile`. I have made quite a few changes on rows 2-8. It would be too tedious to go through the changes one by one.

The contents of Dockerfile now are:

```
FROM ubuntu
RUN apt update
RUN apt -y upgrade
RUN apt install -y git
RUN apt install -y python3-pip
RUN pip3 install --upgrade pip
RUN pip3 install flask
RUN apt install -y cowsay
RUN git clone https://github.com/TDB-UU/csaas.git
WORKDIR /csaas/cowsay
EXPOSE 5000
ENV PATH="${PATH}:/usr/games/"
CMD ["python3", "app.py"]
```

As before, I get the file on to the virtual machine by copying it *from my laptop*:

```
$ scp Dockerfile ubuntu@130.238.29.11:tech-tr/automation/container/.
```

To check that the file was successfully transferred, in the virtual machine I write out the contents in the terminal:

```
$ cat ~/tech-tr/automation/container/Dockerfile
```

1 - Go to the container directory:

```
$ cd ~/tech-tr/automation/container
```

2 - Execute the docker build-image by running the command:

```
# docker image build --no-cache -t cowsay:latest .
```

Answer.

I prefer to change this command slightly [27]:

```
$ sudo docker image build --no-cache -t cowsay:latest ~/tech-tr/automation/container/. (18)
```

The error message I get (after about 65 seconds) is:

```
Err:1 http://archive.ubuntu.com/ubuntu focal InRelease
```

It turns out that this problem has to do with the maximum transmission unit (MTU for short).

Inspired by Jon Moore [28], I run:

```
$ ping -c 2 -M do 8.8.8.8 -s 1501
```

```
ping: local error: Message too long, mtu=1450
```

This shows that the MTU for this cloud environment is 1450. I therefore run:

```
$ echo "{\"mtu\":1450}" | sudo tee -a /etc/docker/daemon.json
```

and then as a check:

```
$ cat /etc/docker/daemon.json
```

```
{"mtu":1450}
```

Before I re-run command (18) – unless I want the same error to occur again – I need to restart the docker service:

```
$ sudo service docker restart
```

If there are no errors, the command will take about 5 minutes to complete.

If it is successful, the last two lines in the output will be:

```
Successfully built 59194bc62420
```

```
Successfully tagged cowsay:latest
```

where 59194bc62420 is the IMAGE ID of the newly built docker image.

The IMAGE ID can be retrieved by running [29]:

```
$ sudo docker image list
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
cowsay	latest	59194bc62420	10 minutes ago	457MB
ubuntu	latest	9140108b62dc	3 weeks ago	72.9MB

3 - Choose **one** of the following two commands. *Only one of them should be run.* Which one?

```
$ sudo docker container run -it cowsay
```

```
$ sudo docker container run -d -p 5000:5000 cowsay
```

Answer.

Choose the second one: '\$ sudo docker container run -d -p 5000:5000 cowsay'. By running:

```
$ sudo docker container run --help
```

```
-d, --detach Run container in background and print container ID
```

```
-p, --publish list Publish a container's port(s) to the host
```

it should be clear that the second command makes more sense.

Here is how to retrieve the list of currently running containers:

```
$ sudo docker container list
```

CONTAINER ID	IMAGE	COMMAND	CREATED
80a2be83a229	cowsay	"python3 app.py"	13 minutes ago

Step - 3: Test that the service is available by executing – from a client:

```
$ curl -i http://<your_public_ip>:5000/cowsay/api/v1.0/saysomething
```

Answer.

Yes. From my laptop I re-run a slight variation of command (14):

```
$ curl -iH 'Cache-Control:no-cache' http://130.238.29.11:5000/cowsay/api/v1.0/saysomething
```

and get the same kind of greeting from the cow as in section 1.2.2. I conclude that the Cowsay-as-a-Service works just fine.

Step - 4: Some optional cleaning up before running a contextualized docker container :

When I want to stop the service, I run:

```
$ sudo docker container list
```

to get the CONTAINER ID, and then:

```
$ sudo docker container stop 80a2be83a229
80a2be83a229
```

If I want to remove a stopped container, I run:

```
$ sudo docker container ps -a (19)
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
80a2be83a229	cowsay	"python3 app.py"	2 hours ago	Exited

followed by:

```
$ sudo docker container rm 80a2be83a229
80a2be83a229
```

Suppose I want to remove a docker image altogether.

First of all I try to convince myself that no containers are using that image – command (19).

Then I run:

```
$ sudo docker image list
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
cowsay	latest	59194bc62420	2 hours ago	457MB
ubuntu	latest	9140108b62dc	3 weeks ago	72.9MB

followed by:

```
$ sudo docker image rm 59194bc62420
```

Untagged: cowsay:latest

Deleted: sha256:59194bc6242087fb0724551631ea64ee1702455fdfoe4bd3dd19f7791982191b

Deleted: sha256: ...

Deleted: sha256:7956295feea4f309391266bcd5f40a45611913d9671e6d6de3a94ee9ec88b910

A control check:

```
$ sudo docker image list
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	9140108b62dc	3 weeks ago	72.9MB

Questions on Task-5. Introduction to Linux Containers.

1. In what category of virtualization do containers fall?

Answer.

Containers belong to the category *container-based virtualization* [30].

2. What frameworks, other than docker, provide container technology? Write at least two names.

Answer.

Some alternatives to Docker are LXC (Linux Containers), Hyper-V containers, Container runtime **rkt** (pronounced "rocket"), Podman, and more [31].

3. Explain the provided Dockerfile. What does it do? How does it work?
Write a brief (one line) description about each line in the Dockerfile.

Answer.

```
FROM ubuntu # Log in as user 'ubuntu'
RUN apt update # Run the command 'apt update'
RUN apt -y upgrade # Run the command 'apt -y upgrade'
RUN apt install -y git # Run the command 'apt install -y git'
RUN apt install -y python3-pip # Install 'python3-pip'
RUN pip3 install --upgrade pip # Upgrade 'python3-pip'
RUN pip3 install flask # Install 'flask' via pip3
RUN apt install -y cowsay # Install 'cowsay' via apt
RUN git clone https://github.com/TDB-UU/csaas.git # Download the 'Hello Student' repo
WORKDIR /csaas/cowsay # Change directory to /csaas/cowsay
EXPOSE 5000 # Expose port 5000 - to be published
ENV PATH="${PATH}:/usr/games/" # Set 'PATH' to include /usr/games
CMD ["python3", "app.py"] # Start running the cowsay service
```

4. Write a brief (one line) description about each command used in Step-2-2.

Answer.

By and large already done. See above. Command (18) builds the docker image.
The command `docker run -it cowsay` should be used to run the container locally [32].

5. What is Docker Hub? Write a brief description of how we could use Docker Hub for our newly built Cowsay as a Service container.

Answer.

Docker Hub is a service provided by Docker for finding and sharing container images with your team [33].

6. Write a Cloud-init script that contextualizes a virtual machine based on steps 1 and 2 in this task. Submit the script with your assignment report.

Answer.

It is **two** files; the `#cloud-config` script which I have named `dockerCtxt.txt` and the Python script `dockerCtxt.py` which calls the Cloud-init script. The source code of these two files is in the appendix.

In the primary virtual machine I create a new directory for the two files:

```
$ mkdir ~/docker-Task5-Q6 && chown ubuntu:ubuntu ~/docker-Task5-Q6
```

The second part of the command is necessary only if you are currently user `root`.

As I did in section 1.2.2, I copy the files from my laptop:

```
$ scp dockerCtxt.txt ubuntu@130.238.29.11:docker-Task5-Q6/. (20)
```

```
$ scp dockerCtxt.py ubuntu@130.238.29.11:docker-Task5-Q6/. (21)
```

Again, in case of `Permission denied`, I check in the virtual machine who is the owner of `~/docker-Task5-Q6` by running `'$ ls -dl ~/dock*'`. In case the owner of `~/docker-Task5-Q6` is `root` rather than `ubuntu`, I rectify this with `'$ sudo chown ubuntu:ubuntu ~/docker-Task5-Q6'`.

Then I re-run commands (20), and (21).

From my laptop I (re-)run command (14), and get the error message:

```
curl: (7) Failed to connect to 130.238.29.11 port 5000: Connection refused
```

This is expected – after all, I have not started the service yet. In fact, I have not even created the secondary virtual machine, nor have I built the docker image in it, and I have not started the docker container that is supposed to provide the `Cowsay-as-a-Service`. To do all these things will take about twice as long as the single-machine contextualization in Task 2 above, some 11-13 minutes.

To start the contextualization of a docker container, in my virtual machine I first run:

```
$ python3 ~/docker-Task5-Q6/dockerCtxt.py (22)
```

and get the error message:

```
File "/home/ubuntu/docker-Task5-Q6/dockerCtxt.py", line 22, in <module>
```

```
    auth = loader.load_from_options(auth_url=env['OS_AUTH_URL'],
```

This means that I need to re-run command (3) before I re-run command (22).

Finally, when I have re-run those two commands, I can see in the SNIC openstack dashboard that the virtual machine `dockHenk` has been created.

In the dashboard I manually add an available floating IP.

When I now re-run command (14), the cow once again greets me, just as it did in section 1.2.2.

If the log does not continue beyond 30 seconds, the reason could be that something is wrong in the `runcmd:` section. If Cloud-init fails at around 400 seconds, then the secondary virtual machine has likely been created, but the `docker image build` command could not find the docker file. ⁵

⁵ If the message in the openstack log is `[[0;1;31mFAILED[0m] Failed to start Execute cloud user/final scripts.`

References

- [1] “Is it possible to remove a particular host key from SSH’s known_hosts file?.”
<https://askubuntu.com/questions/20865#20869>
- [2] “Is virtual machine slower than the underlying physical machine?”
<https://serverfault.com/questions/135431#135491>
- [3] “Pros and cons of the virtual machines - How to access serial devices in VMs”
<https://www.serial-server.net/virtual-machine/>
- [4] “Refresh a proxy cache with Curl.”
<https://martin.hoppenheit.info/blog/2015/refresh-proxy-cache-with-curl/>
- [5] “Apt.”
<https://wiki.debian.org/Apt>
- [6] “List or get details for images (glance).”
<https://docs.openstack.org/glance/pike/admin/manage-images.html#list-or-get-details-for-images-glance>
- [7] “Search for an instance using IP address.”
<https://docs.openstack.org/ocata/user-guide/cli-search-instance-with-ip-address.html#search-for-an-instance-using-ip-address>
- [8] “OpenStack API Documentation.”
<https://docs.openstack.org/api-ref/identity/v3/#what-s-new-in-version-3-12-stein>
- [9] “In a nutshell: How OpenStack works.”
<http://vmartinezdelacruz.com/in-a-nutshell-how-openstack-works/>
- [10] “File:Openstack-conceptual-arch-folsom.jpg.”
<https://commons.wikimedia.org/wiki/File:Openstack-conceptual-arch-folsom.jpg>
- [11] “OpenStack EC2 API.”
<https://docs.openstack.org/ec2-api/latest/>
- [12] “Configure Object Storage with the S3 API.”
<https://docs.openstack.org/mitaka/config-reference/object-storage/configure-s3.html>
- [13] “Git for Windows.”
<https://gitforwindows.org/>
- [14] “Prebaked μ VMs: Scalable, Instant VM Startup for IaaS Clouds.”
<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7164911>
- [15] “contextualization.”
<https://www.oxfordlearnersdictionaries.com/search/english/?q=contextualization>
- [16] “Contextualization: Providing One-Click Virtual Clusters.”
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4736771>
- [17] “The Total Newbie’s Introduction to Heat Orchestration in OpenStack.”
<https://www.cisco.com/c/dam/en/us/products/collateral/cloud-systems-management/metacloud/newbie-tutorial-heat.pdf>
- [18] “Cloud Config Data.”
<https://help.ubuntu.com/community/Cloud-init#line-29>
- [19] “Best 8 Ansible Alternatives in 2020.”
<https://www.guru99.com/ansible-alternative.html>
- [20] “How to run cloud-init manually?”
<https://stackoverflow.com/questions/23151425/how-to-run-cloud-init-manually>
- [21] “vulnerability.”
<https://vigilance.fr/vulnerability/cloud-init-Man-in-the-Middle-via-ssh-deletekeys-32893>
- [22] “What is YAML? A Beginner’s Guide.”
<https://circleci.com/blog/what-is-yaml-a-beginner-s-guide/>
- [23] “Heat’s purpose and vision.”
<https://docs.openstack.org/heat/latest/>

- [24] “OpenStack Orchestration – How it works.”
<https://wiki.openstack.org/wiki/Heat>
- [25] “An Introduction to OpenStack Heat.”
<https://blog.scottlowe.org/2014/05/01/an-introduction-to-openstack-heat/>
- [26] “How to split long sed expression into multiple lines?”
<http://unix.stackexchange.com/questions/146955#146962>
- [27] “I keep getting the "docker build" requires exactly 1 argument(s) error.”
<https://stackoverflow.com/questions/41250326#48467039>
- [28] “Test MTU with ping.”
<http://jonmoore.duckdns.org/index.php/linux-articles/52-test-mtu-with-ping>
- [29] “docker image.”
<https://docs.docker.com/engine/reference/commandline/image/>
- [30] “What are containers (container-based virtualization or containerization)?”
<https://searchitoperations.techtarget.com/definition/container-containerization-or-container-based-virtualization>
- [31] “6 Alternatives to Docker: All-in-One Solutions and Standalone Container Tools.”
<https://jfrog.com/knowledge-base/6-alternatives-to-docker-all-in-one-solutions-and-standalone-container-tools/>
- [32] “Docker Command-Line Tips for Beginners.”
<https://spin.atomicobject.com/2018/10/04/docker-command-line/>
- [33] “Docker Hub Quickstart.”
<https://docs.docker.com/docker-hub/>
- [34] “Basic contextualization and orchestration.”
<https://github.com/SNICScienceCloud/technical-training/tree/master/automation>

Appendix

Listing 1: The contents of the Python file `dockerCtxt.py` which calls the Cloud-init script.

```
# Uppsala University - Applied Cloud Computing, 1TD265
# Copyright (c) 2020 Henrik Alfken(Schulze)
# Assignment C-2 - Automating the creation of cloud services
# Task 5, Question 6 - Contextualizing a docker container
# https://bit.ly/34aLE88
import datetime,os,pathlib,sys,time
import inspect
from os import environ as env

from novaclient import client
import keystoneclient.v3.client as ksclient
from keystoneauth1 import loading
from keystoneauth1 import session

flavor = "ssc.xsmall"
private_net = "UPPMAX 2020/1-2 Internal IPv4 Network"
floating_ip_pool_name = "Public External IPv4 Network"
floating_ip = None
image_name = "Ubuntu 18.04"

loader = loading.get_plugin_loader('password')

auth = loader.load_from_options(
    auth_url=env['OS_AUTH_URL'],
    username=env['OS_USERNAME'],
    password=env['OS_PASSWORD'],
    project_name=env['OS_PROJECT_NAME'],
    project_domain_name=env['OS_USER_DOMAIN_NAME'],
    project_id=env['OS_PROJECT_ID'],
    user_domain_name=env['OS_USER_DOMAIN_NAME'])

sess = session.Session(auth=auth)
nova = client.Client('2.1',session=sess)
print("user authorization completed.")

image = nova.glance.find_image(image_name)

flavor = nova.flavors.find(name=flavor)

if private_net != None:
    net = nova.neutron.find_network(private_net)
    nics = [{'net-id': net.id}]
else:
    sys.exit("private-net not defined.")

#print("Path at terminal when executing this file")
#print(os.getcwd() + "\n")
cfg_file_path = pathlib.Path(__file__).parent.absolute() / 'dockerCtxt.txt'
if os.path.isfile(cfg_file_path):
    userdata = open(cfg_file_path)
```



```
else:
    sys.exit("The file '" + str(cfg_file_path) + "' is missing.")

secgroups = ['default']
print(str(datetime.datetime.now())[0:19])
print("Creating instance ... ")
instance = nova.servers.create(name="dockHenk",image=image,flavor=flavor,
    userdata=userdata,nics=nics,security_groups=secgroups,
    key_name="HenkeKey-200922WSL")
inst_status = instance.status
print("waiting for 10 seconds.. ")
time.sleep(10)

while inst_status == 'BUILD':
    print("Instance: "+instance.name+" is in "+inst_status
        +" state, sleeping for 5 seconds more...")
    time.sleep(5)
    instance = nova.servers.get(instance.id)
    inst_status = instance.status

print("Instance: "+ instance.name +" is in " + inst_status + "state")
```

Listing 2: The contents of the Cloud-init script `dockerCtxt.txt` which creates the Docker container.

```
#cloud-config
# Uppsala University - Applied Cloud Computing, 1TD265
# Copyright (c) 2020 Henrik Alfken(Schulze)
# Assignment C-2 - Automating the creation of cloud services
# Task 5, Question 6 - Contextualizing a docker container

apt_update: true
apt_upgrade: true
packages:
- cowsay
- python3-pip
- python3-dev
- build-essential
- cowsay
byobu_default: system

write_files:
- path: /etc/docker/daemon.json
  content: |
    {"mtu":1450}
- path: ~/Dockerfile
  content: |
    FROM ubuntu
    RUN apt update
    RUN apt -y upgrade
    RUN apt install -y git
    RUN apt install -y python3-pip
    RUN pip3 install --upgrade pip
    RUN apt install -y cowsay
    RUN pip3 install flask
    RUN git clone https://github.com/TDB-UU/csaas.git
    WORKDIR /csaas/cowsay
    EXPOSE 5000
    ENV PATH="${PATH}:/usr/games/"
    CMD ["python3", "app.py"]

runcmd:
- echo "export PATH=$PATH:/usr/games" >> ~/.bashrc
- source ~/.bashrc
- sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
- sudo apt-key add -
- sudo add-apt-repository "deb [arch=amd64] \
- https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
- sudo apt update
- sudo apt install -y docker-ce
- sudo docker image build --no-cache -t cowsay:latest ~/.
- sudo docker container run -d -p 5000:5000 cowsay
```
