# Lab 4: DOCKER & python flask

## Learning Outcomes:

After completing this lab, you should be able to:

1. Build, run, modify, stop, move and restart a Docker Container.

2. Initialise Docker Swarm Cluster.

3. Run services in swarm cluster.

4. Monitor the swarm services.

## Organisation

Please attempt this lab individually as you will need this to be completed in order to complete subsequent labs.

* Part 1 to Part 6 are compulsory.
* You are given two weeks to complete this lab.
* **You will be required to demo it next week 27th of November 2018.**

## Part 1. Create DOCKER hosts

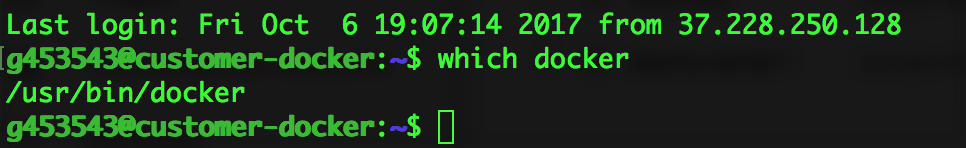
* 1. **You can use AWS EC2, or Qwiklabs Sandbox or Google Compute Engine to Complete this lab**
  2. Create Three Virtual machines based **Ubuntu Server 18.04 LTS** and name them: manager-<<yourStudentNo>>, worker-1, worker-2.
  3. Create a security group and call it: ***Docker-SG***
  4. open the following TCP and UDP ports.

*\*\* this step in google compute is accomplished using Firewall rules configurations.*

A screenshot of a cell phone 
 
Description automatically generated

* 1. Verify that Docker is installed in all VMs if not you need to install it as in Step 3:

which docker



* if docker is not installed, install it using the following command:

curl -fsSL get.docker.com|sh

* Verity that Docker is running with full privilege without sudo:

sudo docker --version

docker –version

* If you need sudo to run the docker commands, use the following command to run docker commands without sudo:

sudo usermod -aG docker <username>

* For example, the username in the figure above is: g453543, the command will be sudo usermod -aG docker g453543. In your case it should be your student number or ubuntu.
* **Make sure to restart the VM using the following command.**

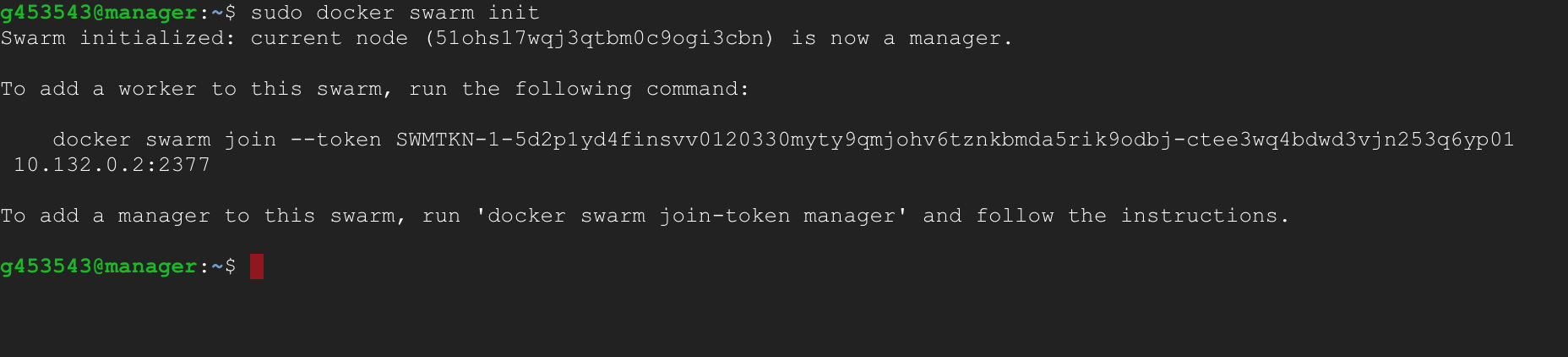
sudo reboot

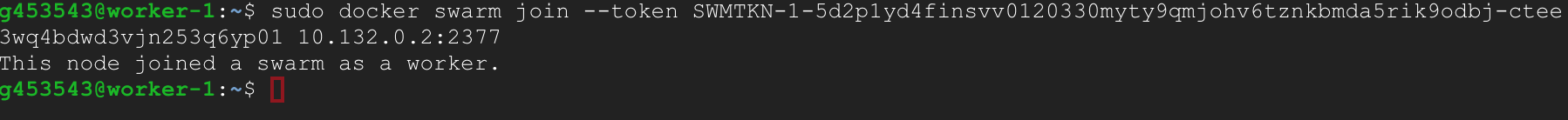
* 1. **Connect to the VM using SSH and check you can run docker commands without sudo**

# part 2: Learn Docker Containers

* 1. **Access the VM manager using SSH.**
  2. **Initiate the swarm manger using the following command. This will make this VM the manager of the swarm.**

sudo docker swarm init

*\*\* Docker engine generates an access token and a join command to be used to add a worker to the swarm*. 

* 1. **Access the Worker-1 VM and use the join command to add it as a worker. Your swarm access token should be different from the token shown in the figure below.** 
  2. Repeat step 3 in Worker-2 VM and use the same join command.
  3. Switch back to the **manager** VM and verify you can see three nods joined the swarm.

docker node ls

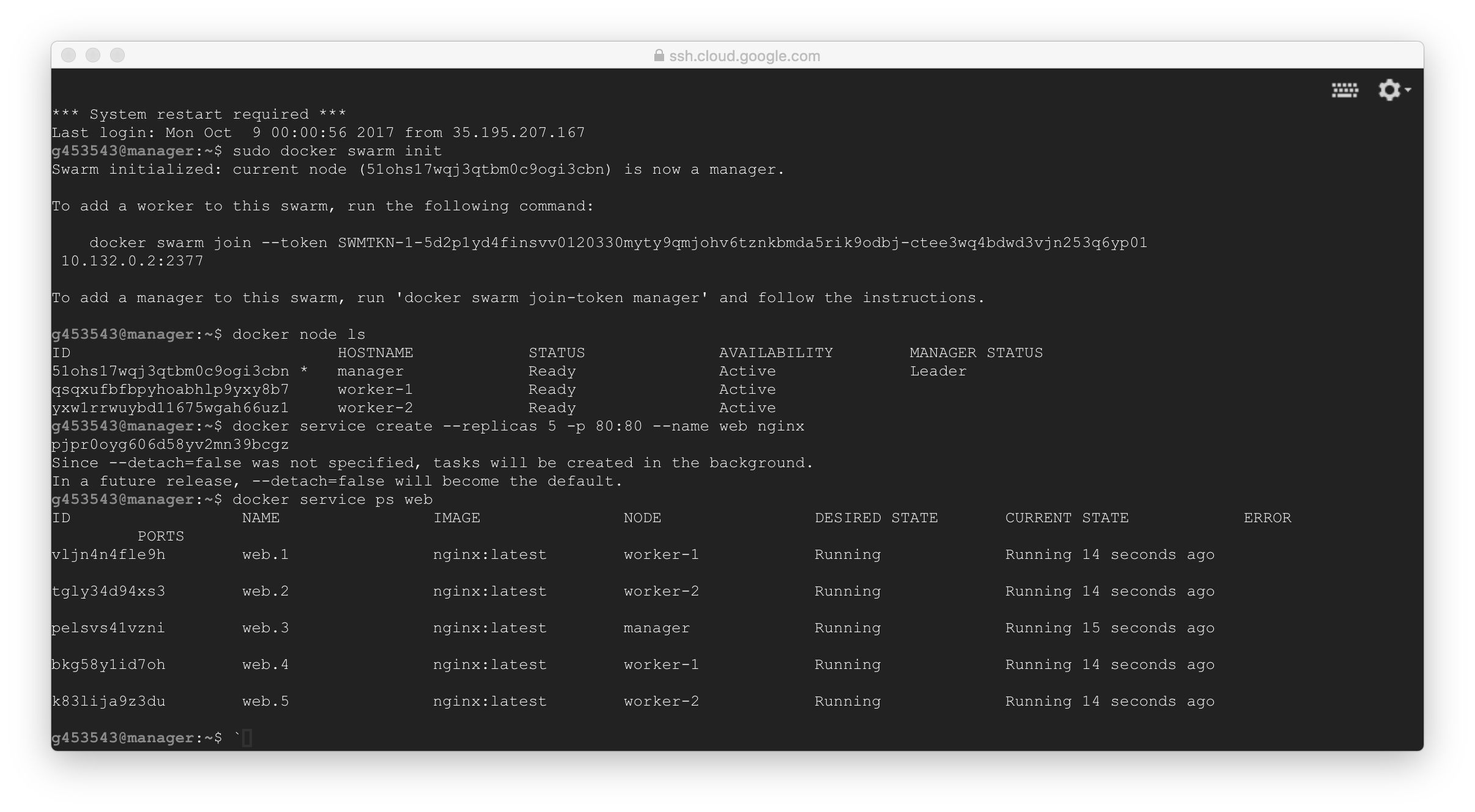


* 1. Now that we have our swarm up and running, it is time to run a service on it. We are going to focus on the application and not to worry about where the application is going to run. All we need to do is tell the manager to run the containers and it will take care of scheduling out the containers, sending the commands to the workers and distributing it cross the swarm workers.
  2. To test this, create a demo service based on nginx web server image. using the following command.

docker service create --replicas 5 -p 80:80 --name web nginx

* 1. Check that the service **web** is running using, observe to which node it was assigned (see the figure below as an example):

docker service ps web



* 1. Visit the external IP/public-dns of any node (manager, worker1, or worker2) and you should be able to see the NGINX welcome page. (make sure you allowed http port 80 in the security group)
  2. Finally, remove the service **web** and verify it not running using:

docker service rm web

docker service ps web

# PART 3: build a docker container

* 1. Create new GITHUB Repo lab5 (with a Readme File in it) and clone it to the manager VM.
  2. Clone a copy of my starter repo: <https://github.com/baselm/lab5-init-repo.git>
  3. Copy the my\_application/ and Dockerfile from my repo into your repo
  4. Build the new docker image and call it lab5

docker build -t lab5-image .

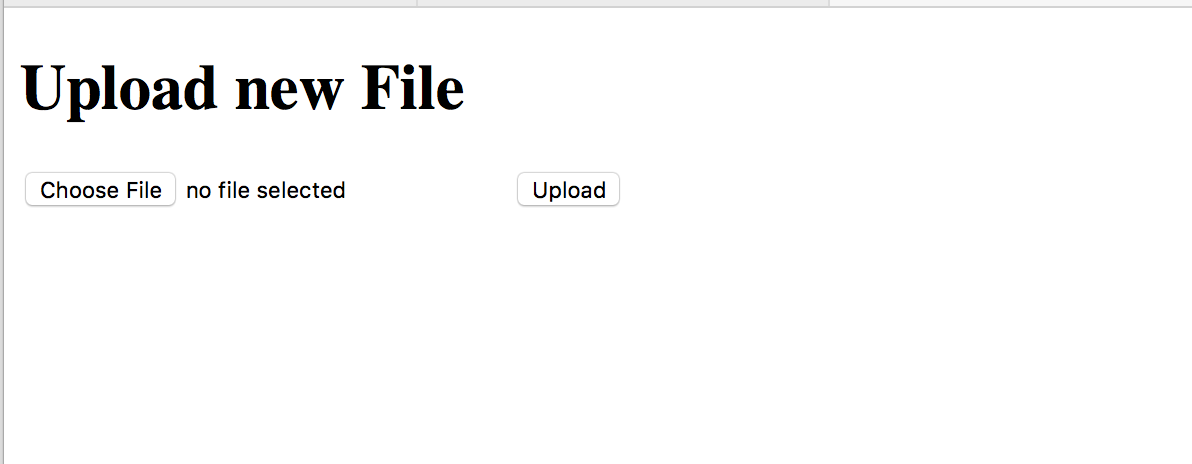
* 1. Check there is a new image called lab5-image using

docker image ls

* 1. Create a container from lab5 image as below and call the container webserver1. The –d runs the container in the background (as a daemon), and –p maps port 80 to the container.

docker run –d –p 80:5000 –name webserver1 lab5-image

* 1. Make sure it works from a browser by accessing the manager’s External IP address
  2. Upload a file to the container which should look similar to the following



* 1. Get the container id using

docker ps

* 1. Verify that the file is now in the container using the docker exec –it command using the following methods (one breaks into the VM the other looks inside it). Data is stored in the /data directory.

docker exec –it <containerid> /bin/bash

docker exec –it <containerid> ls /data

* 1. Try to access the webserver1 using the external IP of worker 1. Explain why you failed to gain access to the webserver1?
  2. Stop webserver1 container and remove it

docker stop webserver1

docker rm webserver1

# Part 4: Run docker image as a service

In this part, we will run the lab5-image as a service in the swarm to get benefit from the swarm load balancing, orchestration features. In order to make the lab5-image available in all swarm nodes, we will be using an image uploaded into docker hub called baselm/lab5-image

1. Create a service based on lab5-image with 10 replicas

docker service create --replicas 3 -p 80:5000 --name webservice baselm/lab5-image

1. Check the service state using the following command. You must see a replica of webservice is assigned to nodes in the swarm.

docker service ps webservice

1. Now let us add some visualisation feature to the swarm. Run the following command in manager 1:

docker service create \

--name=viz \

--publish=8080:8080/tcp \

--constraint=node.role==manager \

--mount=type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \

dockersamples/visualizer

1. Visit the manager’s external IP:8080 to see a visualisation of the swarm you just created in this Lab.

# Part 5: VERTICAL AND HORIZONTAL SCALING

In the following steps we will use docker swarm to scale the services vertically and the AWS autoscaling to do horizontal scaling. In this context, vertical scaling will add more containers to the services, horizontal scaling will add more VM to the swarm.

**Task 1: Vertical Scaling using Docker Swarm**

1. Scale the webservice application to 10 replicas using

docker service scale webservice=10

1. observe the impact of scaling the service using the command

docker service ps

1. Visit the manager public-dns to see the visualisation (viz) service running at port 8080.

**Task 2: Horizontal Scaling**

1. Back to AWS Console
2. In the AWS Management Console, on the Services menu, click EC2
3. In the navigation pane, click Instances.
4. Right-click on manager, and then click Image > Create Image.
5. Configure the following settings (and ignore any settings that are not listed):
   1. Image name: Docker Swarm AMI
   2. Image description: AMI for Docker
6. Click Create Image.
7. Lunch three instances based on your Docker Swarm AMI you created in previous step.
8. From the launch configuration, select My AIMIs, then select Docker Swarm AMI. As you can see in the following image.

A screenshot of a social media post 
 
Description automatically generated

1. Make sure to assign the new created VMs to the security group you created at the beginning of this lab: ***Docker-SG***.
2. Once the initiation of the VM completed access the VMs and make sure Docker is installed
3. Add two of the new created VMs to the Swarm as workers.(To get the join command for worker do the following:
   1. Access the manager VM using SSH
   2. Run the following command:

docker swarm join-token worker

1. Access each one of the new created VMs and use the command to join the swarm as worker.
2. Go back to the manager and access it using SSH. Use the following command to get the manager access token

docker swarm join-token manager

1. Copy the join command and access the third VM using SSH, and execute the join command so it will join the swarm as manager.
2. From the manager SSH session, verify that you can see six nodes joined to the swarm using the following command:

docker node ls

1. Visit the visualisation service at port 8080 to see all nodes and containers in the swarm.

# Part 6: Docker Swarm monitoring

* 1. From the manager node SSH access. Pull the following git repo:

git clone https://github.com/baselm/lab-extra.git

* 1. This Docker Swarm monitoring with [Prometheus](https://prometheus.io/), [Grafana](http://grafana.org/), [cAdvisor](https://github.com/google/cadvisor), [Node Exporter](https://github.com/prometheus/node_exporter), [Alert Manager](https://github.com/prometheus/alertmanager) and [Unsee](https://github.com/cloudflare/unsee).
  2. Go back to AWS console security groups and edit the inbound rules for Docker-SG to open the following TCP ports:

TCP:9323, source:0.0.0.0/0

TCP:9090, source:0.0.0.0/0

TCP:3000, source:0.0.0.0/0

TCP:9093, source:0.0.0.0/0

TCP:9094, source:0.0.0.0/0

* 1. Access the root directory of the repo using: cd lab-extra
  2. Run the following commands:

ADMIN\_USER=admin

ADMIN\_PASSWORD=admin

SLACK\_URL=https://hooks.slack.com/services/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

SLACK\_CHANNEL=devops-alerts

SLACK\_USER=alertmanager

docker stack deploy -c docker-compose.yml mon

* 1. Use the following link to join the DT228 cloud computing channel in Slack.

<https://join.slack.com/t/dt228cloudcomputing/shared_invite/enQtNDg0Mzk3NzczMzAwLWY1MDEwNmNmZDdlYWQ2ZWY0NWY5NzdjNjhmZDIxNmU1YTNiZGM5NTQ3M2IyOWQ3OGYzODA1ZDZmNjdhZTFlMDE>

* 1. Access the dt228cloudcomputing channel in slack
  2. From a new tab, access the manager public-dns at port 3000 to access Grafana Dashboard. Remember the username and password are **admin/admin**
  3. You can use the following command to do a stress test of the docker swarm:

docker run --rm -it progrium/stress --cpu 2 --io 1 --vm 2 --vm-bytes 128M --timeout 200s

* 1. Check if you have received any alerts in the slack channel. The name of the VM you created will be displayed in the Alert’s message.