LOG8415E - TP1

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

LOG8415E - Advanced Concepts of Cloud Computing

Département de génie logiciel
École Polytechnique de Montréal

18 octobre 2022

Clara Serruau	2164678
Lydia Haile	2230615
Cesar Miguel Valdez Cordova	2229636
Svante Sundberg	2229625

1 GitHub repo

The github repo can be found at : `https://github.com/LydiaHaile/CloudComputing_TP1`

2 Flask Application Deployment Procedure

AWS CLI Connection

Each time a new session is started on the Learner Lab platform a set of aws credentials is produced. The credentials are presented in the following format on Learner Lab and should be exactly pasted in the *creds.txt* file, in the provided working directory.

```
[default]
aws_access_key_id=XXX...XXX
aws_secret_access_key=xxx...XXX
aws_session_token=XXX......XXX
```

The aforementioned file manipulation and allocation has to be done each time a new session is started. Moreover, the file *labsuser.pem*, as provided in *Learner Lab*, has to be downloaded and moved to the working directory. Once this is done, a connection can be set up from the terminal or command prompt to the started aws session.

Launching Instances and Deploying Flask Application

The Flask deployment procedure will be further described under the assumption that the connection to an aws instance is already secured. Before the Flask application can be deployed, instances need to be launched and a security group needs to be created.

A script (*script.sh*) was created to automate :
— the creation of a custom security group,
— the launching of nine instances (five t2.large and four m4.large)
— the deployment of flask on each instance

The script *script.sh*, does that by :
— running commands to install aws and the modules Boto3 and Paramiko,
— moving the credentials added to *cred.txt* to the correct location in the system (*/.aws/credentials*),
— changing the ownership of the *labsuser.pem* file, and
— running a python script (*launching.py*)

The python script, *launching.py*, starts off by running the functions *create_security_group* and *create_instance* which produces our security groups and instances. Next it fetches the DNS and IP-addresses for every created instance. The script then configures an ssh connection with help of Paramiko. Finally a list of commands and a python script are created by the function *create_commands* for each instance and then run. Both the commands and the script are necessary for the deployment of the flask application.

The function *create_commands* takes the instance's number as an argument so that each Flask application knows the id of the instance on which it is being run. The list of commands and the python script produced are described next. First, commands to update the package information are run and a python3-pip module is installed. Then the folder containing the flask module is set in the PATH environment variable, and Flask is installed using *pip*. An file, *app.py*, is created to contain the application itself. This is where the instance id is injected. The *SERVER_NAME* is also updated with the first argument given to the script, and run on host='0.0.0.0' and port=80. To run this python code and make the application run on the background, nohup is used. PATH is used for environment. Next, $(ec2metadata –public-ipv4) is given as an argument to the script, to enable a connection to the deployed Flask application using the Public IPV4 address of the created instances.

Important details : Between each command, stdout.read() is printed. This makes the execution wait for the shell to return a value before executing the next command. For the last command, *'sudo nohup env "PATH=$PATH" python3 app.py $(ec2metadata –public-ipv4) '*, stdout is not printed since this command does not return a stdout. Thus, trying to print the stdout for the last command would result in an infinite waiting time.

3 Cluster setup using Application Load Balancer

The creation of application load balancer and clusters is also managed by the *launching.py* python script. The following functions are involved :

— create_target_groups()
— create_load_balancer(security_group_id, subnets)
— setup_listeners(elb_arn, cluster_1_arn, cluster_2_arn)
— create_elb_target_groups_listeners_rules(security_group_id, vpc_id, target_cluster_1, target_cluster_2)

First the two target groups, cluster 1 and cluster 2, are created and the nine instances are divided between them. Four m4.large instances running in availability zone us-east-1b are registered as targets in cluster 1, while five t2.large instances running in availability zone us-east-1a are registered in cluster 2. Next, the application load balancer is instantiated. It is made to be internet facing and to work with IPV4-addresses. A listener is then set up on the load balancer with two customized rules on top of the default rule. The default rule routes traffic to the first cluster and the two customised rules direct traffic to either of the clusters, depending on the specified path.

4 Results of your benchmark

In this section the results of the benchmark tests are presented with Figures.

5 Instructions to run the code

Follow the steps listed below in order to run the code :

1. Copy the AWS credentials and paste them in the *creds.txt* file.
2. Download the *labsuser.pem* and place it in the working directory.
3. Navigate to the working directory in the terminal/command prompt.
   *Run `./script.sh`.

To terminate the instances run : `python3 terminate.py`.

*It might be necessary to run `chmod +x script.sh` and before running the *script.sh*. The `chmod +x script.sh` command makes the *script.sh* executable.