Computer Lab 2: Distributed Infrastructure, Model Serving and CI/CD.
Course: Data Engineering-II

Teacher: Salman Toor, salman.toor@it.uu.se
Teacher assistance: Ben Blamey ben.blamey@it.uu.se
Addi Ait-Mlouk addi.ait-mlouk@it.uu.se

Please carefully read the instructions before you begin this assignment

Summary
The lab assignment covers the practical part of the discussed concepts of dynamic contextualization and model serving in a scalable production environment and reliable continuous integration and development process. The lab consists of ***four*** tasks and one optional task. First three tasks are compulsory, forth task is a bonus task and optional task does not have any points.

Grade for the assignment
Total grade for this assignment ranges from 1 to a maximum of 3 points. First three tasks have a total of 1 point, whereas, forth task has a maximum of 2 points.

Submission Policy
The solutions should be submitted no later than 27-04-2022 **23.59** (Central European Time).

Discussions
Problems can be discussed on discussion forums for the lab on Studium. The forum encourages students to discuss issues related to the assignments / lab work.

Note: Assignments can be discussed with your fellow students but copying solutions of your fellow student is NOT allowed which can result in disciplinary action.

Important:

1- Please terminate VMs once you finish the task.
2- We recommend you to create a VM in SSC and execute all the tasks on that virtual machine. You can run the tasks on your laptops but it may break your local working environment.
3- For all the tasks, clone the repository: https://github.com/sztoor/model_serving.git
4- The code for tasks is available in the "model_serving" directory.

```
- model_serving
  -- Single_server_without_docker
  -- Single_server_with_docker
  -- CI_CD
  -- OpenStack-Client
```

Dynamic contextualization

Dynamic contextualization is a process of preparing a customized computing environment at runtime. The process ranging from creating/defining user roles and permissions to updating/installing different packages and initiating services.

Task-1: Single server deployment

In this task, we will learn how the dynamic contextualization works using Cloudinit package. For this task, we need to use OpenStack APIs to start a new VM and contextualize it at run time. The contextualization process will setup the following working environment:

1- Flask based web application as a frontend server
2- Celery and RabbitMQ server for backend server
3- Model execution environment based on Keras and TensorFlow

In case you are not familiar with the above-mentioned packages please read the following links:

1- Flask Application -> https://flask.palletsprojects.com/en/1.1.x/
2- Celery and RabbitMQ -> https://docs.celeryproject.org/en/stable/getting-started/
3- Keras and TensorFlow -> https://www.tensorflow.org/guide/keras

The process will start when a client sends a new prediction request from the front-end web server. The server will pass the request to the backend Celery environment where running workers in the setup will pick up the task, run the predictions by loading the available model, send back the results and finally frontend server will display the results.

Steps for VM contextualization

1- Start a VM from the SSC dashboard and login.
2- Clone the git repository.

    git clone https://github.com/sztoor/model_serving.git

3- Goto the “model_serving/single_server_without_docker/production_server/” directory. The directory contains the code that will run on the production server VM.

Following is the structure of the code:

```
- Flask Application based frontend
   -- app.py
   -- static
   -- templates
 - Celery and RabbitMQ setup
   -- run_task.py
   -- workerA.py
 - Machine learning Model and Data
   -- model.h5
   -- model.json
   -- pima-indians-diabetes.csv
```

Open files and understand the application's structure.

4- Goto the "model_serving/openstack-client/single_node_without_docker_client/" directory. This is the code that we will use to contextualize our production server. The code is based on the following two files:

```
- CloudInit configuration file
   -- cloud-cfg.txt
- OpenStack python code
   -- start_instance.py
```

Open the files and understand the steps. NOTE: You need to setup variable values in the start_instance.py script.

Important: In order to run this code, you need to have OpenStack API environment running.

NOTE: Openstack APIs are only need to be installed on the client VM.

5- Follow the instructions available on the following links:

https://docs.openstack.org/install-guide/environment-packages-ubuntu.html , http://docs.openstack.org/cli-reference/common/cli_install_openstack_command_line_clients.html
Download the client tools and API for OpenStack.

Download the Runtime Configuration (RC) file (version 3) from the SSC site (Top left frame, Project->API Access->Download OpenStack RC File).

Set API access password. Goto https://cloud.snic.se/, Left frame, under Services "Set your API password".

Confirm that your RC file have following environment variables:

```
export OS_USER_DOMAIN_NAME="snic"
export OS_IDENTITY_API_VERSION="3"
export OS_PROJECT_DOMAIN_NAME="snic"
export OS_PROJECT_NAME="UPPMAX 2022/1-1"
```

6- Set the environment variables by sourcing the RC-file in the client VM.

```
# source <project_name>_openrc.sh
```

NOTE: You need to enter the API access password.

The successful execution of the following commands will confirm that you have the correct packages available on your client VM:

```
# openstack server list
# openstack image list
```

For the API communication, we need following extra packages:

```
# apt install python3-openstackclient
# apt install python3-novaclient
# apt install python3-keystoneclient
```

NOTE: You need to setup variable values in the start_instance.py script.

7- Once you setup the environment, run the following command.

```
# python3 start_instance.py
```

The command will start a new server and initiate the contextualization process. It will take approximately 10 to 15 minutes. The progress can be seen on the cloud dashboard. Once the process finish, attach a floating IP to your production server and access the webpage from your client machine.

Welcome page http://<PRODUCTION-SERVER-IP>:5100
Predictions page http://<PRODUCTION-SERVER-IP>:5100/predictions

Questions

1- Explain how the application works? Write a short paragraph about the framework.

2- What are the drawbacks of the contextualization strategy adopted in the task-1? Write at least four drawbacks.

3- Currently, the contextualization process takes 10 to 15 minutes. How can we reduce the deployment time?

4- Write half a page summary about the task and add screenshots if needed.

TERMINATE THE PRODUCTION SERVER VM STARTED FOR TASK-1!

Task-2: Single server deployment with Docker Containers

In this task, we will repeat the same deployment process but with Docker containers. This time we will create a flexible containerized deployment environment where each container has a defined role.

1- Goto “model_serving/single_server_with_docker/production_server/” directory on the client VM. The directory contains the code that will run on your production server. Following is the structure of the code:

```
- Flask Application based frontend
   -- app.py
   -- static
   -- templates
 - Celery and RabbitMQ setup
   -- run_task.py
   -- workerA.py
 - Machine learning Model and Data
   -- model.h5
   -- model.json
   -- pima-indians-diabetes.csv
 - Docker files
   -- Dockerfile
   -- docker-compose.yml
```

Open files and understand the application's structure.

2- Goto the “model_serving/openstack-client/single_node_with_docker_client/” directory. The directory contains the code that we will use to contextualize the production server. The code is based on the following two files:

```
- CloudInit configuration file
  -- cloud-cfg.txt
- OpenStack python code
  -- start_instance.py
```

Open the files and understand the steps. You will need to update the *key_name* value in *start_instance.py* file with a previously created keypair.

NOTE: You need to setup variable values in the start_instance.py script.

3- Run the following command.

```
# python3 start_instance.py
```

The command will start a new server and initiate the contextualization process. It will take approximately 10 to 15 minutes. The progress can be seen on the cloud dashboard. Once the process finish, attach a floating IP to your production server and access the webpage from your client machine.

Welcome page http://<PRODUCTION-SERVER-IP>:5100
Predictions page http://<PRODUCTION-SERVER-IP>:5100/predictions

4- The next step is to test the horizontal scalability of the setup.

- Login to the production server.

```
# ssh -i <PRIVATE KEY> ubuntu@<PRODUCTION-SERVER-IP>
```

- Switch to the superuser mode.

```
# sudo bash
```

- Check the cluster status:

```
# cd /model_serving/single_server_with_docker/production_server
# docker-compose ps

The output will be as following:

   Name                  Command             State                                  Ports
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
production_server_rabbit_1      docker-entrypoint.sh rabbi ...   Up      15671/tcp, 0.0.0.0:15672->15672/tcp, 25672/tcp, 4369/tcp,
5671/tcp, 0.0.0.0:5672->5672/tcp
production_server_web_1         python ./app.py --host=0.0.0.0   Up      0.0.0.0:5100->5100/tcp
production_server_worker_1_1    celery -A workerA worker - ...   Up

Following three containers are running on the production server:

production_server_rabbit_1  -> RabbitMQ server
production_server_web_1 -> Flask based web application
production_server_worker_1_1 -> Celery worker
```

- Now we will add multiple workers in the cluster using docker commands. Currently, there is one worker available. We will add two more workers.

```
# docker-compose up --scale worker_1=3 -d

Check the cluster status.

# docker-compose ps

           Name                  Command             State                                  Ports
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
production_server_rabbit_1      docker-entrypoint.sh rabbi ...   Up      15671/tcp, 0.0.0.0:15672->15672/tcp, 25672/tcp, 4369/tcp,
5671/tcp, 0.0.0.0:5672->5672/tcp
production_server_web_1         python ./app.py --host=0.0.0.0   Up      0.0.0.0:5100->5100/tcp
production_server_worker_1_1    celery -A workerA worker - ...   Up
production_server_worker_1_2    celery -A workerA worker - ...   Up
production_server_worker_1_3    celery -A workerA worker - ...   Up
Now we have 3 workers running in the system.

Scale down the cluster.

# docker-compose up --scale worker_1=1 -d
```

Questions

1- What problems Task-2 deployment strategy solves compared to the strategy adopted in Task-1?

2- What are the outstanding issues that the deployment strategy of Task-2 cannot not address?

3- What are the possible solutions to address those outstanding issues?

4- What is the difference between horizontal and vertical scalability? Is the strategy adopted in Task-2 follow horizontal or vertical scalability?

5- Write half a page summary about the task and add screenshots if needed.

TERMINATE THE SERVER VM STARTED FOR TASK-2!

Continuous Integration and Continuous Delivery (CI/CD)

The next two tasks will cover scalable cluster deployments using Ansible playbooks and CI/CD environment using versioning system Git and an extension called Git Hooks.

Task-3: Deployment of multiple servers using Ansible

For this task we need a setup based on following three VMs:

- Client VM: This machine will serve as an Ansible host name and initiate the configuration process.
- Production Server: The machine will host the dockerised version of the application discussed in task-1.
- Development Server: The machine will host the development environment and push the new changes to the production server.

1. Steps to setup Ansible host and orchestration environment

If you are not familiar with the Ansible, visit following URLs:

https://www.ansible.com/

https://www.ansible.com/blog/it-automation
https://www.ansible.com/overview/how-ansible-works

2. Login to the Client machine

```
# ssh -i <PRIVATE KEY> ubuntu@<PRODUCTION-SERVER-IP>
```

3. Goto “model_serving/ci_cd” directory. This directory contains the following two sub-directories

```
/production_server
 - Flask Application based frontend
    -- app.py
    -- static
    -- templates
 - Celery and RabbitMQ setup
    -- run_task.py
    -- workerA.py
 - Machine learning Model and Data
    -- model.h5
    -- model.json
    -- pima-indians-diabetes.csv
/development_server
 - Machine learning Model and Data
    -- model.h5
    -- model.json
    -- pima-indians-diabetes.csv
    -- neural_net.py
```

Open files and understand the application's structure.

4. Goto “model_serving/openstack-client/single_node_with_docker_ansible_client” directory. The files in the directory will be used to contextualize the production and deployment servers.

```
- CloudInit files
  -- prod-cloud-cfg.txt
  -- dev-cloud-cfg.txt
- OpenStack code
  -- start_instance.py
- Ansible files
  -- setup_var.yml
  -- configuration.yml
```

The client code will start two VMs and by using Ansible orchestration environment. It will contextualize both of the VMs simultaneously.

5. Install and configure Ansible on the client machine.

- First generate a cluster SSH key.
- Check the username you are login as

```
# whoami
ubuntu
```

Important: You need to be as ubuntu user. If you are root user switch back to ubuntu user.

Create a directory

```
# mkdir -p /home/ubuntu/cluster-keys
# ssh-keygen -t rsa
```

Set the file path /home/ubuntu/cluster-keys/cluster-key Do not set the password, simply press Enter twice.

Output

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
/home/ubuntu/cluster-keys/cluster-key
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/cluster-keys/cluster-
key.
Your public key has been saved in /home/ubuntu/cluster-keys/cluster-
key.pub.
The key fingerprint is:
4a:dd:0a:c6:35:4e:3f:ed:27:38:8c:74:44:4d:93:67 demo@a
The key's randomart image is:
+--[ RSA 2048]----+
|        .oo.   |
|       . o.E  |
|      + . o   |
|    . = = .    |
|     = S = .   |
|    o + = +    |
|     . o + o .  |
|        . o   |
|              |
+-----------------+
```

- The step will generate cluster ssh keys at the following location:
    - Private key: /home/ubuntu/cluster-keys/cluster-key
    - Public key: /home/ubuntu/cluster-keys/cluster-key.pub

6. Next, we will start the **Production** and **Development** servers.

- Goto the "model_serving/openstack-client/single_node_with_docker_ansible_client", open prod-cloud-cfg.txt delete the old key from the section **ssh_authorized_keys:** and copy the complete contents of "/home/ubuntu/cluster-keys/cluster-key.pub" in the "**prod-cloud-cfg.txt**" file.

- Repeat same step 3 with the **dev-cloud-cfg.txt**. Delete the old key from the section **ssh_authorized_keys:** and copy the complete contents of **/home/ubuntu/cluster-keys/cluster-key.pub** in the **dev-cloud-cfg.txt** file.

7. Run the start_instance.py code.

```
# python3 start_instance.py

The output will give you the internal IPs of the VMs.

user authorization completed.
Creating instances ...
waiting for 10 seconds..
Instance: prod_server_with_docker_6225 is in BUILD state, sleeping for 5 seconds more...
Instance: dev_server_6225 is in BUILD state, sleeping for 5 seconds more...
Instance: prod_server_with_docker_6225 is in ACTIVE state, sleeping for 5 seconds more...
Instance: dev_server_6225 is in BUILD state, sleeping for 5 seconds more...
Instance: __prod_server_with_docker_6225__ is in ACTIVE state ip address: __192.168.1.19__
Instance: __dev_server_6225__ is in ACTIVE state ip address: __192.168.1.17__
```

Now we have two VM running with the internal IP addresses.

8. Install Ansible packages on the client machine.

```
# sudo bash
# apt update; apt upgrade
# apt-add-repository ppa:ansible/ansible
# apt update
# apt install ansible
```

9. Next step is to enter these IP addresses in the Ansible hosts file.

- prod_server_with_docker_6225 -> 192.168.1.xx
- dev_server_6225 -> 192.168.1.xx

Open the Ansible inventory file and add the IP addresses in that file. For this step you need to

```
swtich to root user.
sudo bash
nano  /etc/ansible/hosts
file contents

[servers]
prodserver ansible_host=<production server IP address>
devserver ansible_host=<development server IP address>

[all:vars]
ansible_python_interpreter=/usr/bin/python3

[prodserver]
prodserver ansible_connection=ssh ansible_user=appuser

[devserver]
devserver ansible_connection=ssh ansible_user=appuser
```

If you need to learn more about Ansible, here are some useful links:

Easy installation instructions https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-ubuntu-18-04

- Just to confirm the access permission are correctly set, access production and development server from the client VM.

First switch back to user ubuntu

```
ssh -i /home/ubuntu/cluster-keys/cluster-key appuser@<PRODUCTION-SERVER-IP>
```

If the login is successful, exit from the production server. Repeat the step with development server.

```
# ssh -i /home/ubuntu/cluster-keys/cluster-key appuser@<DEVELOPMENT-SERVER-IP>
```

If the login is successful, exit from the development server.

For this step, switch to ubuntu user on the client VM.

Now we will run the Ansible script available in the “model_serving/openstack-client/single_node_with_docker_ansible_client” directory.

```
# export ANSIBLE_HOST_KEY_CHECKING=False
# ansible-playbook configuration.yml --private-key=/home/ubuntu/cluster-keys/cluster-key
```

The process will take 10 to 15 minutes to complete.

Attach floating IP address to the production server and access the same predictions webpage as we did in previous tasks.

Questions

1 - What is the difference between CloudInit and Ansible?

2 - Explain the configurations available in **dev-cloud-cfg.txt** and **prod-cloud-cfg.txt** files.

3 - What problem we have solved by using Ansible?

Important: Task-4 is the continuation of Task-3. Do NOT terminate your instances setup in Task-3.

Task-4: CI/CD using Git HOOKS

In this task, we will build a reliable execution pipeline using Git Hooks. The pipeline will allow continuous integration and delivery of new machine learning models in a production environment.

1- Enable SSH key based communication between production and development servers

- Login to the development server

```
# ssh -i cluster-key appuser@<DEVELOPMENT-SERVER-IP>
```

- Generate SSH key

```
# ssh-keygen
```

Important: Do not change the default /home/appuser/.ssh/id_rsa path of the file.
Do not set the password, simply press Enter twice.

NOTE: This step will create two files, private key /home/appuser/.ssh/id_rsa and public key /home/appuser/.ssh/id_rsa.pub.

- Copy the contents of public key file /home/appuser/.ssh/id_rsa.pub from the development server.

2- Login to production server

```
# ssh -i cluster-key appuser@<PRODUCTION-SERVER-IP>
```

- Open file "/home/appuser/.ssh/authorized_keys" and past the contents of the public key file in the authorized_keys file.

```
# nano /home/appuser/.ssh/authorized_keys
```

- Create a directory (it will be jump directory)

```
# nano /home/appuser/.ssh/authorized_keys
# pwd
Output
/home/appuser/
# mkdir my_project
# cd my_project
```

- Here is the path to your jump directory. Double check that user appuser is the owner of my_project directory.

```
# pwd
/home/appuser/my_project
Create git empty directory
git init --bare
```

The command will initialize empty Git repository in /home/appuser/my_project/

- Create a git hook post-receive

```
# nano hooks/post-receive
File contents
------------
#!/bin/bash
while read oldrev newrev ref
do
    if [[ $ref =~ .*/master$ ]];
    then
        echo "Master ref received.  Deploying master branch to production..."
        sudo git --work-tree=/model_serving/ci_cd/production_server --git-dir=/home/appuser/my_project
checkout -f
    else
        echo "Ref $ref successfully received.  Doing nothing: only the master branch may be deployed on this
```

- Change permissions

```
# nano chmod +x hooks/post-receive
```

- Exit Production Server

3- Login to the Development Server

```
# ssh -i cluster-key appuser@<DEVELOPMENT-SERVER-IP>
```

- Create a directory

```
# pwd
/home/appuser/
# mkdir my_project
# cd my_project
```

- This is your development directory. Double check that user appuser is the owner of my_project directory.

```
# pwd
/home/appuser/my_project
```

- Create empty git repository

```
# git init
Output

Initialized empty Git repository in `/home/appuser/my_project/.git/`
```

- Add files **model.h5** and **model.json** in **/home/appuser/my_project/** directory. The files are available in **"/model_serving/ci_cd/development_server/"** directory.

- Goto the **/home/appuser/my_project** directory

- Add files for the commit

```
# git add .
```

- Commit files

```
# git commit -m "new model"
```

- Connect development server's git to production server's git.

```
# git remote add production appuser@<PRODUCTIONS-SERVER-IP>:/home/appuser/my_project
```

- Push your commits to the production server

```
# git push production master
Output

Delta compression using up to 2 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 2.36 KiB | 2.36 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
remote: Master ref received.  Deploying master branch to production...
To 192.168.1.21:/home/appuser/my_project
* [new branch]      master -> master
```

In case you want to learn more about Git Hooks, visit the following link: https://www.digitalocean.com/community/tutorials/how-to-use-git-hooks-to-automate-development-and-deployment-tasks

4- Goto "/model_serving/ci_cd/development_server/" directory. The directory contains the **neural_net.py** python script. The script will train a model and generate new model files **model.h5** and **model.json**. Open the training script **neural_net.py**, make changes in the model, install dependencies (numpy will be required) and run the script, move them in the git repository, commit changes and then push new model files in the running production pipeline.

- Run the script

```
# python3 neural_net.py
```

- Copy model files (**model.h5** and **model.json**) from the development directory model_serving/ci_cd/development_server/ to the git repository /home/appuser/project on the development server.

```
# cp /model_serving/ci_cd/development_server/model*  /home/appuser/project/.
```

- Add to the git repository.

```
# git add .
```

- Finally, commit and push the new model files.

```
# git commit -m "new model"
# git push production master
Output

Delta compression using up to 2 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 2.36 KiB | 2.36 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
remote: Master ref received.  Deploying master branch to production...
To 192.168.1.21:/home/appuser/my_project
* [new branch]      master -> master
```

- Access the URL http://<PRODUCTION-SERVER-IP>:5100/predictions and you will see the changes in predictions.

- Improve the model by changing parameters or network architecture in the neural_net.py file and repeat the step-3 to push the new model to the production pipeline.

Questions

1- What are git hooks? Explain the post-receive script available in this task.

2- We have created an empty git repository on the production server. Why we need that directory?

3- Write the names of four different git hooks and explain their functionalities.

Hints - Read the link available in the task or access sample scripts available in the hooks directory.

4- How deployment of multiple servers (in task 3) can be achieved with Kubernetes? Draw a diagram of the deployment highlighting the nodes, services, configuration map and secret.

5- Write half a page summary about the task and add screenshots if needed.

Optional Task

Write Kubernetes/vault compatible configurations (yaml files) to deploy multiple servers discussed in task 3 (practical demonstration).