Report of Computer Lab 2

Course: Data Engineering II
Uppsala University
Junjie Chu

Task 1

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

The application bases on 2 VMs. The first VM is the client VM. We need to install Openstack API on the client VM. After that, we set the cloud-cfg.txt and run start-instance.py on that. 'start-instance.py' is used to create an instance on Openstack and 'cloud-cfg.txt' is used to configure the environment of the server VM.

Both front-end server and backend server run in the server VM. The 'Celery' and 'RabbitMQ' load machine learning models and data from 'Machine learning Model and Data'. They are used as backend server. The backend server distributes the task to the worker A. And the Flask based web application runs as the frontend server. The frontend server manages incoming requests, reads the result from the backend server and show them in the website.

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

1. Poor reliability. All programs run in a VM and are not isolated from each other. When we update some package dependencies or perform other operations, they may affect each other.
2. Poor scalability. If you want to add a new worker, you need to add a new script, or even modify all other scripts.
3. There is a single point of failure. Once a front-end service or back-end service has a failure, the entire program fails.
4. Low resource utilization efficiency. Compared with containers, a lot of resources are used in virtual systems and hardware.
5. Difficult to maintain. When a problem occurs, you need to log in the second VM to manually check each module.
6. The developed application is difficult to port to other VMs.
7. Let users log in production servers directly is not very safe.

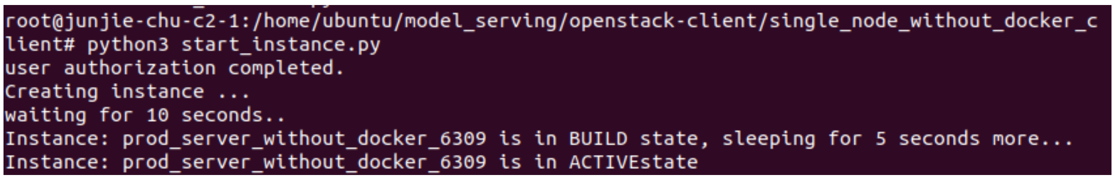
3 - Currently, the contextualization process takes 10 to 15 minutes. How can we reduce the deployment time?

In the process, the image we used to create the instance is specified in the 'start-instance.py'. I use a native image of Ubuntu 20.04 and many packages are not installed in the image. So, after we create the instance, we need to use a lot of time to download and install them. But we could install all the packages we need in an instance in advance, then upload that image. If we use the new image to create the instance, the process will be much faster. There is no need to install and only some commands need to be run.

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

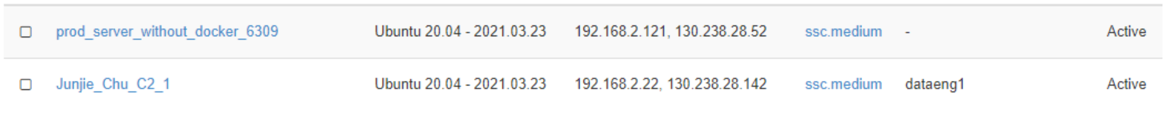
In the task, we use a client VM and 'cloud-init' to create an instance. There are 2 VMs in total. This instance is used as both frontend server and backend server. And we could access the result via a web page.

The process bases on 2 VMs. The first VM is the client VM. We need to install Openstack API on the client VM. After that, we set the cloud-cfg.txt and run start-instance.py on that. ‘start-instance.py’ is used to create an instance on Openstack and ‘cloud-cfg.txt’ is used to configure the environment of the instance. After we successfully run the start-instance.py, we could see:



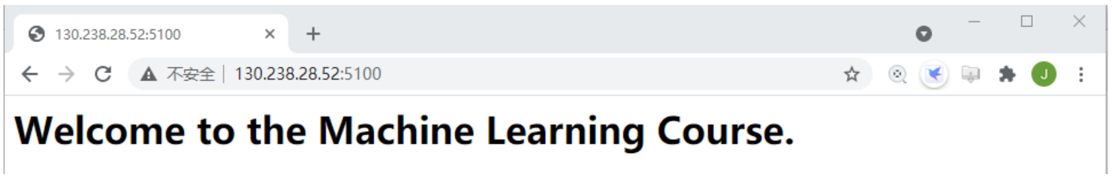
```
root@junjie-chu-c2-1:/home/ubuntu/model_serving/openstack-client/single_node_without_docker_c
lient# python3 start_instance.py
user authorization completed.
Creating instance ...
waiting for 10 seconds..
Instance: prod_server_without_docker_6309 is in BUILD state, sleeping for 5 seconds more...
Instance: prod_server_without_docker_6309 is in ACTIVEstate
```

And on Openstack dashboard, we could see:



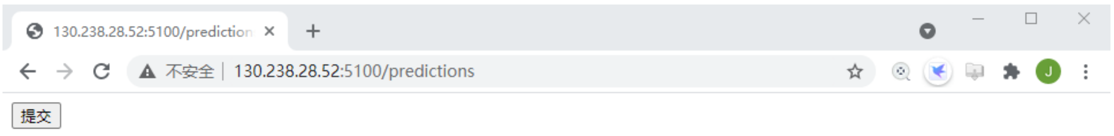
▢	prod_server_without_docker_6309	Ubuntu 20.04 - 2021.03.23	192.168.2.121, 130.238.28.52	ssc.medium	-	Active
▢	Junjie_Chu_C2_1	Ubuntu 20.04 - 2021.03.23	192.168.2.22, 130.238.28.142	ssc.medium	dataeng1	Active

Via ‘cloud-cfg.txt’, we made the instance download and install some packages automatically. And let the instance run the application of machine learning. We could access the application via a webpage using the floating ip to check if the application runs properly. If everything is right, the result is as follows.

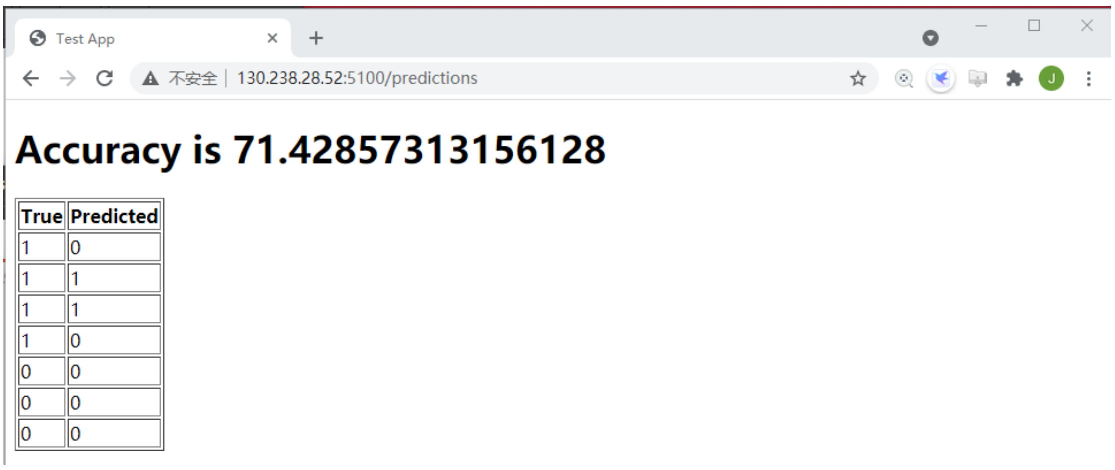


Welcome to the Machine Learning Course.

And the submit page and result is as follows.



130.238.28.52:5100/prediction
不安全 | 130.238.28.52:5100/predictions
提交



Accuracy is 71.42857313156128

True	Predicted
1	0
1	1
1	1
1	0
0	0
0	0
0	0

Task 2

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

1. The portability of the application is better. The environment required by the application is

packaged in a container, which is easy to transplant.
2. Good scalability. We can easily add and reduce workers.
3. The maintenance is more convenient. The application runs in a container, and we can easily find out which container is down.
4. The reliability is better. Applications run in containers isolated from each other, and when the environment or scripts of one application are modified, other applications will not be affected.
5. Resource utilization is high, and multiple containers can run under the same virtual system. Compared with virtual machines, the overhead of containers is smaller.

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

1. The single point of failure still exists. Once the container running the front-end service or the container running the back-end service goes down, the entire application cannot continue to run.
2. It is still not convenient and easy to manually manage multiple dockers. We could not see the status of tasks very easily. We need to log in the second VM to manage.
3. Let users log in production servers directly is not very safe.

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

1. We can run multiple front-end services and back-end services in different containers to improve the fault tolerance of the application.
2. Use k8s or swarm and other tools to help us manage docker, realize automatic scaling, high availability and other functions.

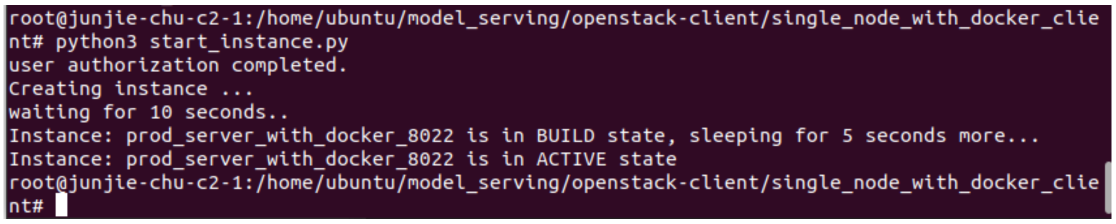
4- What is the difference between horizontal and vertical scalability? Is the strategy adopted in Task-2 follow horizontal or vertical scalability?

Horizontal scalability refers to reduce or increase the number of workers while vertical scalability refers to replace an old worker with a new worker with different performance. In Task-2, the strategy follows the horizontal scalability. It increases and reduces the number of workers.

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

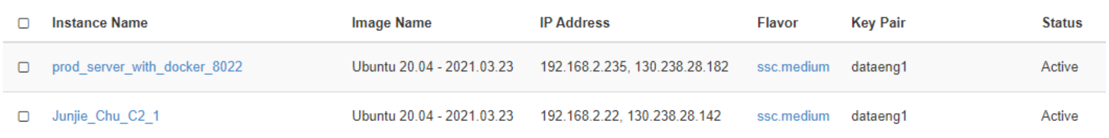
Like task 1, in the task 2, we use a client VM and ‘cloud-init’ to create an instance. There are 2 VMs in total. But in the second VM, we do not run the applications directly. Instead, we run the front-end service, the back-end service and the workers in different containers. We could easily increase or reduce the number of workers via docker. And then we could access the result via a web page.

The process bases on 2 VMs. The first VM is the client VM. We need to install Openstack API on the client VM. After that, we set the cloud-cfg.txt and run start-instance.py on that. ‘start-instance.py’ is used to create an instance on Openstack and ‘cloud-cfg.txt’ is used to configure the environment of the instance. In ‘start-instance.py’, we should set the key name, so that we could log in the instance. In ‘cloud-cfg.txt’, we specify the packages we would like to install and the commands to run. Some commands and packages about docker are included which is different with that in task 1. After we successfully run the start-instance.py, we could see:



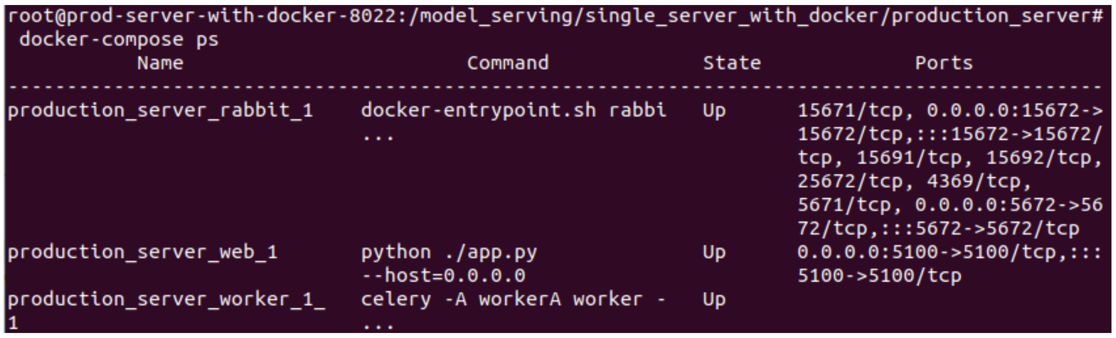
```
root@junjie-chu-c2-1:/home/ubuntu/model_serving/openstack-client/single_node_with_docker_clie
nt# python3 start_instance.py
user authorization completed.
Creating instance ...
waiting for 10 seconds..
Instance: prod_server_with_docker_8022 is in BUILD state, sleeping for 5 seconds more...
Instance: prod_server_with_docker_8022 is in ACTIVE state
root@junjie-chu-c2-1:/home/ubuntu/model_serving/openstack-client/single_node_with_docker_clie
nt#
```

And on Openstack dashboard, we could see:



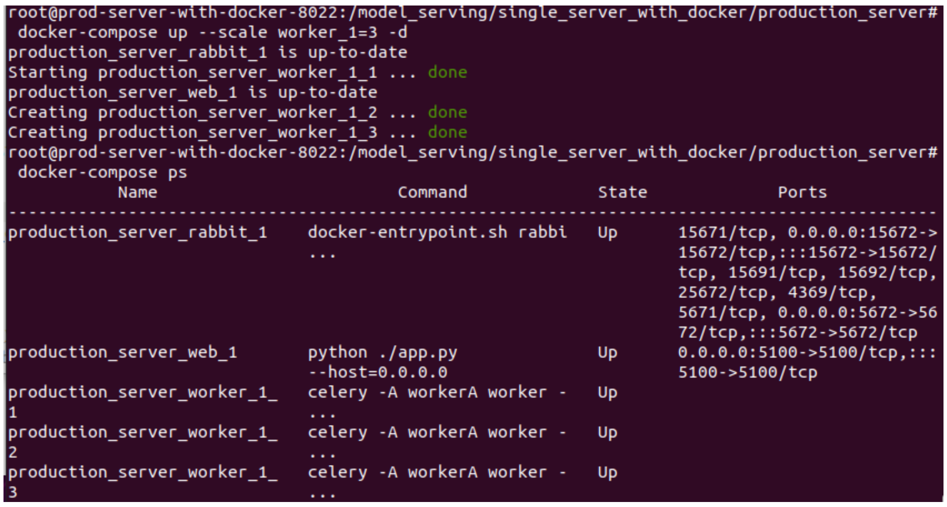
☐	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status
☐	prod_server_with_docker_8022	Ubuntu 20.04 - 2021.03.23	192.168.2.235, 130.238.28.182	ssc.medium	dataeng1	Active
☐	Junjie_Chu_C2_1	Ubuntu 20.04 - 2021.03.23	192.168.2.22, 130.238.28.142	ssc.medium	dataeng1	Active

And we could log in the instance and have a view of the containers:



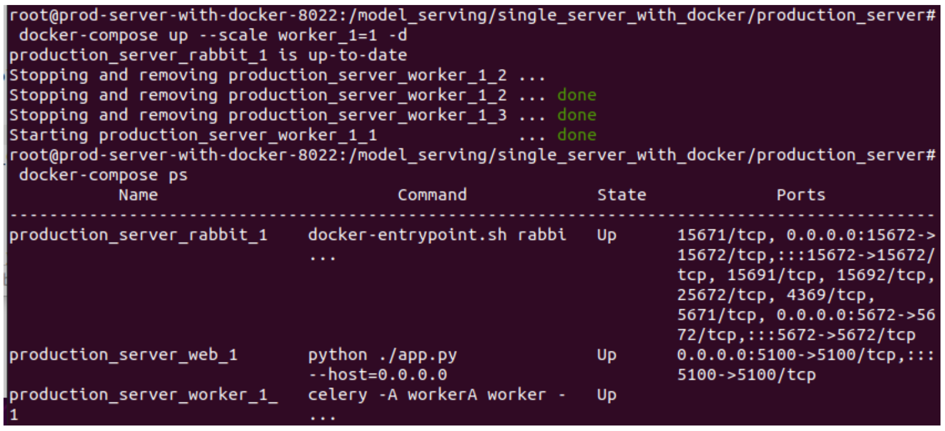
```
root@prod-server-with-docker-8022:/model_serving/single_server_with_docker/production_server#
 docker-compose ps
          Name                        Command             State              Ports
------------------------------------------------------------------------------------------------
production_server_rabbit_1    docker-entrypoint.sh rabbi   Up      15671/tcp, 0.0.0.0:15672->
                              ...                                  15672/tcp,:::15672->15672/
                                                                   tcp, 15691/tcp, 15692/tcp,
                                                                   25672/tcp, 4369/tcp,
                                                                   5671/tcp, 0.0.0.0:5672->56
                                                                   72/tcp,:::5672->5672/tcp
production_server_web_1       python ./app.py              Up      0.0.0.0:5100->5100/tcp,:::
                              --host=0.0.0.0                       5100->5100/tcp
production_server_worker_1_   celery -A workerA worker -   Up
1                             ...
```

Then we could increase the number of workers:



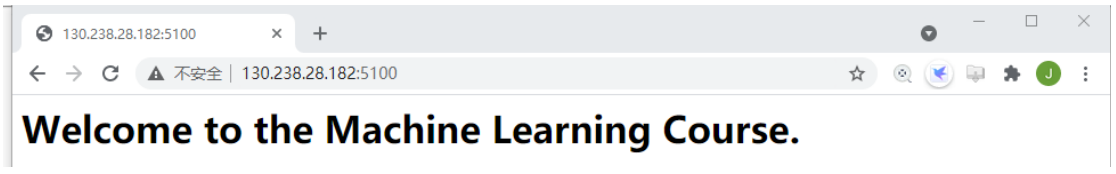
```
root@prod-server-with-docker-8022:/model_serving/single_server_with_docker/production_server#
 docker-compose up --scale worker_1=3 -d
production_server_rabbit_1 is up-to-date
Starting production_server_worker_1_1 ... done
production_server_web_1 is up-to-date
Creating production_server_worker_1_2 ... done
Creating production_server_worker_1_3 ... done
root@prod-server-with-docker-8022:/model_serving/single_server_with_docker/production_server#
 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,:::15672->15672/
                                                                   tcp, 15691/tcp, 15692/tcp,
                                                                   25672/tcp, 4369/tcp,
                                                                   5671/tcp, 0.0.0.0:5672->56
                                                                   72/tcp,:::5672->5672/tcp
production_server_web_1       python ./app.py              Up      0.0.0.0:5100->5100/tcp,:::
                              --host=0.0.0.0                       5100->5100/tcp
production_server_worker_1_   celery -A workerA worker -   Up
1                             ...
production_server_worker_1_   celery -A workerA worker -   Up
2                             ...
production_server_worker_1_   celery -A workerA worker -   Up
3                             ...
```

And then reduce the number of workers:



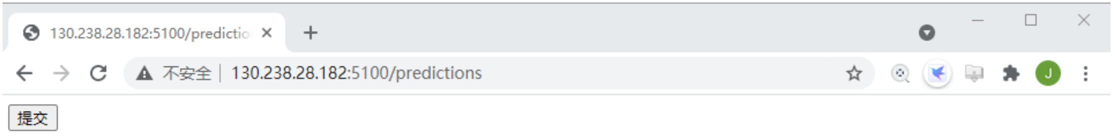
```
root@prod-server-with-docker-8022:/model_serving/single_server_with_docker/production_server#
 docker-compose up --scale worker_1=1 -d
production_server_rabbit_1 is up-to-date
Stopping and removing production_server_worker_1_2 ...
Stopping and removing production_server_worker_1_2 ... done
Stopping and removing production_server_worker_1_3 ... done
Starting production_server_worker_1_1             ... done
root@prod-server-with-docker-8022:/model_serving/single_server_with_docker/production_server#
 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,:::15672->15672/
                                                                   tcp, 15691/tcp, 15692/tcp,
                                                                   25672/tcp, 4369/tcp,
                                                                   5671/tcp, 0.0.0.0:5672->56
                                                                   72/tcp,:::5672->5672/tcp
production_server_web_1       python ./app.py              Up      0.0.0.0:5100->5100/tcp,:::
                              --host=0.0.0.0                       5100->5100/tcp
production_server_worker_1_   celery -A workerA worker -   Up
1                             ...
```

To see if the webpage runs properly:

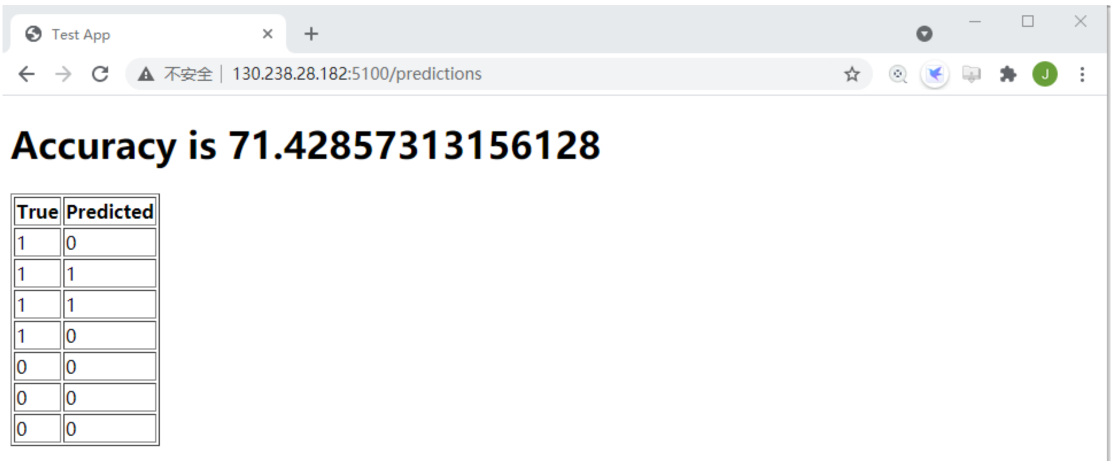

130.238.28.182:5100

不安全 | 130.238.28.182:5100

Welcome to the Machine Learning Course.


130.238.28.182:5100/predictio

不安全 | 130.238.28.182:5100/predictions


Test App

不安全 | 130.238.28.182:5100/predictions

Accuracy is 71.42857313156128

True	Predicted
1	0
1	1
1	1
1	0
0	0
0	0
0	0

Task 3

1 - What is the difference between CloudInit and Ansible?

Cloudinit helps us create instances and configure the system settings of the instance. The operation of Cloudinit is one-off. Ansible could also help us do such tasks. But it could also help us monitor the status of different tasks. We could see the status via the client VM. In addition, we could manage the instances continuously, not just one-off. We could control the instances via ansible in the client VM.

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

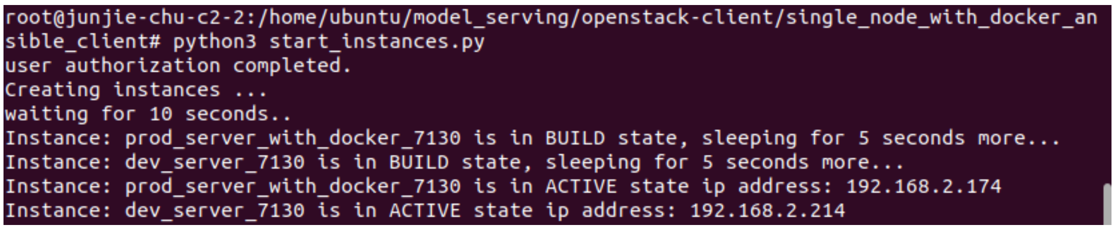
```
users:
  - name: appuser
    sudo: ALL=(ALL) NOPASSWD:ALL
    home: /home/appuser
    shell: /bin/bash
    ssh_authorized_keys:
        - ssh-rsa
byobu_default: system
```

In the .txt files, we could set the settings in dev-server and prod-server. The user name is set as appuser. The 'sudo' sets all the users (includding root) have no password. The 'shell' sets the default directory of shell command. The 'home' set the default home directory. 'ssh_authorized_keys' is used to store the public key. So that the client VM could visit the 2 instances via Ansible using the private key. The 'byobu_default' sets the default session.

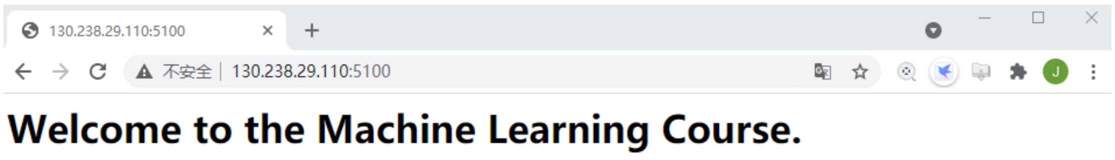
3 - What problem we have solved by using Ansible?

1. We could see the status of tasks of production servers and deployment servers in the client VM via Ansible as well as manage the servers. If something runs wrong, we could run 'ansible-playbook configuration.yml' to restart the tasks.
2. We add a deployment server to deliver the applications so that there is no need to log in the production server directly. The production server is much safer.

Appendix:



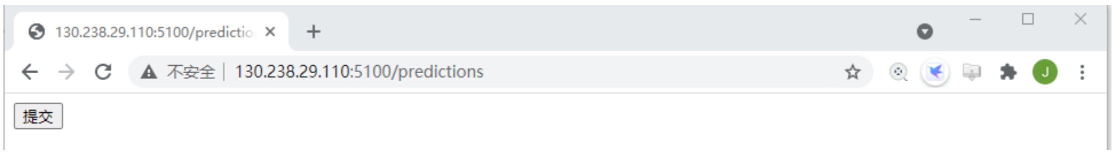
```
root@junjie-chu-c2-2:/home/ubuntu/model_serving/openstack-client/single_node_with_docker_an
sible_client# python3 start_instances.py
user authorization completed.
Creating instances ...
waiting for 10 seconds..
Instance: prod_server_with_docker_7130 is in BUILD state, sleeping for 5 seconds more...
Instance: dev_server_7130 is in BUILD state, sleeping for 5 seconds more...
Instance: prod_server_with_docker_7130 is in ACTIVE state ip address: 192.168.2.174
Instance: dev_server_7130 is in ACTIVE state ip address: 192.168.2.214
```



130.238.29.110:5100

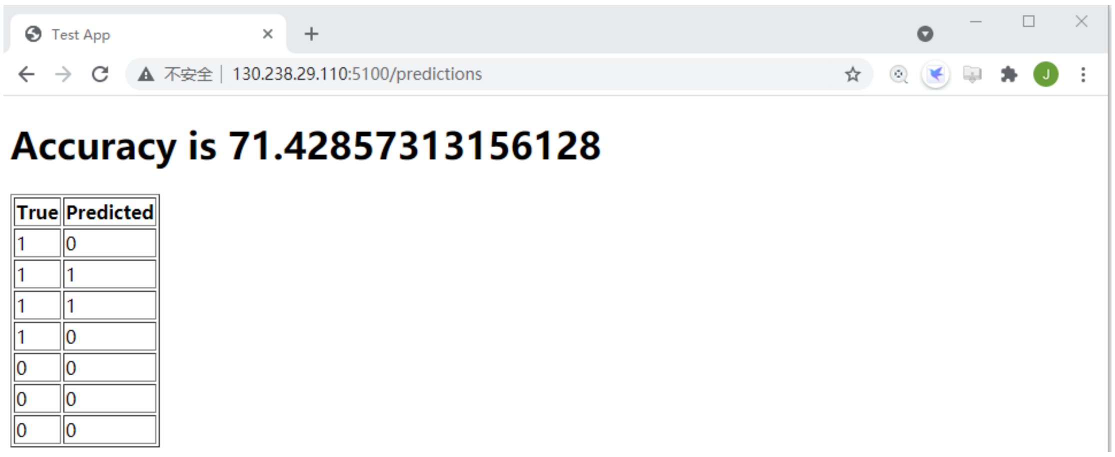
不安全 | 130.238.29.110:5100

Welcome to the Machine Learning Course.



130.238.29.110:5100/predictio

不安全 | 130.238.29.110:5100/predictions

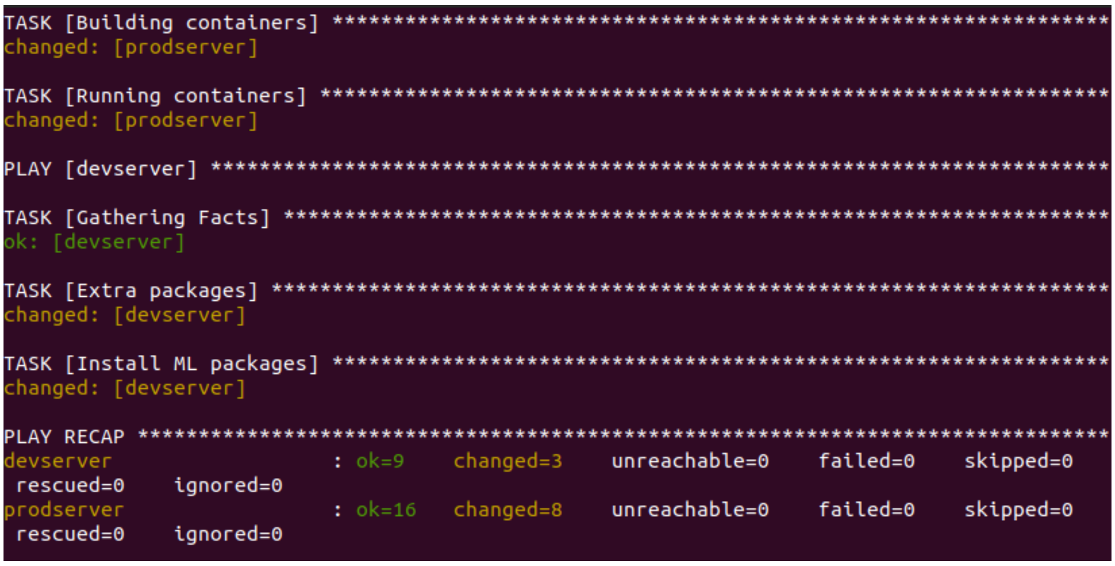


Test App

不安全 | 130.238.29.110:5100/predictions

Accuracy is 71.42857313156128

True	Predicted
1	0
1	1
1	1
1	0
0	0
0	0
0	0

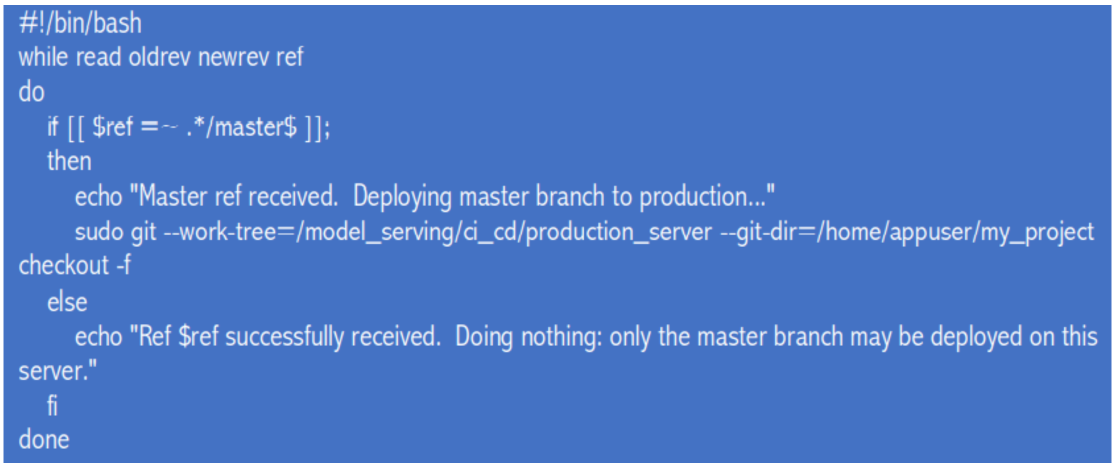


```
TASK [Building containers] *****************************************************************
changed: [prodserver]

TASK [Running containers] ******************************************************************
changed: [prodserver]

PLAY [devserver] ***************************************************************************

TASK [Gathering Facts] *********************************************************************
ok: [devserver]

TASK [Extra packages] **********************************************************************
changed: [devserver]

TASK [Install ML packages] *****************************************************************
changed: [devserver]

PLAY RECAP *********************************************************************************
devserver                  : ok=9    changed=3    unreachable=0    failed=0    skipped=0
 rescued=0    ignored=0
prodserver                 : ok=16   changed=8    unreachable=0    failed=0    skipped=0
 rescued=0    ignored=0
```

Task 4

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

Git hooks are scripts that run automatically every time a particular event occurs in a Git repository. They let you customize Git's internal behavior and trigger customizable actions at key points in the development life cycle.



```
#!/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
server."
    fi
done
```

For the post-receive hook, git passes the old revision's commit hash, the new revision's commit hash, and the reference that is being pushed as standard input to the script.

First, read the standard input. For each ref being pushed, the three pieces of info (old rev, new rev, ref) will be fed to the script, separated by white space, as standard input. We can read this with *'while read oldrev newrev ref'*.

For a master branch push, the ref object will contain something that looks like refs/heads/master. We can check to see if the ref the server is receiving has this format by using *'if [[$ref =~ .*/master$]]'*.

After that, we make sure the receive messages are what we want and we can type out the checkout command to change the paths on this machine by using *'sudo git --work-tree=/model_serving/ci_cd/production_server --git-dir=/home/appuser/my_project checkout -f'*.

For server-side hooks, git can pass messages back to the client. Anything sent to standard out will be redirected to the client. This gives us an opportunity to explicitly notify the user about what decision has been made. We add 'echo' statements here.

We should also add some text describing what situation was detected, and what action was taken. An else block is added to notify the user when a non-master branch was successfully received, even though the action won't trigger a deploy.

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

By running *git init –bare,* we create an empty git repository in production server. This repository is used to receive the scripts pushed by the deployment server. After we git commit something in the deployment server, then git push, the documents will be pushed into the empty git repository in production server.

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

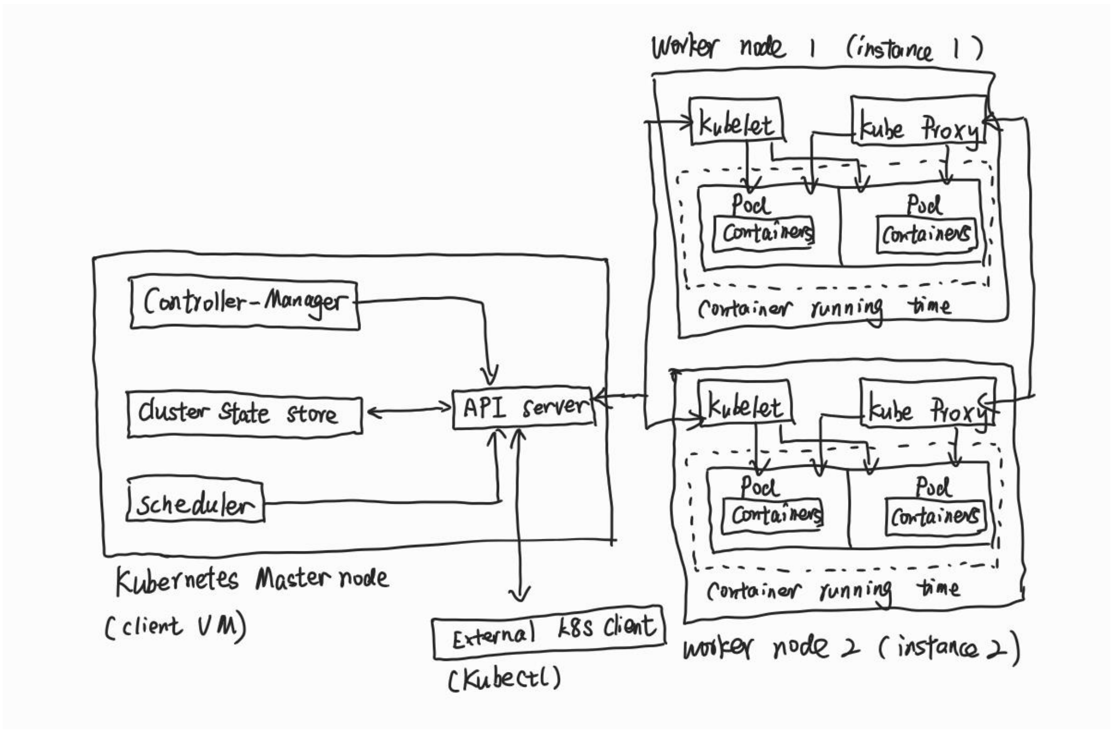
Hints - Read the link available in the task or access sample scripts available in the hooks directory.

1. applypatch-msg
It can edit the commit message file and is often used to verify or actively format a patch's message to a project's standards. A non-zero exit status aborts the commit.
2. commit-msg
It can be used to adjust the message after it has been edited in order to ensure conformity to a standard or to reject based on any criteria. It can abort the commit if it exits with a non-zero value.
3. update
This is invoked on the remote repository. This is run on the remote repo once for each ref being pushed instead of once for each push. A non-zero status will abort the process. This can be used to make sure all commits are only fast-forward, for instance.
4. post-receive
This is invoked on the remote repository. This is run on the remote when pushing after the all refs have been updated. It does not take parameters, but receives info through stdin in the form of "<old-value> <new-value> <ref-name>". Because it is called after the updates, it cannot abort the process.
5. post-update
This is invoked on the remote repository. This is run only once after all of the refs have been pushed. It is similar to the post-receive hook in that regard, but does not receive the old or new values. It is used mostly to implement notifications for the pushed refs.
6. pre-rebase
It is called when rebasing a branch. It is mainly used to halt the rebase if it is not desirable.

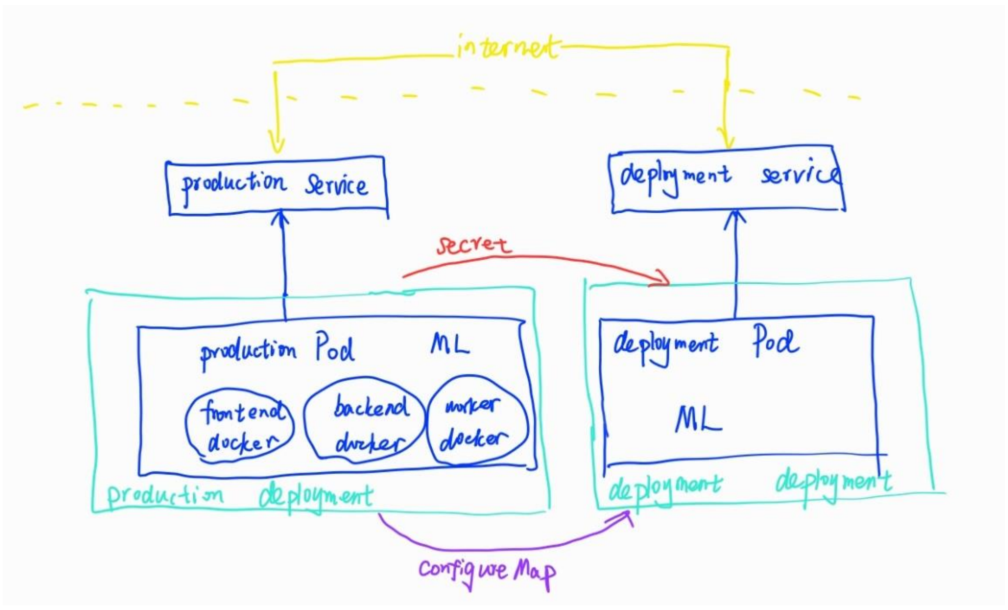
4- How deployment of multiple servers (in task 3) can be achieved with Kubernetes? Draw a diagram of the deployment highlighting the nodes, services, configuration map and secret.

1 Components of the architecture:
API server is the component that is responsible for processing the REST requests, validating them, and performing appropriate CRUD operations on corresponding abstract Kubernetes objects.
Cluster state store is a persistent storage instance which stores the state of all the abstract Kubernetes objects configured in the system. The cluster state store has support for watch functionality.
Controller Manager is the component of master that runs controllers. Controllers run loops and monitor the actual cluster state and state represented in the abstract Kubernetes objects.
Scheduler is the component of the master responsible for allocating physical resources on the cluster to run applications/jobs added to the abstract data store.
Kubectl is a command line tool that is intended to be used by an end-user responsible for managing application deployments.
Kubelet is the component of worker responsible for making sure that the containers scheduled by the master on this node are running and are healthy.
Container runtime is the software that is responsible for running containers.
Kube proxy is the component of Worker responsible for maintaining network rules on the worker and performing connection forwarding. This essentially enables efficient and

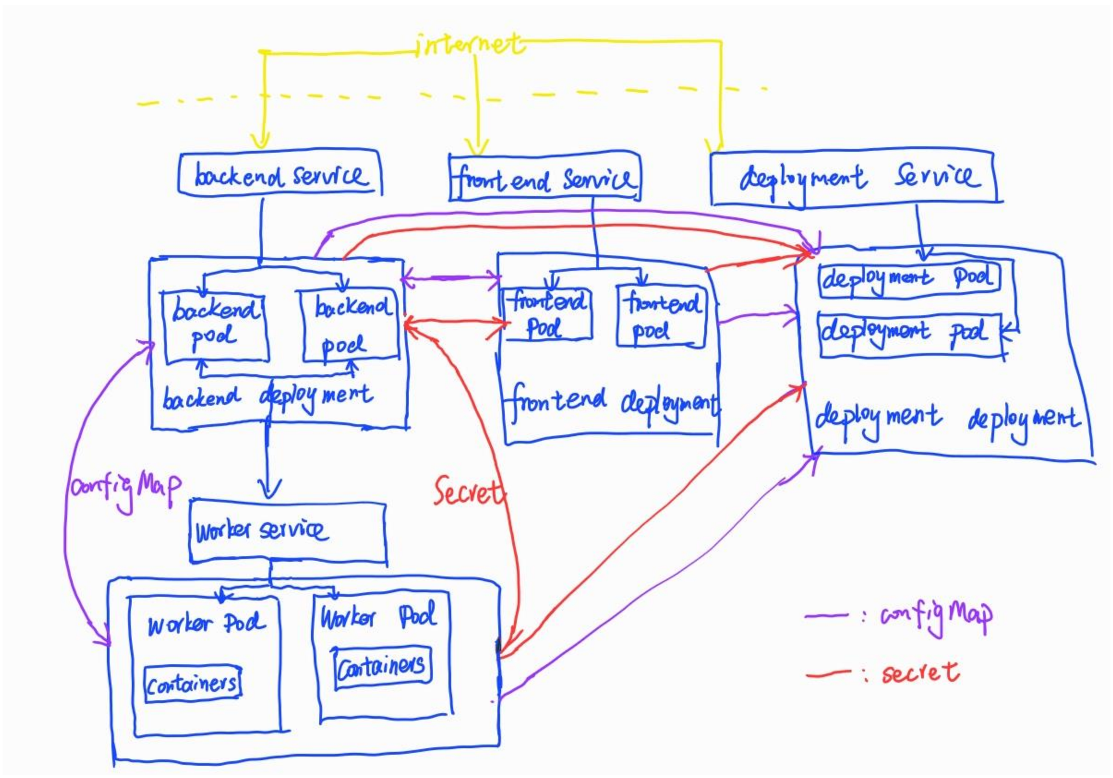
effective communication throughout the cluster.
External application traffic will get redirected to the appropriate container through these components.
2 Achitecture:
Firstly, we have to set up a Kubernetes cluster using the three VMs. One of them is a master node and 2 of them are working nodes. The architecture could be shown as follows:



The client VM will be used as the master node of the Kubernetes cluster. The 2 instance it creates will be used as working nodes. The pods will run in the nodes. Once one of the nodes crashes, the pods running on the node will restart in another node.
3 Deployment diagrams
After that, I would like to describe my diagram. I have 2 solutions but I am not sure which one is better, so I would like to introduce both of them.



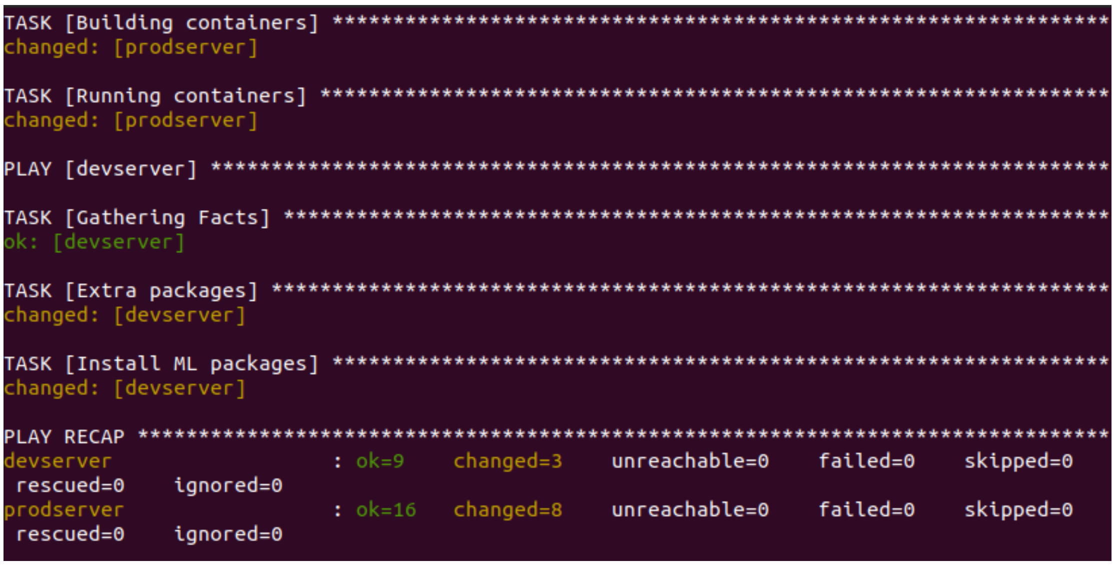
Here, I first created a Service of type LoadBalancer which exposes the deployment pods. Loadbalancer type provides an External-IP, through which one could access the deployment services externally. Next, I created the Deployment object configured to contain the replicas of the deployment pods. After this, I injected the environment variable from configMap and secrets. The production service's values like host and name in this ConfigMap object and username and password in Secret object. So the deployment pods could communicate with the production pods.
The process of deploying the production part are similar to deployment part. In production pods, there are some containers running. The containers are running as front-end, back-end and worker.



In the second solution, I split the production service into 3 sub-services. Each service will have 2 replicas(pods).
In the worker part, there are some containers where some machine learning programs are running. In the backend part, there are some containers where some Celery and RabbitMQ servers are running. In the frontend part, there are some containers where some Flask based web application are running.
The values like host and name are stored in this ConfigMap object and some values like username and password are stored in Secret object. The actual relationship is shown in the diagram.
The deployment part should have the values from frontend part, backend part and worker part. So that the deployment service could access the three parts. We could push the updates of the three parts via deployment service.
The worker part and the backend part should have the values about each other so that they could communicate with each other.
The backend part and the frontend part should have the values about each other, too. In

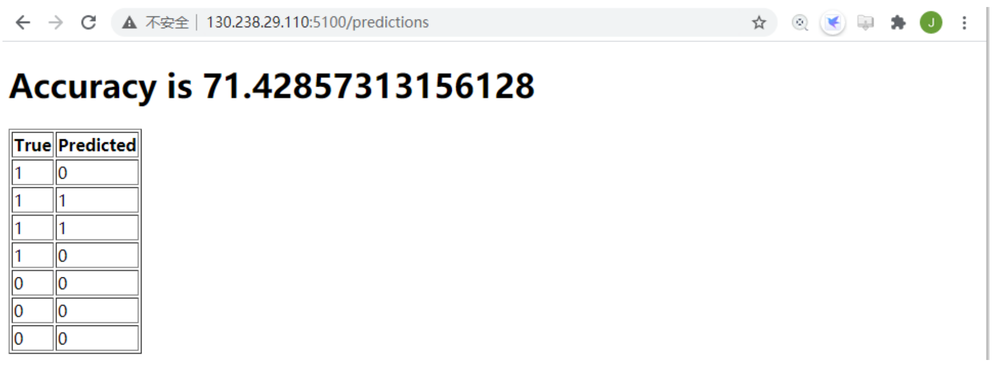
this case, they are able to communicate with each other.
All the information injection are based on the ConfigMap object and Secret object.
The other configurations and steps are similar to those in solution 1.

5- Write half a page summary about the task and add screenshots if needed.
The process bases on 3 VMs. The first VM is the client VM. We need to install Openstack API on the client VM. After that, we set the dev-cloud-cfg.txt and prod-cloud-cfg.txt and run start-instance.py on that. 'start-instance.py' is used to create 2 instances on Openstack and dev-cloud-cfg.txt and prod-cloud-cfg.txt is used to configure the environment of the instances. In 'start-instance.py', we should set the key name, so that we could log in the instance. In dev-cloud-cfg.txt and prod-cloud-cfg.txt, we specify the variables we would like to use, such as password, the key pairs and user name. We could visit the production server and deployment server on the client VM vis SSH key.
Different from the previous 2 tasks, in the task, we will use Ansible to deploy the 2 servers. First, we add some contents including the ip addresses of the 2 servers in the Ansible inventory file. Then we export some environment variables and settings. Then we run ansible-playbook. How the servers will be deployed is included in the configuration.yml. After that, we could monitor the status of tasks in the client VM. If something wrong, we could re-run the playbook.



```
TASK [Building containers] *****************************************************************
changed: [prodserver]

TASK [Running containers] ******************************************************************
changed: [prodserver]

PLAY [devserver] ***************************************************************************

TASK [Gathering Facts] *********************************************************************
ok: [devserver]

TASK [Extra packages] **********************************************************************
changed: [devserver]

TASK [Install ML packages] *****************************************************************
changed: [devserver]

PLAY RECAP *********************************************************************************
devserver                  : ok=9    changed=3    unreachable=0    failed=0    skipped=0
 rescued=0    ignored=0
prodserver                 : ok=16   changed=8    unreachable=0    failed=0    skipped=0
 rescued=0    ignored=0
```

Based on task 3, to use git hooks, we modify some settings about ssh to make the deployment server communicate with the production server. And then we install some tools and initialize the git repository on the two servers. We need to initialize a 'bare' repository on production server and a repository on deployment server. Then modify the post-receive. After we finish all settings, we are able to run a new version of program on production server without logging it, which make the server much safer. We could change the code or models on development server and push them to the production server without logging to the production server. Thus, we realize 'Continuous Integration and Continuous Delivery'.
At first, the output is:

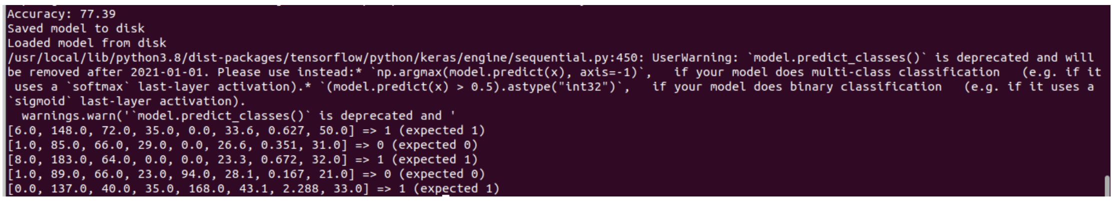


不安全 | 130.238.29.110:5100/predictions

Accuracy is 71.42857313156128

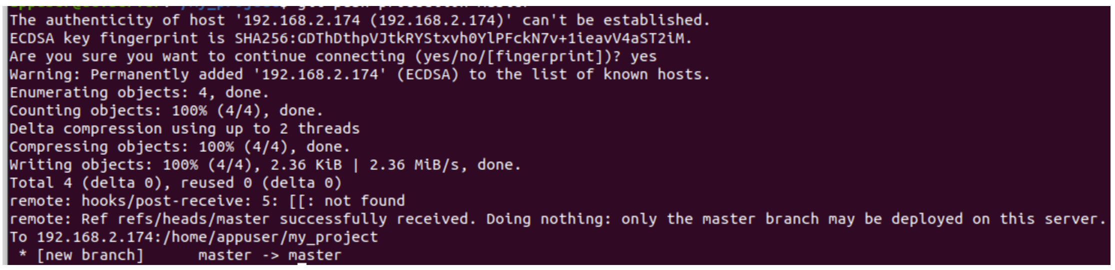
True	Predicted
1	0
1	1
1	1
1	0
0	0
0	0
0	0

Modify some parameters and train the new model:



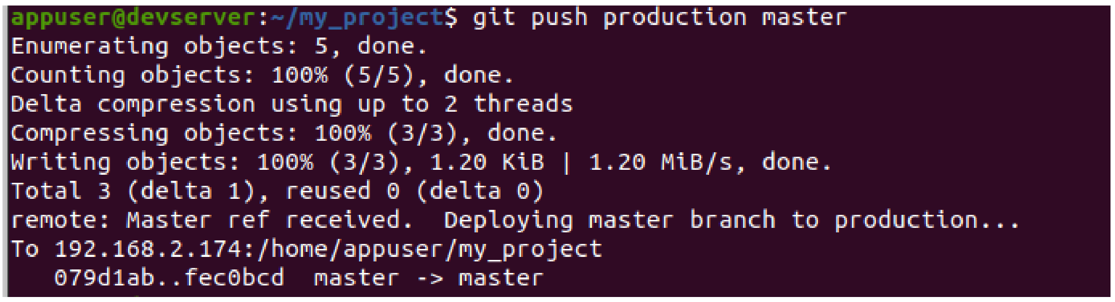
```
Accuracy: 77.39
Saved model to disk
Loaded model from disk
/usr/local/lib/python3.8/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will
be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it
 uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a
`sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 1 (expected 1)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1)
```

Once we finish setting the git hooks, and push wrong branches, the output is:



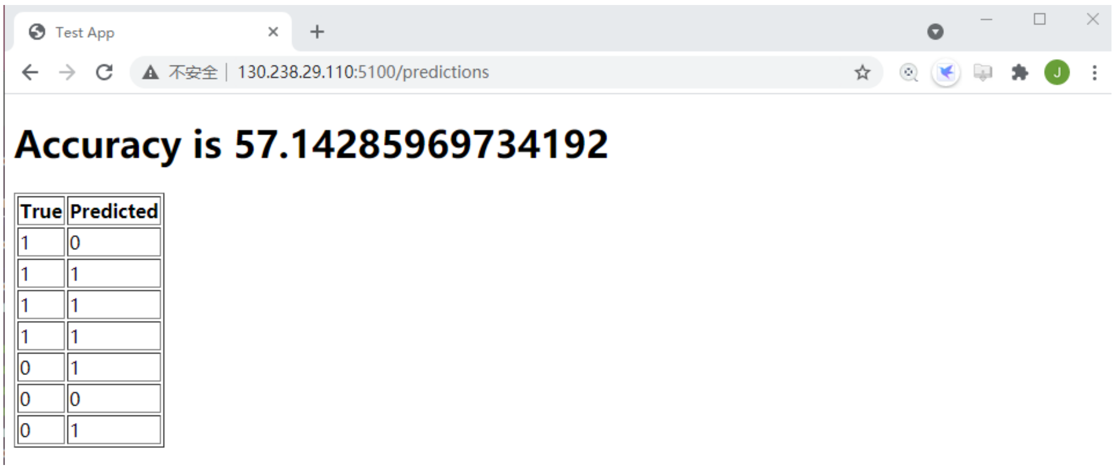
```
The authenticity of host '192.168.2.174 (192.168.2.174)' can't be established.
ECDSA key fingerprint is SHA256:GDThDthpVJtkRYStxvh0YlPFckN7v+1ieavV4aST2iM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.2.174' (ECDSA) to the list of known hosts.
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
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: hooks/post-receive: 5: [[: not found
remote: Ref refs/heads/master successfully received. Doing nothing: only the master branch may be deployed on this server.
To 192.168.2.174:/home/appuser/my_project
 * [new branch]      master -> master
```

If the branch is the master, the output is:



```
appuser@devserver:~/my_project$ git push production master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.20 KiB | 1.20 MiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Master ref received.  Deploying master branch to production...
To 192.168.2.174:/home/appuser/my_project
   079d1ab..fec0bcd  master -> master
```

And we could visit the webpage:



Test App

不安全 | 130.238.29.110:5100/predictions

Accuracy is 57.14285969734192

True	Predicted
1	0
1	1
1	1
1	1
0	1
0	0
0	1