Xavier Goovaerts
Jeremy Moore
COSC 480 Cloud Computing
Fall 2019

REPOSITORY: https://github.com/xfgoovaerts/DroneProjectPartOne

WHAT WE GOT TO WORK:
- Video is now working within the Flask web application virtual environment, but lag must be explored to determine if it was a one-off issue due to a specific connection or caused by the virtual environment.
- Message about the required and optional arguments prints on the command line if no arguments are provided.
- Command line accepts arguments with spaces if enclosed in double- or single-quotes.
- Test_flight.py functions as required (video but no command line arguments or database connection).
- As before, flight data is written to a local DSE database running in a docker container.
- A three-node cluster has been created in AWS. We created a python script (test_cassandra.py) to generate fake data and can send it to the AWS cluster.
- Added button to web application to cancel a flight (valid = false) and finish a flight (valid = true). Sent query to Cassandra to update the valid field after button clicked.
- Added command line prompt after script is stopped to determine if flight was successful. Sent query to Cassandra to update the valid field after prompt is answered.

WHAT WE TRIED HARD BUT DIDN'T GET TO WORK:
- Cannot connect to both the drone and the AWS cluster at the same time yet.
- Need to explore if the video lag we experienced is due to the web applications virtual environment or was it due to a slow network connection.
- AWS Cassandra cluster needs to be manually setup again after the instances are restarted. This involves manual updates to the Cassandra.yaml file. I suspect this can be changed in a DSE configuration file. Just have to find it.
- Have not been able to explore improving the accuracy of the X, Y, Z data.

Xavier Goovaerts
Jeremy Moore
COSC 480 Cloud Computing
Fall 2019

WIKI:

The instructions we followed to create both the single-node local database in a docker cluster and the three-node AWS cluster were included in the previous report.

1. OPEN AWS CLUSTER TO REMOTE ACCESS

To connect to the AWS cluster from outside the VPC, each nodes **RPC_ADDRESS** must be set to the EC2 instances public IP address.
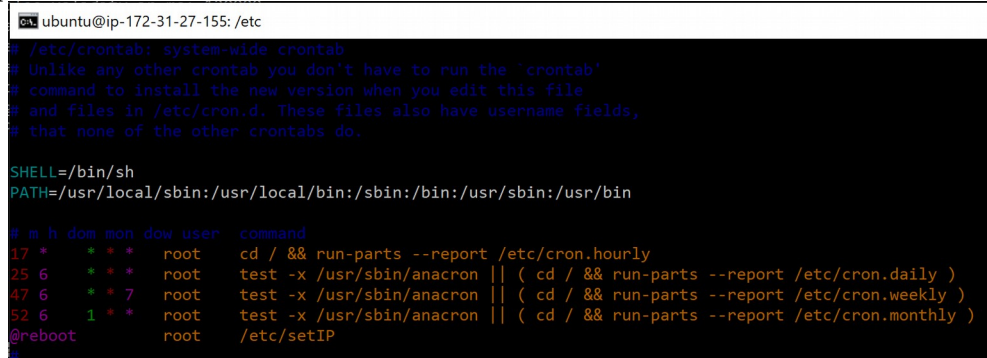
To set the RPC address for a node:
A) SSH into the node.

B) Edit the cassandra.yaml file by typing:
    sudo nano */etc/dse/cassandra/cassandra.yaml*
    It's important to use sudo or you will not be able to save your changes.

C) By default the file does not have an **RPC_ADDRESS** (at least it did not in our case), so you must add a line to the yaml file.
    I don't believe it matters where you add the line, but I always added it right about RPC_KEEPALIVE: true.
    The line that must be added is: **RPC_ADDRESS: <public_ip>** where <public_ip> is the public IP address of the EC2 instance.

To create script to run during instance boot to update the public ip in the YAML file:
A) Navigate to the etc directory
    "cd /etc"
B) Create a file for your script
    "sudo touch setIP"
C) Edit your file
    "vi setIP"
D) Paste the bash script in the file (press I to enter insert mode)
    #!/bin/bash
    ip="$(sudo curl v4.ifconfig.co)"
    grep -q "RPC_ADDRESS: " /etc/dse/cassandra/cassandra.yaml &&
        sudo sed -i s/"RPC_ADDRESS: .*"/"RPC_ADDRESS: ${ip}"/ /etc/dse/
    cassandra/cassandra.yaml ||
        sudo echo "RPC_ADDRESS: ${ip}" >>
    /etc/dse/cassandra/cassandra.yaml
E) Since the file is read only, save it using this command (press Esc to exit insert mode)
        ":w !sudo tee %"

F) Press O, Enter, and then :q!
G) Make the script executable
"sudo chmod +x setIP"
H) Edit the crontab file to make the script run on reboot
"vi crontab"
I
"@reboot root /etc/setIP"
Esc
:w !sudo tee %
O
Enter
:q!

Note this process must be done on each node.

More information about the cassandra.yaml file can be found in the DSE
documentation here:
https://docs.datastax.com/en/archived/cassandra/3.0/cassandra/configuration/
configCassandra_yaml.html

Xavier Goovaerts
Jeremy Moore
COSC 480 Cloud Computing
Fall 2019

2. CONNECT TO AWS CLUSTER FROM PYTHON SCRIPT

At the moment, we are using the native Cassandra python drivers to connect to the DSE instance by importing the following classes:

```
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider
```

These can easily be converted to the DSE python driver using the instructions found here: https://docs.datastax.com/en/developer/python-dse-driver/1.1/getting_started/

Because the tutorial we followed included setting credentials on the cluster, the connect command must include an **auth_provider**. Example code is shown below:

```
def connect_to_db():
    global session

    auth_provider=PlainTextAuthProvider(
        username='cassandra',
        password='eagles29')

    cluster = Cluster(
        auth_provider=auth_provider,
        contact_points=['3.230.244.15', '3.228.63.63',
        '3.231.140.68'])

    try:
        session = cluster.connect('competition')
        print('Connected to Cassandra cluster.')

    except:
        print('Cannot connect to database. Exiting ...')
        exit(1)
```

The contact points are the public IP addresses of the AWS EC2 instances running each node of the cluster.

The try/except isn't strictly necessary, but without it the flight would continue without being recorded in the database and the console output would include a series of errors as queries execution is executed.

We thought it would be better if the program quit, so flights are not inadvertently flown without being recorded.