Polytechnique Montréal
Department of Computer Engineering

Ndeye Anna NDIAYE - 1764819
Papy Boweya Ikoelenga - 1826188

May 4, 2017

LOG8415 - Report for TP3
Scaling Databases and Implementing Cloud Patterns

1 Introduction

For this assignment, we will Setup a MysQL Cluster on Amazon EC2. Mysql Cluster is one of the database cluster that allows people to overcome challenges like Web , communication, cloud services on topics like scalability, uptime and agility. The other part of this assignment concerns cloud patterns. Cloud patterns are generally used as template solutions for frequent problems that occur repetitively. Here we are asked to design and implement two common Cloud Patterns which are the Proxy Pattern and the Gatekeeper Pattern.

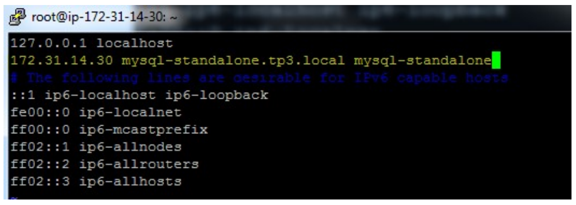
2 Setting Up MySQL and MySQL Cluster

2.1 Setting Up MySQL Standalone

In this section, we will create a t2.micro instance on Amazon AWS and install MySQL standalone on it. After downloading and installing, will configure and manage MySQL running Ubuntu 16.04 LTS.

2.1.1 Installation and Configuration

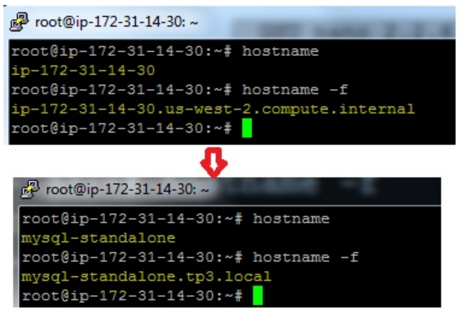
First of all, we will set the system's hostname and fully qualified domain name (FQDN) to our instance using the command **echo "mysql-standalone" > /etc/hostname && hostname -F /etc/hostname**. The first command set the hostname, and the second one set fully qualified domain name (FQDN).
We will need also to create static associations between IP addresses and hostnames by updating the /etc/hosts file as below:



```
127.0.0.1 localhost
172.31.14.30 mysql-standalone.tp3.local mysql-standalone
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Figure 1: Updating /etc/hosts file on MySQL standalone server

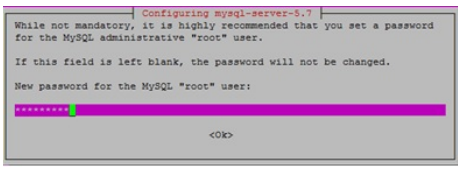
Before installing Mysql, we need to update the system using the following commands : **sudo apt-get -y update && apt-get -y upgrade** and **sudo apt-get -y install mysql-server**
The figure below shows the installation process. The system will prompt us to set a password for the root user.



```
root@ip-172-31-14-30: ~
root@ip-172-31-14-30:~# hostname
ip-172-31-14-30
root@ip-172-31-14-30:~# hostname -f
ip-172-31-14-30.us-west-2.compute.internal
root@ip-172-31-14-30:~#
```

```
root@ip-172-31-14-30: ~
root@ip-172-31-14-30:~# hostname
mysql-standalone
root@ip-172-31-14-30:~# hostname -f
mysql-standalone.tp3.local
root@ip-172-31-14-30:~#
```

Figure 2: Setting the Hostname & FQDN for MySQL standalone server



```
Configuring mysql-server-5.7
While not mandatory, it is highly recommended that you set a password
for the MySQL administrative "root" user.

If this field is left blank, the password will not be changed.

New password for the MySQL "root" user:

*********

<Ok>
```

Figure 3: Setting password for the MySQL root user

After the completion of the installation, we need to access to MySQL by using the mysql client, which is the standard tool for interacting with MySQL. The MySQL client is accessed through a terminal. Then, we will use the following commands to connect and manage MySQL: **mysql -u root -p** for logging into Mysql as the root user, **create database tp3db;** to create a test database, **create user 'testuser'@'localhost' identified by 'password';** to create a user and finally grant all the permissions with the command **grant all on testdb.* to 'testuser'; use testdb;**

2.2 Setting Up MySQL Cluster

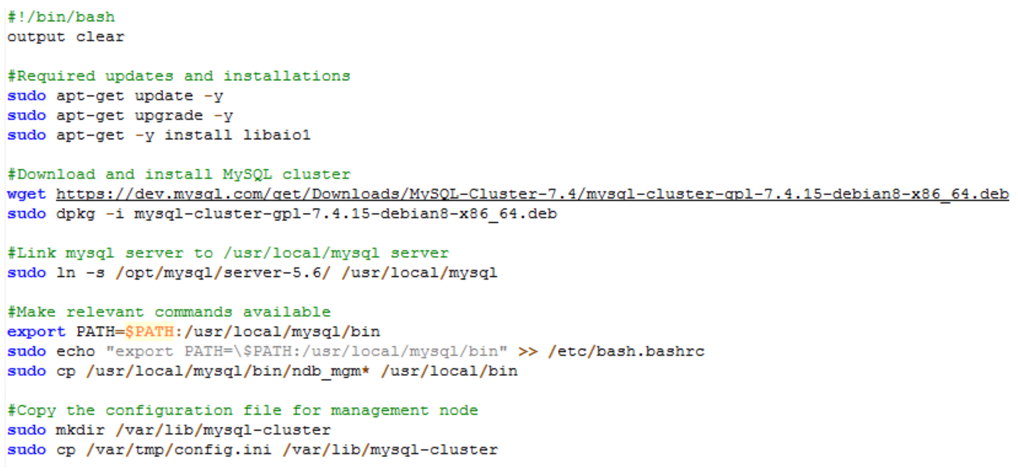
MySQL Cluster is a technology providing shared-nothing clustering and auto-sharding functions for database management system. It is designed to provide high availability and throughput with low latency, while allowing for near linear scalability. MySQL Cluster is implemented through NDB(Network Database) storage engine which were made using a distributed, shared-nothing architecture making it behave differently from InnoDB(a high-reliability and high-performance storage engine for MySQL) in a number of ways. MySQL cluster has several major components which typically have their own server (a node). Those components are :

- Management node (distributed configuration, meta-data, ...)
- Data node (back-end)
- SQL node (front-end)

Generally, application requests are sent to the SQL nodes, which then talk to the data nodes and coordinate each other via management node to recover the content. So among our 5 t2.micro instances, we have 1 Management Node and 2 SQL/Data Nodes.

2.3 Management Node

We used the below script to set up the management node. This script download and install the debian MySQL cluster package, version 7.4.15 which is not the latest but we wanted to use a confirmed stable one. Next, we create a link between the location of MySQL server 5.6 and the directory `/usr/local/mysql` and made the commands `ndb_mgm*` available from anywhere.

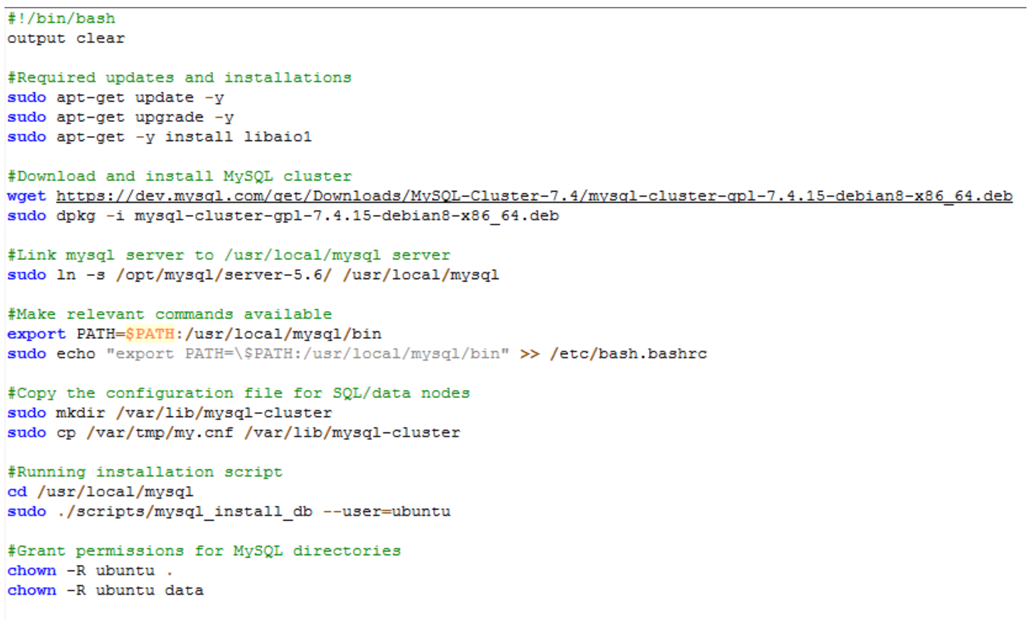


```
#!/bin/bash
output clear

#Required updates and installations
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get -y install libaio1

#Download and install MySQL cluster
wget https://dev.mysql.com/get/Downloads/MySQL-Cluster-7.4/mysql-cluster-gpl-7.4.15-debian8-x86_64.deb
sudo dpkg -i mysql-cluster-gpl-7.4.15-debian8-x86_64.deb

#Link mysql server to /usr/local/mysql server
sudo ln -s /opt/mysql/server-5.6/ /usr/local/mysql

#Make relevant commands available
export PATH=$PATH:/usr/local/mysql/bin
sudo echo "export PATH=\$PATH:/usr/local/mysql/bin" >> /etc/bash.bashrc
sudo cp /usr/local/mysql/bin/ndb_mgm* /usr/local/bin

#Copy the configuration file for management node
sudo mkdir /var/lib/mysql-cluster
sudo cp /var/tmp/config.ini /var/lib/mysql-cluster
```

Figure 4: Script for configuration of Management Node

2.4 SQL/ Data Nodes

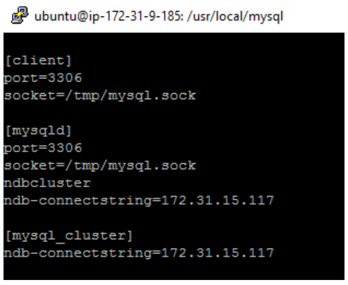
For SQL/data nodes, we made a similar script which instead of `config.ini`, use `my.cnf` configuration file.



```
#!/bin/bash
output clear

#Required updates and installations
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get -y install libaio1

#Download and install MySQL cluster
wget https://dev.mysql.com/get/Downloads/MySQL-Cluster-7.4/mysql-cluster-gpl-7.4.15-debian8-x86_64.deb
sudo dpkg -i mysql-cluster-gpl-7.4.15-debian8-x86_64.deb

#Link mysql server to /usr/local/mysql server
sudo ln -s /opt/mysql/server-5.6/ /usr/local/mysql

#Make relevant commands available
export PATH=$PATH:/usr/local/mysql/bin
sudo echo "export PATH=\$PATH:/usr/local/mysql/bin" >> /etc/bash.bashrc

#Copy the configuration file for SQL/data nodes
sudo mkdir /var/lib/mysql-cluster
sudo cp /var/tmp/my.cnf /var/lib/mysql-cluster

#Running installation script
cd /usr/local/mysql
sudo ./scripts/mysql_install_db --user=ubuntu

#Grant permissions for MySQL directories
chown -R ubuntu .
chown -R ubuntu data
```

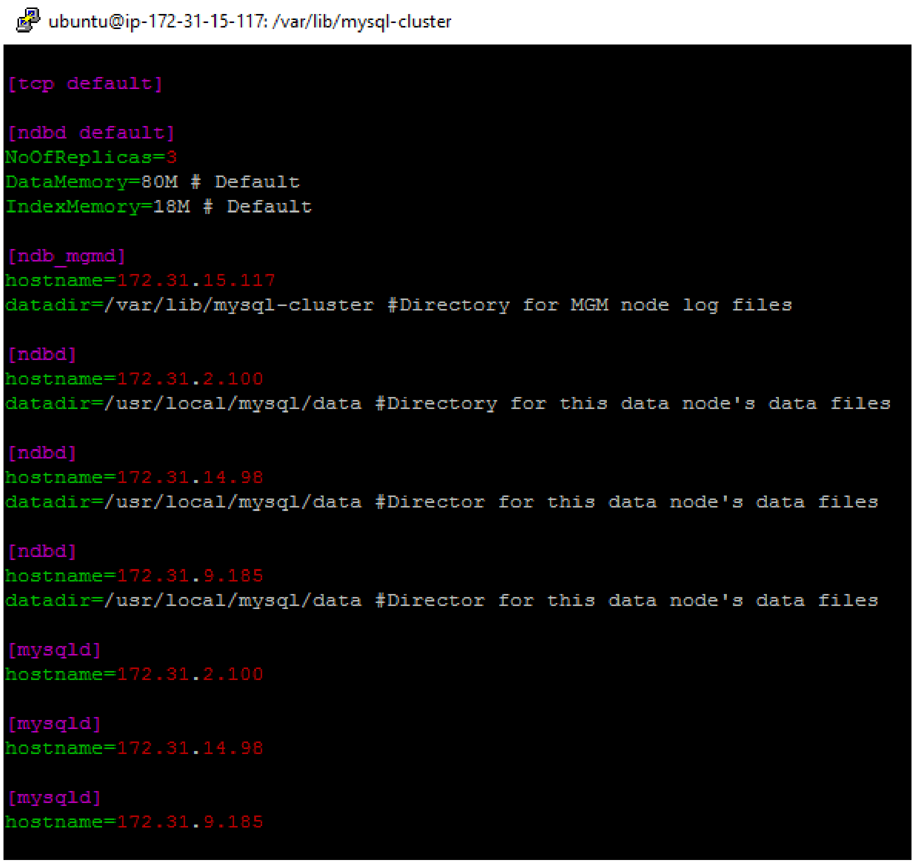
Figure 5: Script for configuration of Data/SQL Nodes

The following pictures show the contents of `config.ini` and `my.cnf` configuration files.The `ndbd` sections indicate the behavior of the cluster's data nodes and the `mysqld` sections concerns SQL nodes which in our case are the same so they have the same hostname. `ndbd_mgmd` section is about the management node.



```
ubuntu@ip-172-31-9-185: /usr/local/mysql

[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
ndbcluster
ndb-connectstring=172.31.15.117

[mysql_cluster]
ndb-connectstring=172.31.15.117
```

Figure 6: /Configuration file my.cnf for Data/SQL Nodes

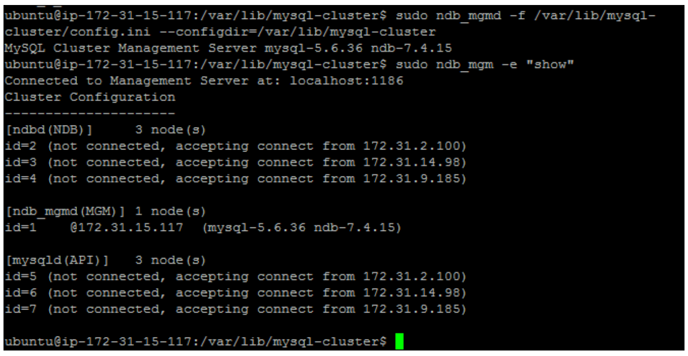


```
ubuntu@ip-172-31-15-117: /var/lib/mysql-cluster

[tcp default]

[ndbd default]
NoOfReplicas=3
DataMemory=80M # Default
IndexMemory=18M # Default

[ndb_mgmd]
hostname=172.31.15.117
datadir=/var/lib/mysql-cluster #Directory for MGM node log files

[ndbd]
hostname=172.31.2.100
datadir=/usr/local/mysql/data #Directory for this data node's data files

[ndbd]
hostname=172.31.14.98
datadir=/usr/local/mysql/data #Director for this data node's data files

[ndbd]
hostname=172.31.9.185
datadir=/usr/local/mysql/data #Director for this data node's data files

[mysqld]
hostname=172.31.2.100

[mysqld]
hostname=172.31.14.98

[mysqld]
hostname=172.31.9.185
```

Figure 7: Configuration file config.ini for Management Node

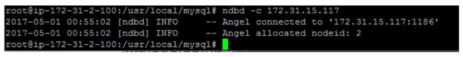
3 Starting MYSQL Cluster

Once the configuration properly set up, we can start MySQL Cluster. To do so, we begin with the management node which read the cluster configuration file and share informations with the other members of cluster while logging different activities going on in the clustered environment.Then we start each of the data nodes with the `ndbd` command. The "show" option in `ndb_mgm` command let us see the different SQL and data nodes supposed to connect to the cluster. Before starting data nodes, they appear as not connected, but once data nodes start, all of them become connected.



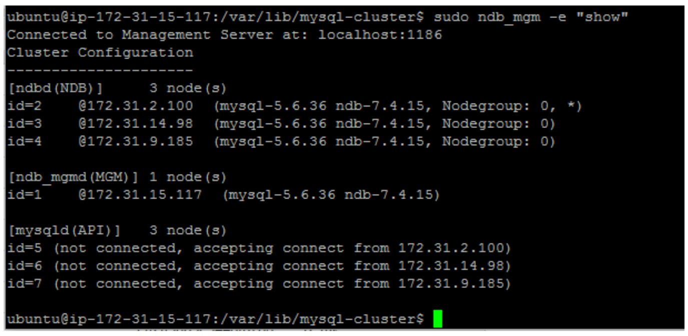
```
ubuntu@ip-172-31-15-117:/var/lib/mysql-cluster$ sudo ndb_mgmd -f /var/lib/mysql-
cluster/config.ini --configdir=/var/lib/mysql-cluster
MySQL Cluster Management Server mysql-5.6.36 ndb-7.4.15
ubuntu@ip-172-31-15-117:/var/lib/mysql-cluster$ sudo ndb_mgm -e "show"
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     3 node(s)
id=2 (not connected, accepting connect from 172.31.2.100)
id=3 (not connected, accepting connect from 172.31.14.98)
id=4 (not connected, accepting connect from 172.31.9.185)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @172.31.15.117  (mysql-5.6.36 ndb-7.4.15)

[mysqld(API)]   3 node(s)
id=5 (not connected, accepting connect from 172.31.2.100)
id=6 (not connected, accepting connect from 172.31.14.98)
id=7 (not connected, accepting connect from 172.31.9.185)

ubuntu@ip-172-31-15-117:/var/lib/mysql-cluster$
```

Figure 8: Starting cluster on Management Node



```
root@ip-172-31-2-100:/usr/local/mysql# ndbd -c 172.31.15.117
2017-05-01 00:55:02 [ndbd] INFO     -- Angel connected to '172.31.15.117:1186'
2017-05-01 00:55:02 [ndbd] INFO     -- Angel allocated nodeid: 2
root@ip-172-31-2-100:/usr/local/mysql#
```

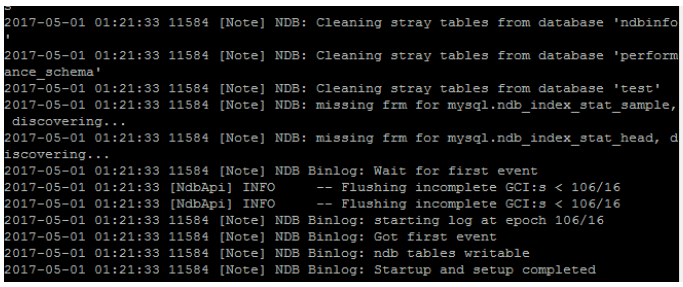
Figure 9: Starting cluster on SQL/Data Nodes



```
ubuntu@ip-172-31-15-117:/var/lib/mysql-cluster$ sudo ndb_mgm -e "show"
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     3 node(s)
id=2    @172.31.2.100  (mysql-5.6.36 ndb-7.4.15, Nodegroup: 0, *)
id=3    @172.31.14.98  (mysql-5.6.36 ndb-7.4.15, Nodegroup: 0)
id=4    @172.31.9.185  (mysql-5.6.36 ndb-7.4.15, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @172.31.15.117  (mysql-5.6.36 ndb-7.4.15)

[mysqld(API)]   3 node(s)
id=5 (not connected, accepting connect from 172.31.2.100)
id=6 (not connected, accepting connect from 172.31.14.98)
id=7 (not connected, accepting connect from 172.31.9.185)

ubuntu@ip-172-31-15-117:/var/lib/mysql-cluster$
```

Figure 10: Showing connected SQL/Data Nodes

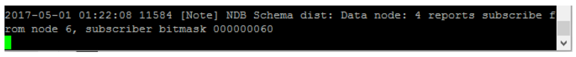
For now, we notice that SQL nodes are not connected yet. To connect them, we use the command :
`/usr/local/mysql/bin/mysqld -user=ubuntu &`



```
2017-05-01 01:21:33 11584 [Note] NDB: Cleaning stray tables from database 'ndbinfo
'
2017-05-01 01:21:33 11584 [Note] NDB: Cleaning stray tables from database 'perform
ance_schema'
2017-05-01 01:21:33 11584 [Note] NDB: Cleaning stray tables from database 'test'
2017-05-01 01:21:33 11584 [Note] NDB: missing frm for mysql.ndb_index_stat_sample,
 discovering...
2017-05-01 01:21:33 11584 [Note] NDB: missing frm for mysql.ndb_index_stat_head, d
iscovering...
2017-05-01 01:21:33 11584 [Note] NDB Binlog: Wait for first event
2017-05-01 01:21:33 [NdbApi] INFO     -- Flushing incomplete GCI:s < 106/16
2017-05-01 01:21:33 [NdbApi] INFO     -- Flushing incomplete GCI:s < 106/16
2017-05-01 01:21:33 11584 [Note] NDB Binlog: starting log at epoch 106/16
2017-05-01 01:21:33 11584 [Note] NDB Binlog: Got first event
2017-05-01 01:21:33 11584 [Note] NDB Binlog: ndb tables writable
2017-05-01 01:21:33 11584 [Note] NDB Binlog: Startup and setup completed
```

Figure 11: Mysql Cluster set up and start up completed

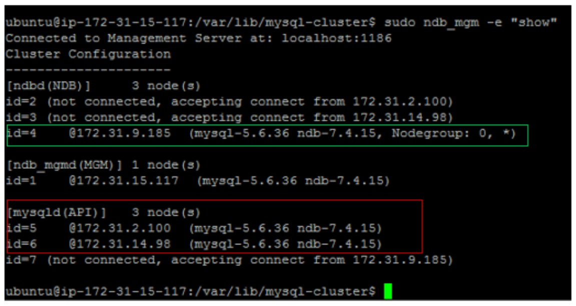
When another node get connected, we can see the occurring event appear on other cluster members.



```
2017-05-01 01:22:08 11584 [Note] NDB Schema dist: Data node: 4 reports subscribe f
rom node 6, subscriber bitmask 000000060
```

Figure 12: New cluster member appear on co-members

The `ndb_mgm show` command let us know that we have two SQL nodes connected to the cluster and one data node. We wanted to have all data nodes and SQL nodes connected at the same time but we have tried with many tutorials and documents, by the end we could only get some connected as SQL Nodes and others as Data Nodes, so we decide to move forward with it.

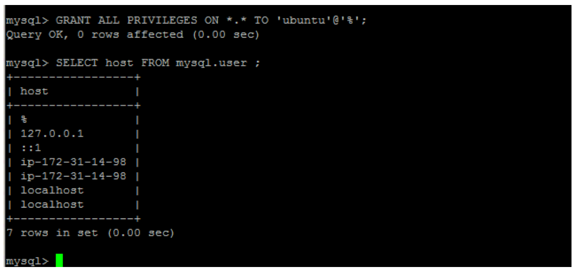


```
ubuntu@ip-172-31-15-117:/var/lib/mysql-cluster$ sudo ndb_mgm -e "show"
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     3 node(s)
id=2 (not connected, accepting connect from 172.31.2.100)
id=3 (not connected, accepting connect from 172.31.14.98)
id=4    @172.31.9.185  (mysql-5.6.36 ndb-7.4.15, Nodegroup: 0, *)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @172.31.15.117  (mysql-5.6.36 ndb-7.4.15)

[mysqld(API)]   3 node(s)
id=5    @172.31.2.100  (mysql-5.6.36 ndb-7.4.15)
id=6    @172.31.14.98  (mysql-5.6.36 ndb-7.4.15)
id=7 (not connected, accepting connect from 172.31.9.185)

ubuntu@ip-172-31-15-117:/var/lib/mysql-cluster$
```

Figure 13: Connected SQL and Data nodes

3.1 Testing MYSQL Cluster

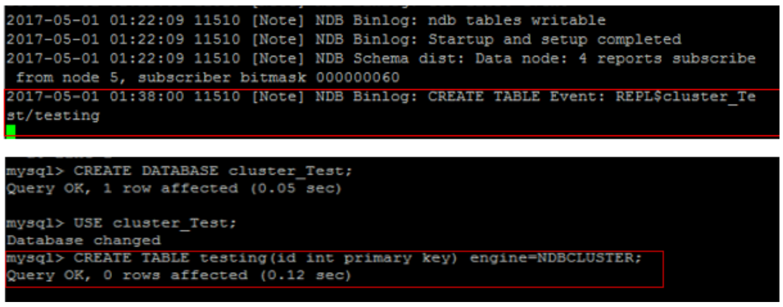
To test the cluster, we will need to connect to it remotely and create a test table with ndbcluster engine. Ndbcluster is the distributed database system underlying MySQL cluster. The first thing to do is to grant remote access permissions to MySQL server as shown in the picture below :



```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'ubuntu'@'%';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT host FROM mysql.user ;
+-----------------+
| host            |
+-----------------+
| %               |
| 127.0.0.1       |
| ::1             |
| ip-172-31-14-98 |
| ip-172-31-14-98 |
| localhost       |
| localhost       |
+-----------------+
7 rows in set (0.00 sec)

mysql>
```

Figure 14: Grant permissions for Mysql server

After granting permissions, we can proceed with some tests on the cluster to see the effects or usual SQL commands on a MySQL NDB cluster. As shown below, the create table event appear

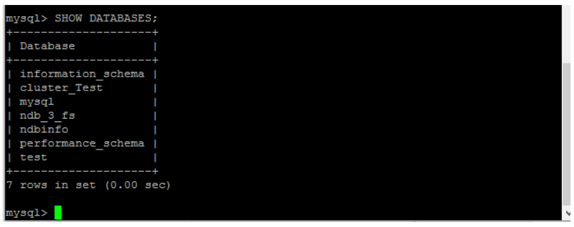


```
2017-05-01 01:22:09 11510 [Note] NDB Binlog: ndb tables writable
2017-05-01 01:22:09 11510 [Note] NDB Binlog: Startup and setup completed
2017-05-01 01:22:09 11510 [Note] NDB Schema dist: Data node: 4 reports subscribe
 from node 5, subscriber bitmask 000000060
2017-05-01 01:38:00 11510 [Note] NDB Binlog: CREATE TABLE Event: REPL$cluster_Te
st/testing
```

```
mysql> CREATE DATABASE cluster_Test;
Query OK, 1 row affected (0.05 sec)

mysql> USE cluster_Test;
Database changed
mysql> CREATE TABLE testing(id int primary key) engine=NDBCLUSTER;
Query OK, 0 rows affected (0.12 sec)
```

Figure 15: Creating Table Event on a Mysql Cluster

on all members of the SQL cluster. The existing default databases on each SQL nodes are :



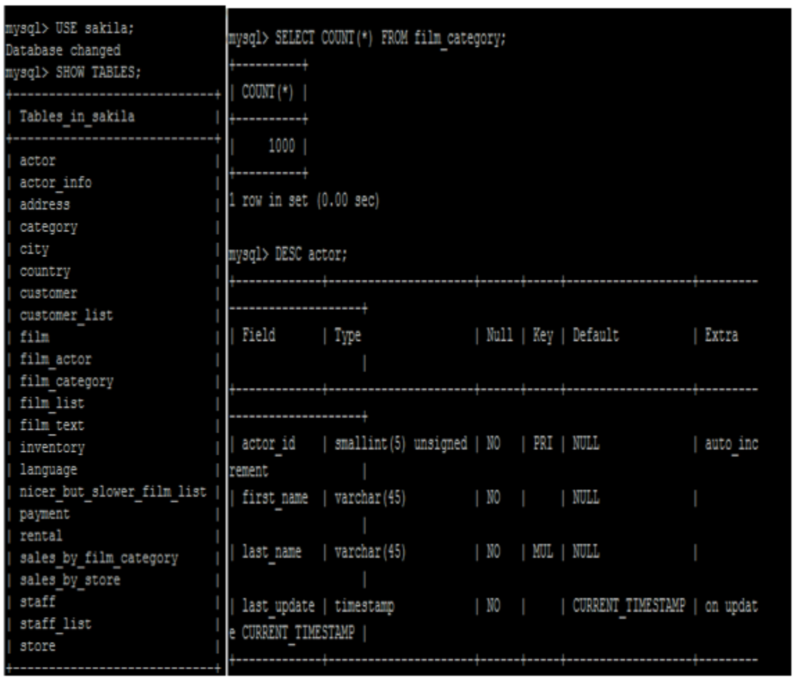
```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cluster_Test       |
| mysql              |
| ndb_3_fs           |
| ndbinfo            |
| performance_schema |
| test               |
+--------------------+
7 rows in set (0.00 sec)

mysql>
```

Figure 16: Default databases on SQL nodes

3.2 Sakila

The Sakila sample database has been developed to provide a standard schema that can be used for examples in books, tutorials, articles, samples, and so forth. Sakila sample database also serves to highlight the latest features of MySQL such as Views, Stored Procedures, and Triggers.
We have download Sakila from http://dev.mysql.com/doc/index-other.html which is containing three files: sakila-schema.sql, sakila-data.sql, and sakila.mwb. The *sakila-schema.sql* file contains all the CREATE statements required to create the structure of the Sakila database including tables, views, stored procedures, and triggers. The *sakila-data.sql* file contains the INSERT statements required to populate the structure created by the sakila-schema.sql file, along with definitions for triggers that must be created after the initial data load. The *sakila.mwb* file is a MySQL Workbench data model that we can open within MySQL Workbench to examine the database structure. To install the Sakila sample database (on a standalone MySQL), we have followed these steps:

- Downloading and extracting the installation archive to a location (such as /tmp/)
- Connect to the MySQL server using the command-line client with *mysql -u root -p*
- Executing the sakila-schema.sql script to create the database structure by using *mysql> SOURCE /tmp/sakila-db/sakila-schema.sql*
- Executing the sakila-data.sql script to populate the database structure with *mysql> SOURCE /tmp/sakila-db/sakila-data.sql;*
- Confirming that the sample database is installed correctly by executing the following statements. We are able to see output similar to that as shown below:



```
mysql> USE sakila;
Database changed
mysql> SHOW TABLES;
+----------------------------+
| Tables_in_sakila           |
+----------------------------+
| actor                      |
| actor_info                 |
| address                    |
| category                   |
| city                       |
| country                    |
| customer                   |
| customer_list              |
| film                       |
| film_actor                 |
| film_category              |
| film_list                  |
| film_text                  |
| inventory                  |
| language                   |
| nicer_but_slower_film_list |
| payment                    |
| rental                     |
| sales_by_film_category     |
| sales_by_store             |
| staff                      |
| staff_list                 |
| store                      |
+----------------------------+
```

```
mysql> SELECT COUNT(*) FROM film_category;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)

mysql> DESC actor;
+-------------+----------------------+------+-----+-------------------+---------
-------------------+
| Field       | Type                 | Null | Key | Default           | Extra
                   |
+-------------+----------------------+------+-----+-------------------+---------
-------------------+
| actor_id    | smallint(5) unsigned | NO   | PRI | NULL              | auto_inc
rement             |
| first_name  | varchar(45)          | NO   |     | NULL              |
                   |
| last_name   | varchar(45)          | NO   | MUL | NULL              |
                   |
| last_update | timestamp            | NO   |     | CURRENT_TIMESTAMP | on updat
e CURRENT_TIMESTAMP |
+-------------+----------------------+------+-----+-------------------+---------
```

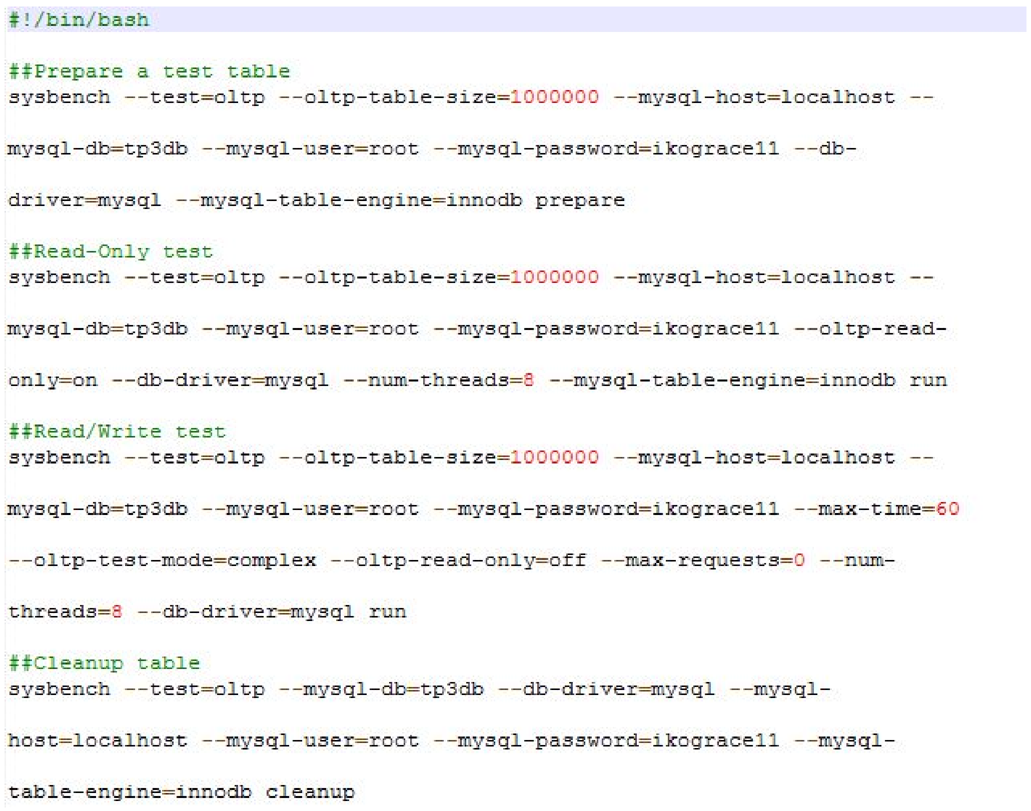
Figure 17: Default databases on SQL nodes

4 Benchmarking MYSQL Standalone against MYSQL CLuster

After installation of MySQL, we will configure MySQL and mount the database Sakila. In this section we will need to benchmark these installations by using Sysbench.
Sysbench is a popular open source benchmark to test open source Data Base Management Servers. It is a Benchmarking utilities that can be used to test out Physical and Cloud Server performance. Sysbench t allows us to test out performance of the following system characteristics:

1. CPU performance
2. File I/O performance
3. Memory allocation and transfer speed
4. Database server performance.

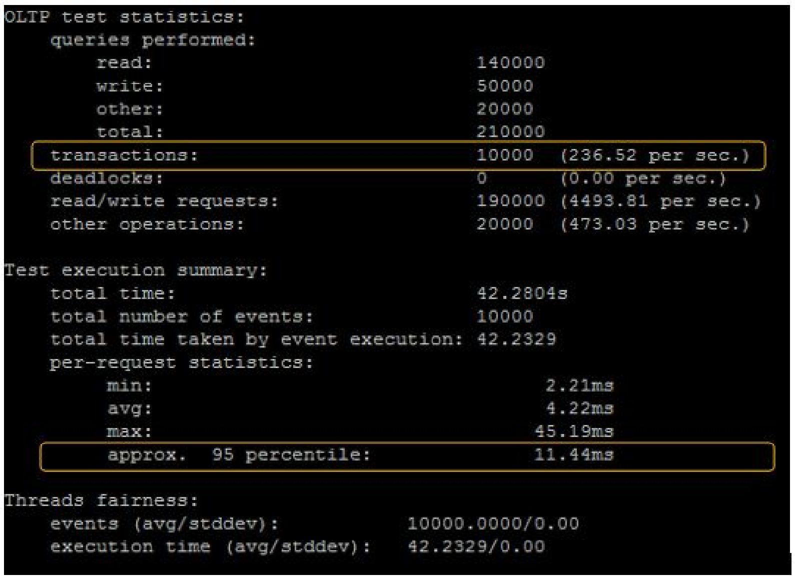
We have used the below script to run the test locally.



```
#!/bin/bash

##Prepare a test table
sysbench --test=oltp --oltp-table-size=1000000 --mysql-host=localhost --

mysql-db=tp3db --mysql-user=root --mysql-password=ikograce11 --db-

driver=mysql --mysql-table-engine=innodb prepare

##Read-Only test
sysbench --test=oltp --oltp-table-size=1000000 --mysql-host=localhost --

mysql-db=tp3db --mysql-user=root --mysql-password=ikograce11 --oltp-read-

only=on --db-driver=mysql --num-threads=8 --mysql-table-engine=innodb run

##Read/Write test
sysbench --test=oltp --oltp-table-size=1000000 --mysql-host=localhost --

mysql-db=tp3db --mysql-user=root --mysql-password=ikograce11 --max-time=60

--oltp-test-mode=complex --oltp-read-only=off --max-requests=0 --num-

threads=8 --db-driver=mysql run

##Cleanup table
sysbench --test=oltp --mysql-db=tp3db --db-driver=mysql --mysql-

host=localhost --mysql-user=root --mysql-password=ikograce11 --mysql-

table-engine=innodb cleanup
```

Figure 18: Script to run sysbench locally

Sysbench runs a specified number of threads and they all execute requests in parallel. We have used OLTP modes in Sysbench to evaluate MySQL server performance in standalone mode.



```
OLTP test statistics:
    queries performed:
        read:                            140000
        write:                           50000
        other:                           20000
        total:                           210000
    transactions:                        10000  (236.52 per sec.)
    deadlocks:                           0      (0.00 per sec.)
    read/write requests:                 190000 (4493.81 per sec.)
    other operations:                    20000  (473.03 per sec.)

Test execution summary:
    total time:                          42.2804s
    total number of events:              10000
    total time taken by event execution: 42.2329
    per-request statistics:
         min:                                  2.21ms
         avg:                                  4.22ms
         max:                                 45.19ms
         approx.  95 percentile:              11.44ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   42.2329/0.00
```

Figure 19: Result of sysbench on a standalone MySQL

The result shows the number of transactions the test managed to complete and the time it tooks (per second). We could also consider "95 percentile" as another factor to evaluate execution time per request.

4.1 Comparison

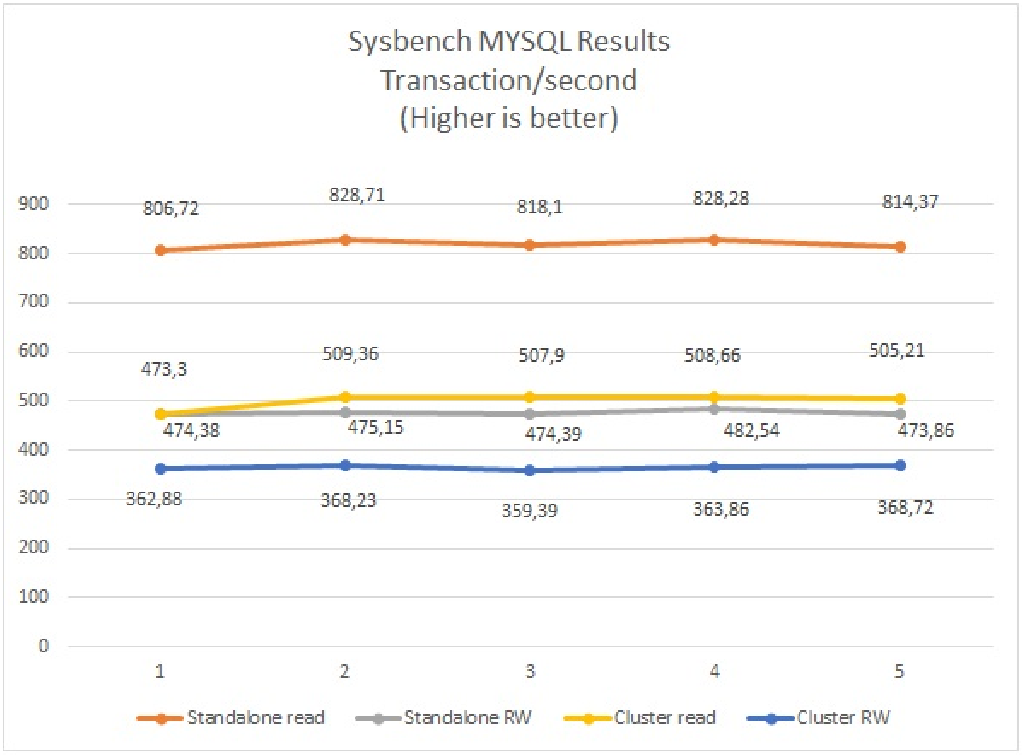


Figure 20: MySQL standalone and cluster performance

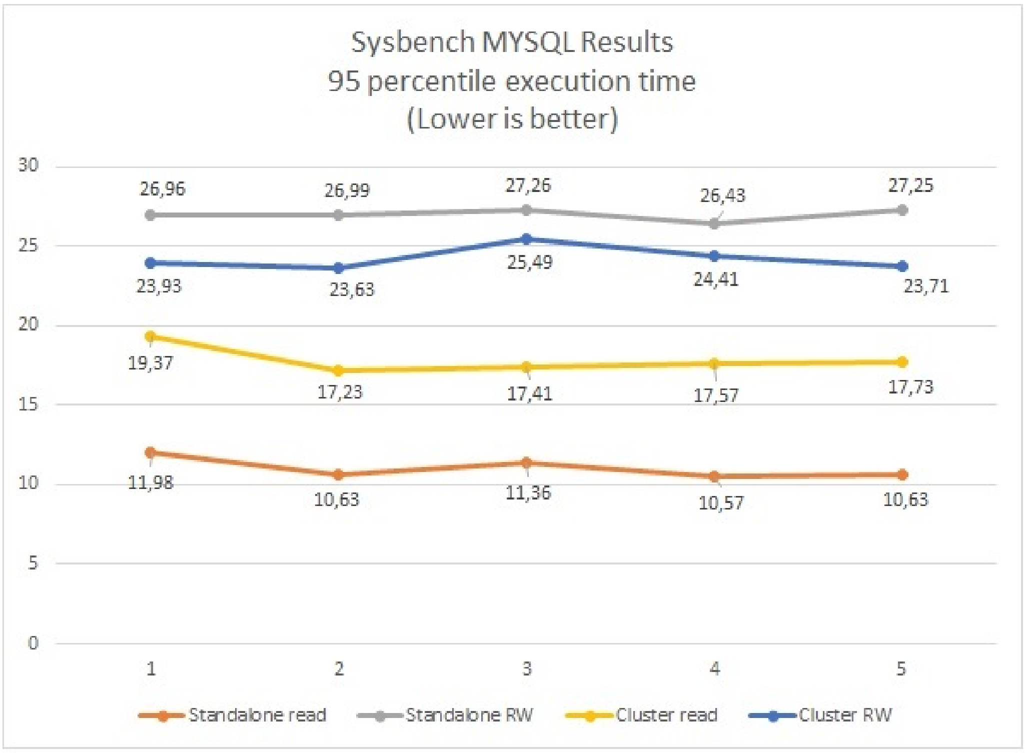
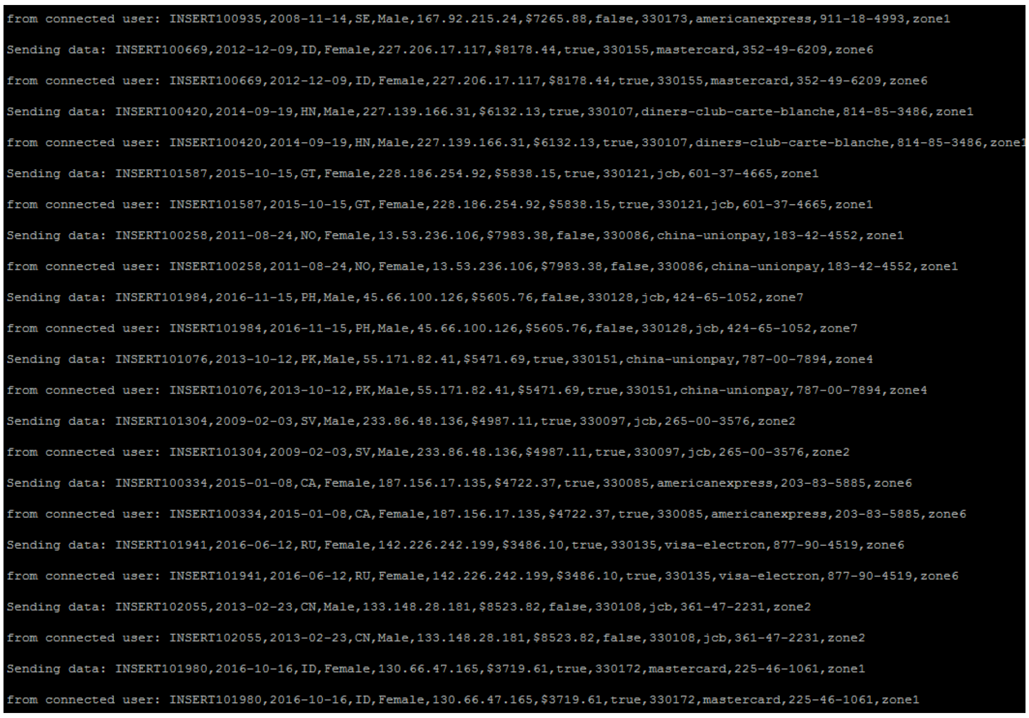


Figure 21: MySQL standalone and cluster performance

The results show that the performance of the standalone instance is higher than clustered setup. The standalone setup has 1.6 times better Transaction/seconde in comparison with clustered setup and also it has 1.6 times faster query execution time. This is because we have executed our test on a small range of records as expected for a real distributed test. MySQL cluster is designed to have more performance when facing with large amount of records.

5 Implementing Cloud Patterns

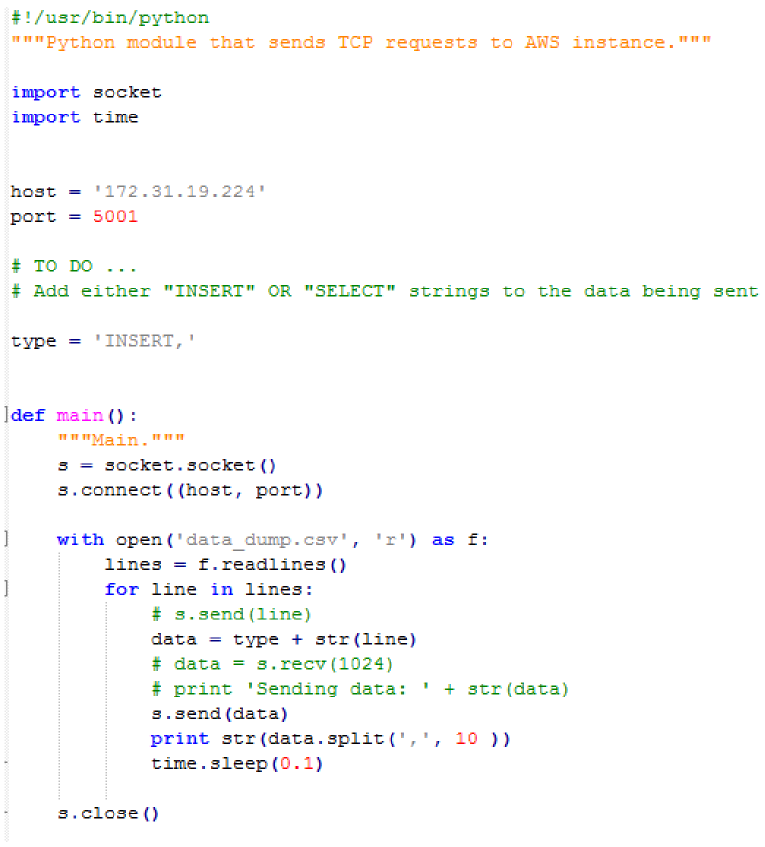
To implement cloud patterns, we used the python code gaved by the author of the lab and we made an architecture with the 2 previous SQL nodes and a t2.large instance to be the proxy node. We used the data set from the previous assignment to make the tests. At first we tested the the codes for the receiver and the sender locally and ensuring to start the receiver first for it to listen to the sender's requests.



```
from connected user: INSERT100935,2008-11-14,SE,Male,167.92.215.24,$7265.88,false,330173,americanexpress,911-18-4993,zone1

Sending data: INSERT100669,2012-12-09,ID,Female,227.206.17.117,$8178.44,true,330155,mastercard,352-49-6209,zone6

from connected user: INSERT100669,2012-12-09,ID,Female,227.206.17.117,$8178.44,true,330155,mastercard,352-49-6209,zone6

Sending data: INSERT100420,2014-09-19,HN,Male,227.139.166.31,$6132.13,true,330107,diners-club-carte-blanche,814-85-3486,zone1

from connected user: INSERT100420,2014-09-19,HN,Male,227.139.166.31,$6132.13,true,330107,diners-club-carte-blanche,814-85-3486,zone1

Sending data: INSERT101587,2015-10-15,GT,Female,228.186.254.92,$5838.15,true,330121,jcb,601-37-4665,zone1

from connected user: INSERT101587,2015-10-15,GT,Female,228.186.254.92,$5838.15,true,330121,jcb,601-37-4665,zone1

Sending data: INSERT100258,2011-08-24,NO,Female,13.53.236.106,$7983.38,false,330086,china-unionpay,183-42-4552,zone1

from connected user: INSERT100258,2011-08-24,NO,Female,13.53.236.106,$7983.38,false,330086,china-unionpay,183-42-4552,zone1

Sending data: INSERT101984,2016-11-15,PH,Male,45.66.100.126,$5605.76,false,330128,jcb,424-65-1052,zone7

from connected user: INSERT101984,2016-11-15,PH,Male,45.66.100.126,$5605.76,false,330128,jcb,424-65-1052,zone7

Sending data: INSERT101076,2013-10-12,PK,Male,55.171.82.41,$5471.69,true,330151,china-unionpay,787-00-7894,zone4

from connected user: INSERT101076,2013-10-12,PK,Male,55.171.82.41,$5471.69,true,330151,china-unionpay,787-00-7894,zone4

Sending data: INSERT101304,2009-02-03,SV,Male,233.86.48.136,$4987.11,true,330097,jcb,265-00-3576,zone2

from connected user: INSERT101304,2009-02-03,SV,Male,233.86.48.136,$4987.11,true,330097,jcb,265-00-3576,zone2

Sending data: INSERT100334,2015-01-08,CA,Female,187.156.17.135,$4722.37,true,330085,americanexpress,203-83-5885,zone6

from connected user: INSERT100334,2015-01-08,CA,Female,187.156.17.135,$4722.37,true,330085,americanexpress,203-83-5885,zone6

Sending data: INSERT101941,2016-06-12,RU,Female,142.226.242.199,$3486.10,true,330135,visa-electron,877-90-4519,zone6

from connected user: INSERT101941,2016-06-12,RU,Female,142.226.242.199,$3486.10,true,330135,visa-electron,877-90-4519,zone6

Sending data: INSERT102055,2013-02-23,CN,Male,133.148.28.181,$8523.82,false,330108,jcb,361-47-2231,zone2

from connected user: INSERT102055,2013-02-23,CN,Male,133.148.28.181,$8523.82,false,330108,jcb,361-47-2231,zone2

Sending data: INSERT101980,2016-10-16,ID,Female,130.66.47.165,$3719.61,true,330172,mastercard,225-46-1061,zone1

from connected user: INSERT101980,2016-10-16,ID,Female,130.66.47.165,$3719.61,true,330172,mastercard,225-46-1061,zone1
```

Figure 22: Execution of TCP connections on a local host

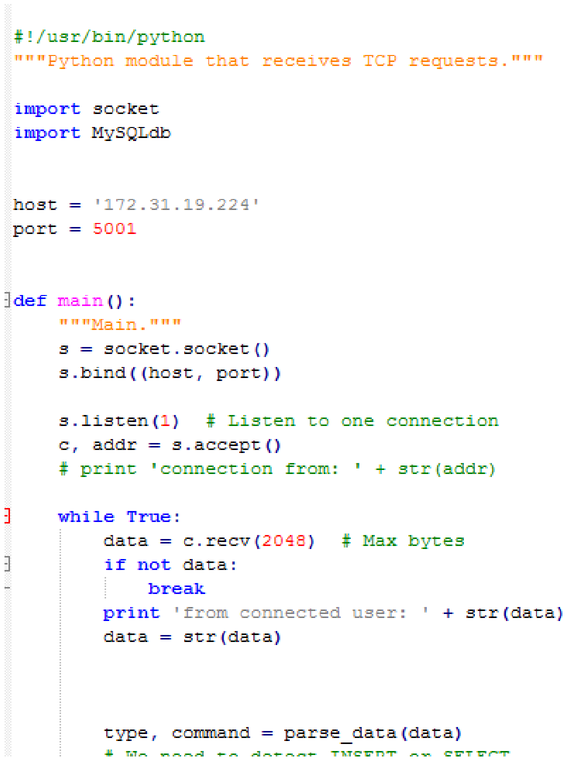
After ensuring the connections would work on a clustered setup, we changed the default localhost values to the proxy's node in both the sender and receiver applications.



```python
#!/usr/bin/python
"""Python module that sends TCP requests to AWS instance."""

import socket
import time


host = '172.31.19.224'
port = 5001

# TO DO ...
# Add either "INSERT" OR "SELECT" strings to the data being sent

type = 'INSERT,'


def main():
    """Main."""
    s = socket.socket()
    s.connect((host, port))

    with open('data_dump.csv', 'r') as f:
        lines = f.readlines()
        for line in lines:
            # s.send(line)
            data = type + str(line)
            # data = s.recv(1024)
            # print 'Sending data: ' + str(data)
            s.send(data)
            print str(data.split(',', 10 ))
            time.sleep(0.1)

    s.close()
```

Figure 23: Proxy Pattern Sender Code

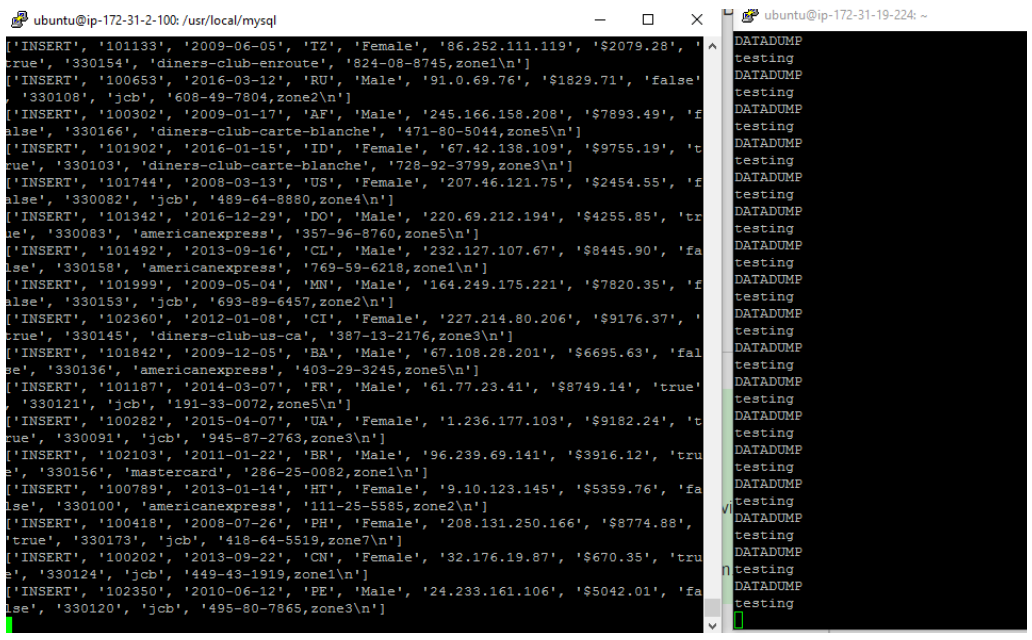
The sender is a random node we choosed among the different cluster members. In this case, it was one of the SQL node. The receiver is of course the Proxy node.



```python
#!/usr/bin/python
"""Python module that receives TCP requests."""

import socket
import MySQLdb


host = '172.31.19.224'
port = 5001


def main():
    """Main."""
    s = socket.socket()
    s.bind((host, port))

    s.listen(1)  # Listen to one connection
    c, addr = s.accept()
    # print 'connection from: ' + str(addr)

    while True:
        data = c.recv(2048)  # Max bytes
        if not data:
            break
        print 'from connected user: ' + str(data)
        data = str(data)



        type, command = parse_data(data)
        # We need to detect INSERT or SELECT
```

Figure 24: Proxy Pattern Receiver Code

The Proxy pattern act like a forwarder, in case of write requests, for example INSERT, it asked to the master node to take care of it, but when it is only about reading, the slaves can be just fine. So to implement this pattern, as in our dataset, most values are separated by commas, we added to each command value a comma and then we used a map function to match each data (to insert or select) with the function attended to.

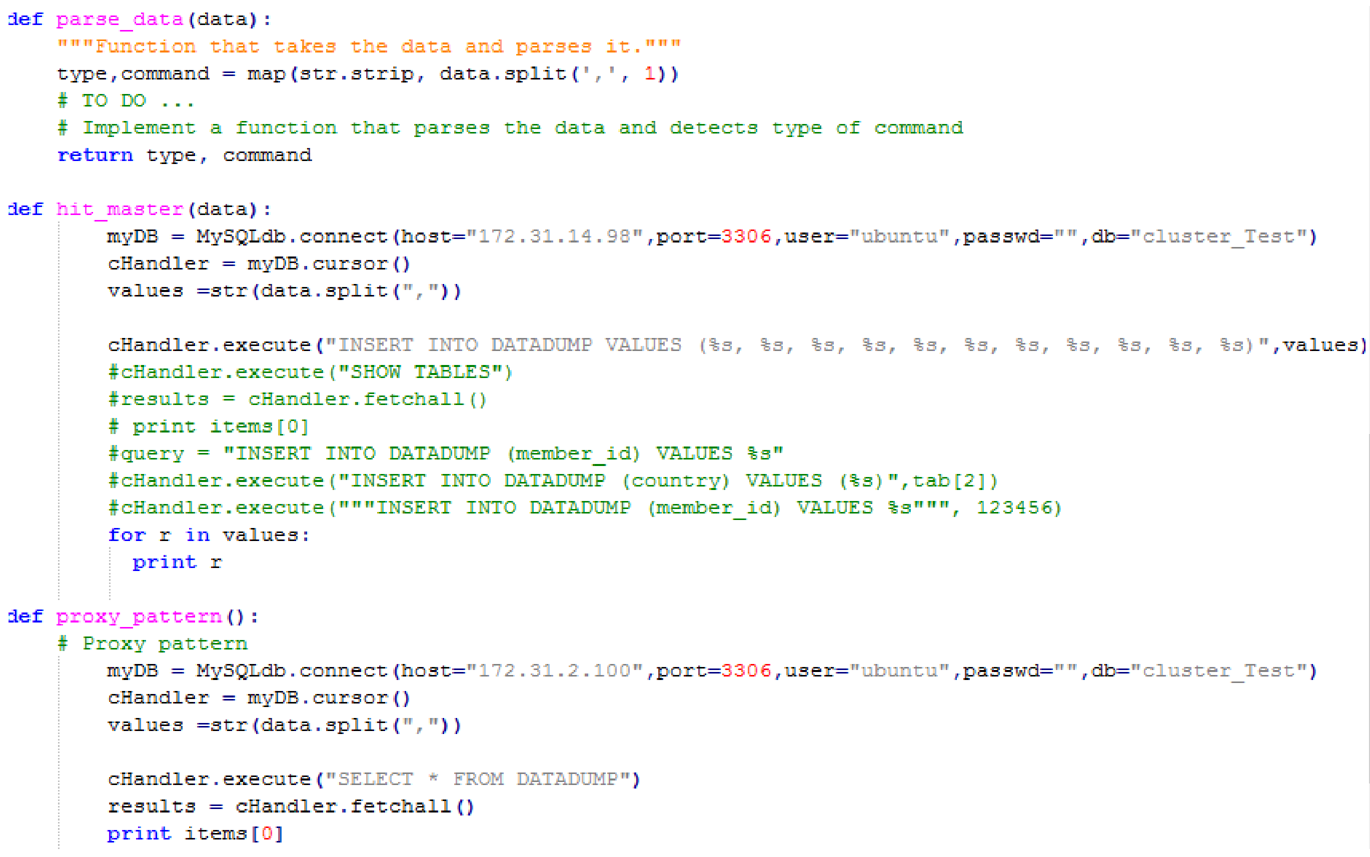


ubuntu@ip-172-31-2-100: /usr/local/mysql
ubuntu@ip-172-31-19-224: ~

```
['INSERT', '101133', '2009-06-05', 'TZ', 'Female', '86.252.111.119', '$2079.28', '
true', '330154', 'diners-club-enroute', '824-08-8745,zone1\n']
['INSERT', '100653', '2016-03-12', 'RU', 'Male', '91.0.69.76', '$1829.71', 'false'
, '330108', 'jcb', '608-49-7804,zone2\n']
['INSERT', '100302', '2009-01-17', 'AF', 'Male', '245.166.158.208', '$7893.49', 'f
alse', '330166', 'diners-club-carte-blanche', '471-80-5044,zone5\n']
['INSERT', '101902', '2016-01-15', 'ID', 'Female', '67.42.138.109', '$9755.19', 't
rue', '330103', 'diners-club-carte-blanche', '728-92-3799,zone3\n']
['INSERT', '101744', '2008-03-13', 'US', 'Female', '207.46.121.75', '$2454.55', 'f
alse', '330082', 'jcb', '489-64-8880,zone4\n']
['INSERT', '101342', '2016-12-29', 'DO', 'Male', '220.69.212.194', '$4255.85', 'tr
ue', '330083', 'americanexpress', '357-96-8760,zone5\n']
['INSERT', '101492', '2013-09-16', 'CL', 'Male', '232.127.107.67', '$8445.90', 'fa
lse', '330158', 'americanexpress', '769-59-6218,zone1\n']
['INSERT', '101999', '2009-05-04', 'MN', 'Male', '164.249.175.221', '$7820.35', 'f
alse', '330153', 'jcb', '693-89-6457,zone2\n']
['INSERT', '102360', '2012-01-08', 'CI', 'Female', '227.214.80.206', '$9176.37', '
true', '330145', 'diners-club-us-ca', '387-13-2176,zone3\n']
['INSERT', '101842', '2009-12-05', 'BA', 'Male', '67.108.28.201', '$6695.63', 'fal
se', '330136', 'americanexpress', '403-29-3245,zone5\n']
['INSERT', '101187', '2014-03-07', 'FR', 'Male', '61.77.23.41', '$8749.14', 'true'
, '330121', 'jcb', '191-33-0072,zone5\n']
['INSERT', '100282', '2015-04-07', 'UA', 'Female', '1.236.177.103', '$9182.24', 't
rue', '330091', 'jcb', '945-87-2763,zone3\n']
['INSERT', '102103', '2011-01-22', 'BR', 'Male', '96.239.69.141', '$3916.12', 'tru
e', '330156', 'mastercard', '286-25-0082,zone1\n']
['INSERT', '100789', '2013-01-14', 'HT', 'Female', '9.10.123.145', '$5359.76', 'fa
lse', '330100', 'americanexpress', '111-25-5585,zone2\n']
['INSERT', '100418', '2008-07-26', 'PH', 'Female', '208.131.250.166', '$8774.88',
'true', '330173', 'jcb', '418-64-5519,zone7\n']
['INSERT', '100202', '2013-09-22', 'CN', 'Female', '32.176.19.87', '$670.35', 'tru
e', '330124', 'jcb', '449-43-1919,zone1\n']
['INSERT', '102350', '2010-06-12', 'PE', 'Male', '24.233.161.106', '$5042.01', 'fa
lse', '330120', 'jcb', '495-80-7865,zone3\n']
```

```
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
DATADUMP
testing
```

Figure 25: Example executing Show table via Proxy pattern



```
def parse_data(data):
    """Function that takes the data and parses it."""
    type,command = map(str.strip, data.split(',', 1))
    # TO DO ...
    # Implement a function that parses the data and detects type of command
    return type, command

def hit_master(data):
        myDB = MySQLdb.connect(host="172.31.14.98",port=3306,user="ubuntu",passwd="",db="cluster_Test")
        cHandler = myDB.cursor()
        values =str(data.split(","))

        cHandler.execute("INSERT INTO DATADUMP VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)",values)
        #cHandler.execute("SHOW TABLES")
        #results = cHandler.fetchall()
        # print items[0]
        #query = "INSERT INTO DATADUMP (member_id) VALUES %s"
        #cHandler.execute("INSERT INTO DATADUMP (country) VALUES (%s)",tab[2])
        #cHandler.execute("""INSERT INTO DATADUMP (member_id) VALUES %s""", 123456)
        for r in values:
          print r

def proxy_pattern():
    # Proxy pattern
        myDB = MySQLdb.connect(host="172.31.2.100",port=3306,user="ubuntu",passwd="",db="cluster_Test")
        cHandler = myDB.cursor()
        values =str(data.split(","))

        cHandler.execute("SELECT * FROM DATADUMP")
        results = cHandler.fetchall()
        print items[0]
```

Figure 26: Proxy Pattern Implementation Code

References

F. Khomh, S. Amirhossein Abtahizadeh.
LOG 8415 : Lab 3 Scaling Databases and Implementing Cloud Patterns.
Département Génie Informatique et Génie Logiciel,
École Polytechnique de Montréal, Québec, Canada

Installing mysql.
https://www.linode.com/docs/databases/mysql/ install-mysql-on-ubuntu-14-04/
https://dev.mysql.com/doc/sakila/en/
https://dev.mysql.com/doc/sakila/en/sakila-installation.html

Tutorial MYSQL Cluster
https://www.youtube.com/watch?v=1SlxZqhth3w&t=822s
http://dev.mysql.com/doc/refman/5.5/en/mysql-cluster-programs-ndb-mgm.html
https://doc.ubuntu-fr.org/mysql

Benchmarking
http://www.mysql.com/why-mysql/benchmarks/mysql-cluster/
https://www.stephenrlang.com/2016/02/benchmark-mysql-with-sysbench/