Lab No. 6: Relational and Non-Relational Databases

In this lab we are going to compare the performance of two database systems. We are going to use an updated version of the minibank application that stores session data in a Relational Database instead of in memory. In previous labs, logging with session cookies was not scalable because session data was stored locally on each minibank process. Now, session data is stored in a database, so it is accessible to every node, which means that we can safely scale minibank horizontally.

Tips mentioned in class:

1. You will need to install python2.7 and virtualenv in `lab06`

```
apt-get update
apt-get install python2.7 virtualenv
```

2. You need to update the Makefile to force it to use python2.7. Make line 38 on the Makefile look like this (make sure you preserve the tab space in front of the command):

```
virtualenv venv --python=python2.7
```

3. If after entering one of the mysql containers, you get stuck and can't get out (that means, you can't type the `exit` command to exit out of the container), you can terminate the interactive session with the container by looking for the `kubectl` process and killing it.

```
ps fax | grep kubectl
```

look at the output of the previous command, determine the process id that you want to terminate and then use the following command to kill it (replace PID with the process id):

```
kill -9 PID
```

4. To delete Kubernetes resources:

```
kubectl delete TYPE RESOURCENAME
```

TYPE is one of `service`, `deployment`, `pod`, etc and *RESOURCENAME* is the name of the resource that you want to delete. Deleting a deployment deletes its associated pods.

Part 1: Setup a Kubernetes Cluster

1. Provision a Kubernetes cluster by following the steps from Part 1 of Lab No. 3. (../lab03/lab03.html#lab03-part1). Name the virtual machine **lab06** and name your Kubernetes Cluster **lab06cluster** Make sure that you test the kubernetes cluster by deploying a test pod. (Any image is fine, as long as it lets you confirm that the Kubernetes cluster is running.
2. Install `make` and `ab` on **labo6**:

```
> sudo apt-get update
> sudo apt-get install -y make apache2-utils build-essential
```

Part 2: Deployment of a SQL database

In this part of the lab, you are going to deploy a 1 node Maria DB, deploy minibank, and then run an AB test to measure performance of both read and write. Download an **updated** version of minibank from https://github.com/jcabmora/minibank/archive/week9.zip (https://github.com/jcabmora/minibank/archive/week9.zip)

1. Build and test the images locally (on the **lab06** VM)

```
> make run-images
```

2. Confirm that all the API endpoints that are listed in Part 2 of Lab No. 5. (../lab05/lab05.html#lab05-part2) work.
3. Follow the procedure of Part 3 of Lab No. 5. (../lab05/lab05.html#lab05-part3) to deploy mysql and minibank to the Kubernetes cluster you provisioned at the beginning of this Lab.

#. We are going to test the performance of the current minibank/mariadb deployments. For this lab, there is tooling added in the `load_test` directory. There is a `make` target that has been added to simplify calling this script. When executing this target, you can use the `REPLICAS` argument to provide a space separated list of replicas for the minibank app and also use `TAG` to specify names of result files:

```
> make load-test REPLICAS='1 3' TAG=mysql
```

This command will take a few minutes. It will scale the `minibank` deployment to first run with 1 replica, run a series of AB tests and collect the results, and then will scale to 3 replicas and repeat. You can open the `load_test/collect.py` to review the script. The make file calls this script and specifies to use the `api/account/login` endpoint. Since this endpoint **writes** to the database, this is a **WRITE** performance test.

Part 3: Deployment of a clustered SQL database

In this part of the lab, you are going to deploy a 3 node MariaDB cluster, deploy minibank and then run an AB test to measure performance of both read and writes. MariaDB can be run in a multi-master synchronous replication mode using Galera. Setting up a MariaDB cluster is a delicate process (details are described here: http://galeracluster.com/documentation-webpages/startingcluster.html (http://galeracluster.com/documentation-webpages/startingcluster.html)) that requires the first node to start the cluster, and then subsequent nodes to be started one by one.

1. Remove the minibank service, minibank deployment, the mysql service and the mysql deployment
2. Deploy again the mysql service, but this time use the `kubernetes/galera.yaml` resource file (do not forget to **update your project id**)

   ```
   > kubectl create –f kubernetes/galera.yaml
   ```

   The previous command will start three pods. However, these are not running the `mysqld` daemon, and we have to manually bootstrap the cluster. In future labs you will automate this process with the help of Zookeeper).

3. We need to start the cluster one node at a time. Run the following command to get a list of your pods:

   ```
   > kubectl get pods
   NAME                     READY     STATUS    RESTARTS   AGE
   mysql–664d7f45f7–b42b4   1/1       Running   0          22m
   mysql–664d7f45f7–lt5kp   1/1       Running   0          22m
   mysql–664d7f45f7–wmqdq   1/1       Running   0          22m
   ```

4. The first pod will *bootstrap* the galera cluster. Pick one of the pods, and start an interactive session into it (In the following example pod `mysql-664d7f45f7-b42b4` was selected):

   ```
   > kubectl exec –it mysql–664d7f45f7–b42b4 bash
   root@mysql–664d7f45f7–b42b4:/#
   ```

5. Once you have an interactive session with the first pod, we are going to start the `mysqld` daemon with the following options:

- an option to enable mysql to be run as the mysql user (`--user=mysql`)
- an option to enable synchronous replication (`--wsrep_on=ON`)
- an option to bootstrap a new galera cluster (`--wsrep-new-cluster`)
- an option to assign a server id of 1 (`--server-id=1`)
- an option to name the cluster (`--wsrep-cluster-name=lab06`)
- an option to enable the appropriate binary log format required by galera (`--binlog_format=ROW`)
- an option to specify the replication domain id (`--wsrep_gtid_domain_id=1`)
- an option to specify the protocol used to perform state transfers (`--wsrep_sst_method=rsync`)
- an option to specify the cluster address (`--wsrep_cluster_address=gcom://`)
- an option to use galera as the replication library (`--wsrep_provider=/usr/lib/galera/libgalera_smm.so`).

Notice that in the following example the pod prompt is included, as a way to reiterate the fact that this command should be executed **in the pod**. Once you run this command, stdout will be written to console immediately, and the shell prompt **will not** be displayed immediately, you need to press `ENTER`. Also, notice that we use the `&` at the end of the command to run it as a background process:

```
root@mysql-664d7f45f7-b42b4:/#  mysqld --user=mysql --wsrep-cluster-name=lab06
```

6. To confirm that the `mysqld` daemon is running and configured properly to use galera replication, we can verify that the status variable `wsrep_cluster_status` is equal to `Primary`:

```
root@mysql-664d7f45f7-b42b4:/# mysql -uroot -phobbes -e 'SHOW STATUS LIKE "wsre
+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| wsrep_cluster_status | Primary |
+----------------------+---------+
```

7. Find out the ipaddress of the pod, since we are going to need it for other nodes to join the cluster:

```
root@mysql-664d7f45f7-b42b4:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defau
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if40: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue state UP
    link/ether 0a:58:0a:3c:00:25 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.60.0.37/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::24c5:65ff:fe6e:b4cc/64 scope link
       valid_lft forever preferred_lft forever
```

8. Exit out of the first pod. Select the second pod and start an interactive session. Execute the following command to start the `mysqld` and have it join the galera cluster that we started previously. Notice that we removed the `--wsrep-new-cluster` flag, that we updated the `--server-id=2` argument and also that **you need to update** the `--wsrep_cluster_address` parameter by replacing `XXX.XXX.XXX.XXX` with the IP Address that you determined on the previous step.

```
root@mysql-664d7f45f7-njgsd:/# mysqld --user=mysql --wsrep-cluster-name=lab06 -
```

9. To verify that all is OK, query again the value of the `wsrep_cluster_status` global variable on the second pod.
10. Repeat the previous two steps on the third pod. Make sure that you update the server id parameter (to be `server-id=3`). For cluster address you can also use the address of the first node.
11. Deploy the minibank service:

```
> kubectl create -f kubernetes/minibank.yaml
> kubectl expose deployment minibank --port 8080 --type LoadBalancer
```

12. Wait until an External IP address is assigned to the minibank service.
13. Test the service:

```
> curl http://<MINIBANK_SERVICE_IP_ADDRESS>:8080/api/account/register -d '{"use
> curl -c cookies.txt http://<MINIBANK_SERVICE_IP_ADDRESS>:8080/api/account/log
```

14. Log into each one of the pods, start a mysql session, and compare the result of the following query on all the pods. Does this match your expectation?

```
> SELECT * FROM minibank.account;
> SELECT * FROM minibank.sessions;
```

15. Delete one row from the `minibank.sessions` table, and compare the effect on the other two nodes (you should see that the other nodes also reflect the deleted row).
16. Run the load test again, this will take several minutes. Use `TAG=galera`

Part 4: Add support to a NO-SQL database

The version of minibank that is provided for this lab has most of the work done so it can store sessions in cassandra. However, you need to update the sources to implement the logic to interact with the database backend. There are three placeholders in the `src/minibank/handlers/account.go` file wher you need to implement the additional logic:

- The function that inserts rows in the sessions table
- The function that looks up a single row in the sessions table
- The function that looks up a list of rows for a specific user in the sessions table.

The purpose of this exercise is to become aware of how NoSQL databases implement interfaces and behaviors that are very similar to SQL databases, but also have some key differences.

For this part you need to do your own research. There are many sites that you can use to learn about cassandra. For this lab you do not need to be an expert, but it will help to understand the basics. The material available at https://www.guru99.com/cassandra-tutorial.html (https://www.guru99.com/cassandra-tutorial.html) is very helpful, as well as the Cassandra project's *Getting Started* page (http://cassandra.apache.org/doc/latest/getting_started/index.html (http://cassandra.apache.org/doc/latest/getting_started/index.html))

To quickly get access to a running Cassandra instance, you can use the `run-images` make target. This will deploy a container running Cassandra. You can log into it and immediately start a `cqlsh` session.

Since we will be using the `gocql` library to interact with cassandra, the obvious starting point is its `README`: https://github.com/gocql/gocql (https://github.com/gocql/gocql) The Cassandra keyspace and the session table are created in the `InitCassandra` function that is in the `src/minibank/models/db.go` file. Please base the CQL statements in your logic on those definitions.

For this part of the lab, add the missing logic, and test it locally. To enable minibank to run and use a Cassandra backend, you need to make sure you run the minibank application with the environment variable `ENABLE_CASSANDRA=TRUE`.

Part 5: Deployment of a NO-SQL database

Once you have updated minibank, we are going to run it with a single cassandra node.

1. Remove all the deployments and services from kubernetes
2. Deploy cassandra:

```
> kubectl create –f kubernetes/cassandra.yaml
```

3. Open the `kubernetes/minibank-cass.yaml` file and update it with your project
4. Deploy minibank:

```
> kubectl create –f kubernetes/minibank–cass.yaml
```

5. Test that all the API endpoints work
6. Log into the cassandra container and verify that session data is being populated in the `minibank.sessions` cassandra table.
7. Run your load tests again. Use `TAG=cassandra`

What to turn in

1. The updated source code for `src/minibank/handlers/account.go`
2. The load test results.
3. A discussion of the load test results.

Back to top