Lab No. 5: Horizontal Scalability and State

In this lab we are going to evaluate the implications for scalabitly when applications store data locally.

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 **lab05** and name your Kubernetes Cluster **lab05cluster** 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. On lab05 install `make`:

```
> sudo apt-get install make
```

Part 2: Authorization APIs

For this part of the lab, you will download an updated version of minibank that implements two different authentication schemas: cookie-based and token-based.

1. Download the updated minibank source from https://github.com/jcabmora/minibank/tree/week6 (https://github.com/jcabmora/minibank/tree/week6)
2. Build and run the images locally on `lab05`:

```
> make run-images
```

3. Verify that there are two docker images running, `minibank` and `mysql`.
4. Test that the service works by registering a user:

```
> curl localhost/api/account/register -d '{"username": "john", "password": "jo
Successfully registered account
```

The new version of minibank exposes two new REST API endpoints:

- **/api/account/login**: authenticates username/password credentials. If the authentication succeeds, generates session data internally and returns an empty `200` response with a `sessionid` cookie. For example:

```
> curl -c cookies.txt localhost/api/account/login -d '{"username": "john", "pas
```

When the previous command executes, it should store cookie information in a file called `cookies.txt`. Once you open that file you should be able to confirm that a cookie was received.

```
> cat cookies.txt
# Netscape HTTP Cookie File
# https://curl.haxx.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.

localhost       FALSE   /api/account/   FALSE   0       sessionid       9de2b79
```

- **/api/account/token**: also authenticates username/password credentials. If the authentication succeeds, then it returns a JSON payload that contains a JSON Web Token that can be used for token authorization.

```
> curl -X POST localhost/api/account/token -d '{"username": "john", "password"
{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1Mzg4NDM2NzMsInVzZXJu
```

- **/api/account/sessions**: returns a list of the sessions that are associated with an authenticated user. Users need to provide either cookie or token proof of authentication. For example, to execute this using cookies:

```
> curl -b cookies.txt localhost/api/account/sessions
{"sessions":["9bf4a24c-71bc-42de-8432-68bfbdd941d9","9de2b793-c6aa-44f4-8e06-9b
```

Or if you prefer to use a token:

```
> curl  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAi
```

Part 3: Deploy to Kubernetes

1. We first need to push the images to the Google Container Registry.

```
> make push-images
```

2. Open the `kubernetes/mysql.yaml` file on a text editor and replace <YOUR_PROJECT_ID> with your google cloud project id.
3. Deploy the service:

```
> kubectl create -f kubernetes/mysql.yaml
```

4. Verify that the deployment is running
5. You can test that your mysql instance is running by executing this command:

```
> kubectl run -it --rm --image=mysql:5.6 --restart=Never mysql-client -- mysql
If you don't see a command prompt, try pressing enter.
mysql> exit
```

6. Open the `kubernetes/minibank.yaml` file on a text editor and replace <YOUR_PROJECT_ID> with your google cloud project id.
7. Deploy minibank:

```
> kubectl create -f kubernetes/minibank.yaml
```

8. Verify that both minibank and mysql pods are running
9. Create the minibank service:

```
> kubectl expose deployment minibank --port 8080 --type LoadBalancer
```

10. Wait until an External IP address is assigned to the minibank service.
11. If there is a `cookies.txt` file in your current working directory, remove it.
12. Once you have an External IP address assigned to the minibank service, test the service:

```
> curl http://<MINIBANK_SERVICE_IP_ADDRESS>:8080/api/account/register -d '{"username"
```

13. Test the `/api/account/login` endpoint:

```
> curl -c cookies.txt http://<MINIBANK_SERVICE_IP_ADDRESS>:8080/api/account/log
```

14. Test the `/api/account/sessions` endpoint (confirm taht the session in the response matches the `sessionid` stored in `cookies.txt`

```
>  curl -b cookies.txt http://<MINIBANK_SERVICE_IP_ADDRESS>:8080/api/account/se
{"sessions":["04ba9908-604b-4ff6-a5c0-12a547afe3fb"]}
```

15. Test the authentication with `/api/account/token` and use the obtained token to test `/api/account/sessions`

Part 4: Scale

1. Scale the minibank service to 5 replicas, and wait until all pods are runninig.
2. Make 5 more calls to the `/api/account/sessions` API using cookie base authorization. What do you observe?
3. Make 5 more call to the `/api/account/sessions` API using the token that was obtained earlier. What do you observe?

What to turn in

1. Answer to both questions from part 4, and provide an explanation for the observed behavior.
2. Which one of the authentication methods proves to be more scalable. What modifications can be done to the authentication method that is less scalable (or not scalable at all) in order to improve its scalability.

Back to top