FUSIONBANK

Naseem Brady, Kush Desai, Zahir Humpries, John Pluznyk

https://github.com/JohnPluznyk/Group7-CSC468CloudComputing

Project Overview

Fusion bank is a cloud-based open banking web application platform. Fusion Bank allows customers to connect several bank accounts under one account. Instead of having several accounts, Fusion Bank allows the consumer to unite all their bank accounts into one account. The goal of Fusion Bank is to try and mitigate the headache of having to manage several bank accounts at once.

Implementation

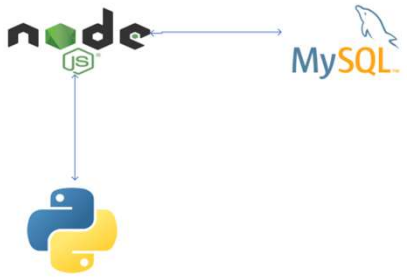
Web UI: Users will sign up and login taking them to a landing page. Client-side JavaScript allows users to connect Bank account.

Webserver: Node.JS servers acts as the backbone of our system it will serves the user their files.

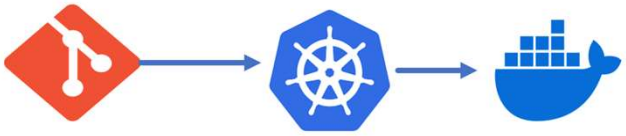
Database: MySQL will store the Fusion Bank user's login information.

Banking Application: Custom banking application is necessary as we do not have direct access any private banking API.

Architecture



CI/CD:



Docker:

Docker allows us to containerize our applications to create the necessary environments for our software to run.

Below is a docker compose file that automatically builds the necessary images needs for Fusion Bank.

Code Blame Raw

```
version: "3"

services:
  mysql:
    build: mysql
    image: 127.0.0.1:30000/mysql:v0.1
    ports:
    - "3306:3306"
    deploy:
      mode: global

  node:
    build: node
    image: 127.0.0.1:30000/node:v0.1
    ports:
    - "3000:3000"
    deploy:
      mode: global

  bankserver:
    build: bankserver
    image: 127.0.0.1:30000/bankserver:v0.1
    ports:
    - "3306:3306"
    deploy:
      mode: global
# CONTINUTE TO EDIT FILE FROM HERE
```

Kubernetes:

Kubernetes acts as an orchestrator allowing us to control the deployment, scaling, and management of our containerized application across cluster node.

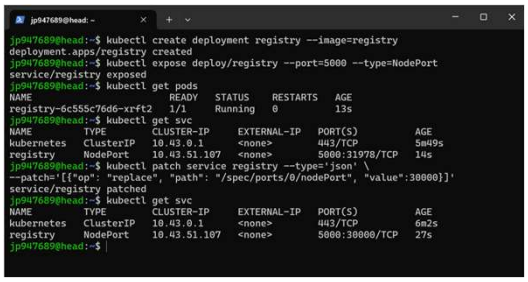
Kubernetes allows us to deploy our docker images within a Kubernetes experiment. Below is a section of our deployment.yml file.

Code Blame Raw

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: 127.0.0.1:30000/mysql:v0.1
        ports:
        - name: mysql-port
          containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: password

```

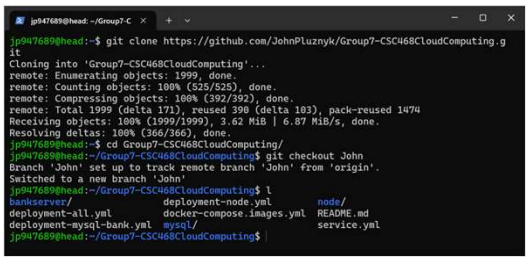
Demo



```
jp947689@head:~$ kubectl create deployment registry --image=registry
deployment.apps/registry created
jp947689@head:~$ kubectl expose deploy/registry --port=5000 --type=NodePort
service/registry exposed
jp947689@head:~$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
registry-6c555c76d6-xrft2   1/1     Running   0          13s
jp947689@head:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.43.0.1      <none>        443/TCP          5m49s
registry     NodePort    10.43.51.107   <none>        5000:31978/TCP   14s
jp947689@head:~$ kubectl patch service registry --type='json' \
--patch='[{"op": "replace", "path": "/spec/ports/0/nodePort", "value":30000}]'
service/registry patched
jp947689@head:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.43.0.1      <none>        443/TCP          6m2s
registry     NodePort    10.43.51.107   <none>        5000:30000/TCP   27s
jp947689@head:~$
```

cd
kubectl create deployment registry --image=registry
kubectl expose deploy/registry --port=5000 --type=NodePort
kubectl get pods
kubectl get svc
kubectl patch service registry --type='json' \ --patch='[{"op": "replace", "path": "/spec/ports/0/nodePort", "value":30000}]'
kubectl get svc

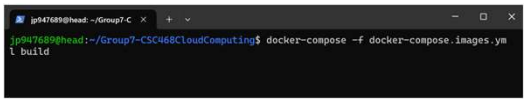
These commands deploy a containerized application named "registry," expose it to the outside world on port 5000, update the NodePort to 30000, and then display information about the pods and services in the cluster. The containerized application "registry" is used to simulate the creation of a local Docker Hub. The registry allows users to push (upload) and pull (download) Docker images from a local repo instead of having to download directly from Docker Hub.



```
jp947689@head:~$ git clone https://github.com/JohnPluznyk/Group7-CSC468CloudComputing.g
it
Cloning into 'Group7-CSC468CloudComputing'...
remote: Enumerating objects: 1999, done.
remote: Counting objects: 100% (525/525), done.
remote: Compressing objects: 100% (392/392), done.
remote: Total 1999 (delta 171), reused 390 (delta 103), pack-reused 1474
Receiving objects: 100% (1999/1999), 3.62 MiB | 6.87 MiB/s, done.
Resolving deltas: 100% (366/366), done.
jp947689@head:~$ cd Group7-CSC468CloudComputing/
jp947689@head:~/Group7-CSC468CloudComputing$ git checkout John
Branch 'John' set up to track remote branch 'John' from 'origin'.
Switched to a new branch 'John'
jp947689@head:~/Group7-CSC468CloudComputing$ l
bankserver/                deployment-node.yml        node/
deployment-all.yml         docker-compose.images.yml  README.md
deployment-mysql-bank.yml  mysql/                     service.yml
jp947689@head:~/Group7-CSC468CloudComputing$
```

git clone https://github.com/JohnPluznyk/Group7-CSC468CloudComputing.git
cd Group7-CSC468CloudComputing
git checkout John --- This command may be optional need if files have been merged with main branch
ls

The following commands download the needed repo which contains the files to setup the FusionBank web application. Be sure to set the current directory to the repo by using the cd command and the repos name. Next use the ls command to check for the necessary files(service.yml, deployment-*.yml, and docker-compose.images.yml). If the following files are not present be sure to use the git checkout command and switch over to the branch John to get the files.



```
jp947689@head:~/Group7-CSC468CloudComputing$ docker-compose -f docker-compose.images.ym
l build
```

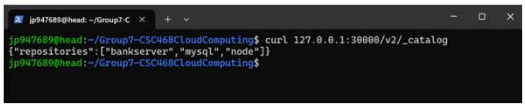
docker-compose –f docker-compose.image.yml build

This command will build all of the dockerfiles within the repo that are needed for the web application.

```
jp947689@head:~/Group7-CSC468CloudComputing$ docker-compose -f docker-compose.images.ym
l push
```

docker-compose –f docker-compose.image.yml push

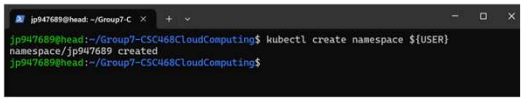
This command push all of the recently build images to the local registry that we have built in the first step.



```
jp947689@head:~/Group7-CSC468CloudComputing$ curl 127.0.0.1:30000/v2/_catalog
{"repositories":["bankserver","mysql","node"]}
jp947689@head:~/Group7-CSC468CloudComputing$
```

curl 127.0.0.1:30000/v2/_catalog

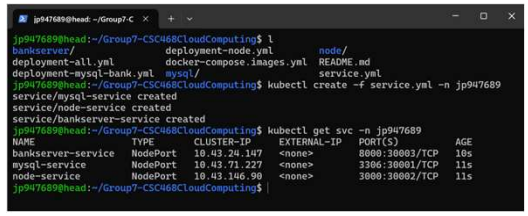
Double check that all of the needed images (bankserver, mysql, node) have been pushed to the local registry.



```
jp947689@head:~/Group7-CSC468CloudComputing$ kubectl create namespace ${USER}
namespace/jp947689 created
jp947689@head:~/Group7-CSC468CloudComputing$
```

Kubectl create namespace ${USER}

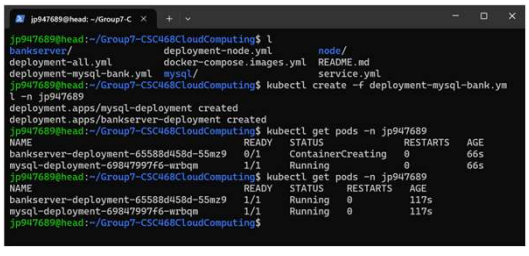
This command allows you to create your own namespace within Kubernetes. Namespaces allow for one to divide cluster resource between multiple users.



```
jp947689@head:~/Group7-CSC468CloudComputing$ l
bankserver/                deployment-node.yml        node/
deployment-all.yml         docker-compose.images.yml  README.md
deployment-mysql-bank.yml  mysql/                     service.yml
jp947689@head:~/Group7-CSC468CloudComputing$ kubectl create -f service.yml -n jp947689
service/mysql-service created
service/node-service created
service/bankserver-service created
jp947689@head:~/Group7-CSC468CloudComputing$ kubectl get svc -n jp947689
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
bankserver-service  NodePort    10.43.24.147    <none>        8000:30003/TCP   10s
mysql-service       NodePort    10.43.71.227    <none>        3306:30001/TCP   11s
node-service        NodePort    10.43.146.90    <none>        3000:30002/TCP   11s
jp947689@head:~/Group7-CSC468CloudComputing$
```

kuebctl create –f service.yml –n ${USER}
Kubectl get svc –n ${USER}

These commands will create the necessary services needed for application. A service in Kubernetes enables communication among various components or microservices within a Kubernetes cluster. In this case the services are binding the exposed ports of our containers to ports of the Kubernetes cluster. Take note of the port(s) column as we will utilize 30002 to access our node.js server.

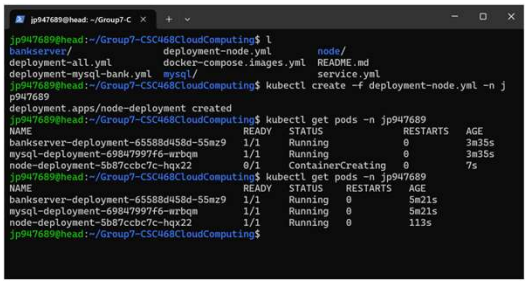


```
jp947689@head:~/Group7-CSC468CloudComputing$ l
bankserver/                deployment-node.yml        node/
deployment-all.yml         docker-compose.images.yml  README.md
deployment-mysql-bank.yml  mysql/                     service.yml
jp947689@head:~/Group7-CSC468CloudComputing$ kubectl create -f deployment-mysql-bank.ym
l -n jp947689
deployment.apps/mysql-deployment created
deployment.apps/bankserver-deployment created
jp947689@head:~/Group7-CSC468CloudComputing$ kubectl get pods -n jp947689
NAME                                  READY   STATUS              RESTARTS   AGE
bankserver-deployment-65588d458d-55mz9   0/1     ContainerCreating   0          66s
mysql-deployment-69847997f6-wrbqm        1/1     Running             0          66s
jp947689@head:~/Group7-CSC468CloudComputing$ kubectl get pods -n jp947689
NAME                                  READY   STATUS    RESTARTS   AGE
bankserver-deployment-65588d458d-55mz9   1/1     Running   0          117s
mysql-deployment-69847997f6-wrbqm        1/1     Running   0          117s
jp947689@head:~/Group7-CSC468CloudComputing$
```

kubectl create –f deployment-mysql-bank.yml –n ${USER}
Kubectl get pods –n ${USER}

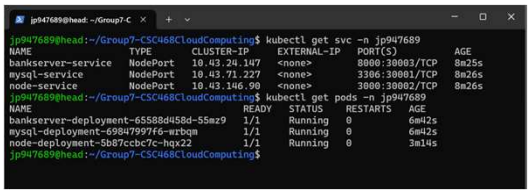
These command will deploy our mysql server and banking server. Make sure after running the pods command that both status say they are running before continuing on to the next step.

Demo Continued



kubectl create –f deployment-node.yml–n ${USER}
Kubectl get pods –n ${USER}

This command will make deploy our NodeJS server.

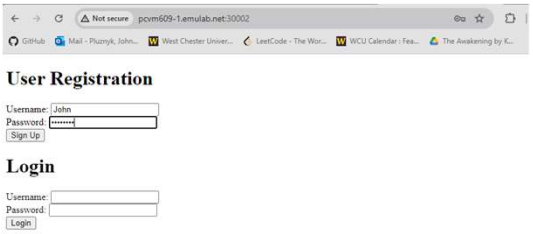


kubectl get svc –n ${USER}
Kubectl get pods –n ${USER}

Be sure to double check that all your services are still active and your pods are up and running.

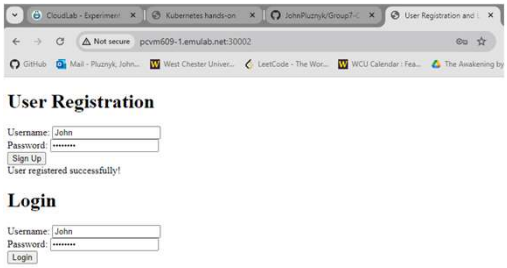
Take note of the hostname of your machine as you will need to use this to access the NodeJS server.

Be sure to double check that all your services are still active and your pods are up and running.



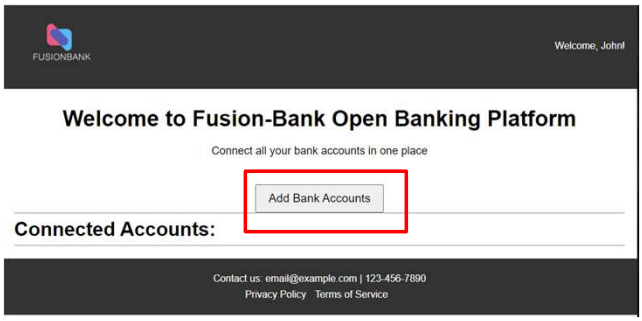
Go to the URL: (hostname):30002

You should see a login screen. Here you can register yourself as a user for the Fusion Bank web app. Enter in a username and password and then signup. After you signing up be sure to use those same credentials to login.

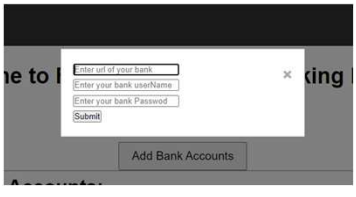


Signup
Login

After signing up you should see a message saying that the user was registered successfully. Here you can then use the same credentials to login.



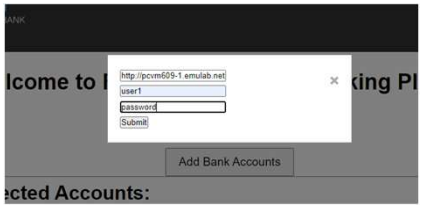
After signing up and logging in one should arrive to the home screen of the Fusion Bank app. Click on the “Add Bank Accounts” button.



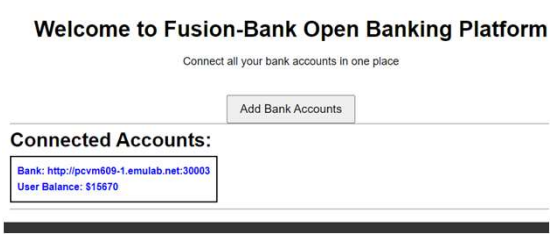
Here on the left is a blank submission form. This submission form will be used to add the bank account you wish to connect. Be sure to type in the URL of the bank server. Which in this case is http://(hostname):30003. Then enter in the username and password of your credentials from the banking server.

NOTE only four users were created on the banking server. The usernames are as follows: user1, user2, user3, user4

All users use the same password: password



Demo Continued



If one submitted the right the information you should see something similar to the image on the left.

Ideally if this was a real-world scenario one would be able to connect multiple accounts and the it would look something similar to what is depicted on the left. Only one banking server was used in this demo but in a real-world scenario one would ideally be able to connect multiple bank accounts all under Fusion Bank.