1. Containerize Applications

1.1. Environment Overview

Your lab environment is hosted with a cloud server instance. This instance is accessible by clicking the “My Lab” icon on the left of your Strigo web page. You can access the terminal session using the built-in interface provided by Strigo or feel free to SSH directly as well (instructions will appear in the terminal).

The appropriate kubernetes related tools (docker, kubectl, kubernetes, etc.) have already been installed ahead of time.

1.2. Prepare Your Lab

Step 1: Start Kubernetes and your Docker Registry

```
k8s-start
```

Step 2: Download lab files and code

For convenience all files, scripts, etc that we will be using for the remainder of this course have been packaged up into an archive. Let’s download the archive and unpack it.

```
cd $HOME
wget http://localhost:8081/_static/lab-files.tar.gz
tar zxvf lab-files.tar.gz
```

If you would like a copy of these files to have on your laptop to have later on, the following command will provide you with a URL you can use to download a copy.

```
echo "http://`curl -s http://169.254.169.254/latest/meta-data/public-ipv4`:8081/_static/lab-
files.tar.gz"
```

1.3. Build Docker image for your frontend application

Step 1: Analyze Dockerfile for your frontend application

```
cd $HOME/gowebapp/gowebapp
```

Let's inpsect the `Dockerfile`. You can do so be opening it in your favorite command line editor, use the built-in editor, or we have embedded the file directly in the lab instructions here for easy viewing. This Dockerfile contains the instructions to tell the Docker build engine how to create an image for our gowebapp.

Dockerfile

```
FROM ubuntu

COPY ./code /opt/gowebapp
COPY ./config /opt/gowebapp/config

EXPOSE 80

WORKDIR /opt/gowebapp/
ENTRYPOINT ["/opt/gowebapp/gowebapp"]
```

Step 3: Build gowebapp Docker image locally

```
cd $HOME/gowebapp/gowebapp
```

Build the gowebapp image locally. Make sure to include "." at the end. Make sure the build runs to completion without errors. You should get a success message.

```
docker build -t gowebapp:v1 .
```

1.4. Build Docker image for your backend application

Step 1: Analyze Dockerfile for your backend application

```
cd $HOME/gowebapp/gowebapp-mysql
```

Let’s inpsect the `Dockerfile`. You can do so be opening it in your favorite command line editor, use the built-in editor, or we have embedded the file directly in the lab instructions here for easy viewing. This Dockerfile contains the instructions to tell the Docker build engine how to create an image for the backend MySQL database.

Dockerfile

```
FROM mysql:5.6

COPY gowebapp.sql /docker-entrypoint-initdb.d/
```

Step 2: Build gowebapp-mysql Docker image locally

```
cd $HOME/gowebapp/gowebapp-mysql
```

Build the gowebapp-mysql image locally. Make sure to include “.“ at the end. Make sure the build runs to completion without errors. You should get a success message.

```
docker build -t gowebapp-mysql:v1 .
```

1.5. Run and test Docker images locally

Before deploying to Kubernetes, let’s run and test the Docker images locally, to ensure that the frontend and backend containers run and integrate properly.

Step 1: Create Docker user-defined network

To facilitate cross-container communication, let’s first define a user-defined network in which to run the frontend and backend containers:

```
docker network create gowebapp
```

Step 2: Launch frontend and backend containers

Next, let’s launch a frontend and backend container using the Docker CLI.

```
docker run --net gowebapp --name gowebapp-mysql --hostname gowebapp-mysql -d -e
MYSQL_ROOT_PASSWORD=mypassword gowebapp-mysql:v1

sleep 20

docker run -p 80:80 --net gowebapp -d --name gowebapp --hostname gowebapp gowebapp:v1
```

First, we launched the database container, as it will take a bit longer to startup, and the frontend container depends on it. Notice how we are injecting the database password into the MySQL configuration as an environment variable:

Next we paused for 20 seconds to give the database a chance to start and initialize.

Finally we launched a frontend container, mapping the container port `80` - where the web application is exposed - to port `80` on the host machine:

Step 3: Test the application locally

Now that we've launched the application containers, let's try to test the web application locally. You should be able to access the application at `http://<EXTERNAL-IP>`

Note

To get the EXTERNAL-IP of your host, run the command `lab-info` in your lab terminal

Create an account and login. Write something on your Notepad and save it. This will verify that the application is working and properly integrates with the backend database container.

Step 4: Inspect the MySQL database

Let's connect to the backend MySQL database container and run some queries to ensure that application persistence is working properly:

```
docker exec -it gowebapp-mysql mysql -u root -pmypassword gowebapp
```

Once connected, run some simple SQL commands to inspect the database tables and persistence:

```
#Simple SQL to navigate
SHOW DATABASES;
USE gowebapp;
SHOW TABLES;
SELECT * FROM <table_name>;
exit;
```

Step 5: Cleanup application containers

When we're finished testing, we can terminate and remove the currently running frontend and backend containers from our local machine:

```
docker rm -f gowebapp gowebapp-mysql
```

1.6. Create and push Docker images to Docker registry

Step 1: Tag images to target another registry

We are finished testing our images. We now need to push our images to an image registry so our Kubernetes cluster can access them. First, we need to tag our Docker images to use the registry in your lab environment:

```
docker tag gowebapp:v1 localhost:5000/gowebapp:v1

docker tag gowebapp-mysql:v1 localhost:5000/gowebapp-mysql:v1
```

Step 2: publish images to the registry

```
docker push localhost:5000/gowebapp:v1
docker push localhost:5000/gowebapp-mysql:v1
```

1.7. Lab 01 Conclusion

Congratulations! By containerizing your application components, you have taken the first important step toward deploying your application to Kubernetes.