Intro to Containers and Kubernetes Documentation

None

VMware, Inc.

Jul 28, 2021

CHAPTER

ONE

1. CONTAINERIZE APPLICATIONS

1.1 1.1. Environment Overview

Your lab environment is hosted on our cloud environment. The appropriate kubernetes related tools (docker, kubectl, kubernetes, etc.) have already been installed ahead of time.

If you would like to view this Guide in a different window so you can work side-by-side with the lab, it can be accomplished by right-clicking on the home/house logo and selecting the option to open in a new window.

1.2 1.2. Prepare Your Lab

1.2.1 Step 1: Kubernetes and your Docker Registry

There should be a registry running that we will push our images to:

```
echo $REGISTRY_HOST
```

1.2.2 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.

1.3 1.3. Build Docker image for your frontend application

1.3.1 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 8080
USER 1000

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

1.3.2 Step 2: 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 1.4. Build Docker image for your backend application

1.4.1 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

USER 1000

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

1.4.2 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 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.

1.5.1 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
```

1.5.2 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 8080:8080 --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 `8080` - where the web application is exposed - to port `8080` on the host machine:

1.5.3 Step 3: Test the application locally

Now that we've launched the application containers, let's try to test the web application locally.

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.

1.5.4 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;
```

(continues on next page)

(continued from previous page)

```
SELECT * FROM <table_name>;
exit;
```

1.5.5 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 1.6. Create and push Docker images to Docker registry

1.6.1 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 $REGISTRY_HOST/gowebapp:v1

docker tag gowebapp-mysql:v1 $REGISTRY_HOST/gowebapp-mysql:v1
```

1.6.2 Step 2: publish images to the registry

```
docker push $REGISTRY_HOST/gowebapp:v1
docker push $REGISTRY_HOST/gowebapp-mysql:v1
```

1.7 1.7. Lab 01 Conclusion

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

CHAPTER

TWO

2. DEPLOY APPLICATIONS USING KUBERNETES

2.1 2.1. Getting Started with kubectl

`kubectl` is the command line interface for interacting with a Kubernetes cluster. Let's explore some of it's features.

2.1.1 Step 1: introduction to kubectl

By executing kubectl you will get a list of options you can utilize kubectl for. kubectl allows you to control Kubernetes cluster manager.

```
kubectl
```

2.1.2 Step 2: use kubectl to understand an object

Use explain to get documentation of various resources. For instance pods, nodes, services, etc.

```
kubectl explain pods
```

2.1.3 Step 3: get more information on an object

Shortcut to object names:

```
kubectl describe
```

2.1.4 Step 4: autocomplete

Use TAB to autocomplete:

```
kubectl describe <TAB> <TAB>
```

2.2 2.2. Create Service Object for MySQL

2.2.1 Step 1: Analyze a Kubernetes Service object for the backend MySQL database

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

Let's inpsect the `gowebapp-mysql-service.yaml` file. 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 kubernetes configuration file contains the definition of what the desired state should be for our Kubernetes Service object for the backend MySQL database should be.

`gowebapp-mysql-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: gowebapp-mysql
  labels:
    run: gowebapp-mysql
    tier: backend
spec:
  type: ClusterIP
  ports:
  - port: 3306
    targetPort: 3306
  selector:
    run: gowebapp-mysql
    tier: backend
```

2.2.2 Step 2: create a Service defined above

Use kubectl to create the Service defined above

```
kubectl apply -f gowebapp-mysql-service.yaml
```

2.2.3 Step 3: test to make sure the Service was created

```
kubectl get service -l "run=gowebapp-mysql"
```

2.3 2.3. Create Deployment Object for MySQL

2.3.1 Step 1: Analyze a Kubernetes Deployment object for the backend MySQL database

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

Let's inpsect the `gowebapp-mysql-deployment.yaml` file. 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 kubernetes configuration file contains the definition of what the desired state should be for our Kubernetes Deployment object for the backend MySQL database should be.

gowebapp-mysql-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gowebapp-mysql
  labels:
    run: gowebapp-mysql
    tier: backend
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      run: gowebapp-mysql
      tier: backend
  template:
    metadata:
      labels:
        run: gowebapp-mysql
        tier: backend
    spec:
      containers:
      - name: gowebapp-mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: mypassword
        image: gowebapp-mysql:v1
        ports:
        - containerPort: 3306
```

We are using a custom image that we created in the previous lab. Therefore we need to addd the registry server to the *image:* line in the YAML so that Kubernetes knows which registry to pull the image from. Otherwise it will try to find the image on the public/default configured registry server.

```
sed -i s/"image: gowebapp"/"image: $REGISTRY_HOST\/gowebapp"/g gowebapp-mysql-
↪deployment.yaml
```

2.3.2 Step 2: create the Deployment defined above

Use kubectl to create the Deployment defined above

```
kubectl apply -f gowebapp-mysql-deployment.yaml
```

2.3.3 Step 3: test to make sure the Deployment was created

```
kubectl get deployment -l "run=gowebapp-mysql"
```

2.4 2.4. Create Service Object for frontend application: gowebapp

2.4.1 Step 1: Analyze a Kubernetes Service object for the frontend gowebapp

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

Let's inpsect the `gowebapp-service.yaml` file. 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 kubernetes configuration file contains the definition of what the desired state should be for our Kubernetes Service object for the frontend gowebapp.

`gowebapp-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: gowebapp
  labels:
    run: gowebapp
    tier: frontend
spec:
  type: NodePort
  ports:
  - port: 8080
  selector:
    run: gowebapp
    tier: frontend
```

2.4.2 Step 2: create a Service defined above

Use kubectl to create the service defined above

```
kubectl apply -f gowebapp-service.yaml
```

Step 3: test to make sure the Service was created

```
kubectl get service -l "run=gowebapp"
```

2.5 2.5. Create Deployment Object for gowebapp

2.5.1 Step 1: Analyze a Kubernetes Deployment object for the frontend gowebapp

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

Let's inpsect the `gowebapp-deployment.yaml` file. 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 kubernetes configuration file contains the definition of what the desired state should be for our Kubernetes Deployment object for the frontend gowebapp.

`gowebapp-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gowebapp
  labels:
    run: gowebapp
    tier: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      run: gowebapp
      tier: frontend
  template:
    metadata:
      labels:
        run: gowebapp
        tier: frontend
    spec:
      containers:
      - name: gowebapp
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: mypassword
        image: gowebapp:v1
        ports:
        - containerPort: 80
```

We are using a custom image that we created in the previous lab. Therefore we need to addd the registry server to the *image:* line in the YAML so that Kubernetes knows which registry to pull the image from. Otherwise it will try to find the image on the public/default configured registry server.

```
sed -i s/"image: gowebapp"/"image: $REGISTRY_HOST\/gowebapp"/g gowebapp-deployment.
↪yaml
```

2.5.2 Step 2: create the Deployment defined above

Use kubectl to create the service defined above

```
kubectl apply -f gowebapp-deployment.yaml
```

2.5.3 Step 3: test to make sure the Deployment was created

```
kubectl get deployment -l "run=gowebapp"
```

2.6 2.6. Test Your Application

2.6.1 Step 1: Access gowebapp through the Service

You can now sign up for an account, login and use the Notepad.

Warning: Note: Your browser may cache an old instance of the gowebapp application from previous labs. When the webpage loads, look at the top right. If you see 'Logout', click it. You can then proceed with creating a new test account.

2.7 2.7. Lab 02 Conclusion

Congratulations! You have successfully deployed your applications using Kubernetes!