Corndel DevOps Engineering Programme

in association with Softwire

Module 14: Project Exercise

Corndel Digital.

DB6F

Module 14

Project Exercise Brief

In this exercise we will set up our To-Do app on a local **minikube** cluster. We'll start with a simple Nginx app, then replace it with the To-Do app.

What's minikube?

Minikube is a version of Kubernetes that you can run locally on a development machine. It can be a great way to learn about Kubernetes and test changes locally without having to set up and pay for hosting.

Prerequisites

We'll need these tools during the exercise.

- Docker
    - Docker Desktop (Windows)
    - Docker Desktop (Mac)
    - Docker Engine (Linux)
- Kubectl
- minikube

Spinning up a minikube cluster

Let's spin up a minikube cluster running a simple Nginx server.

Run `minikube start` in an admin terminal to spin up your minikube cluster.

If you get an error like `minikube command not found`, then make sure that `minikube` is on your `PATH` and restart your terminal.

Create a `deployment.yaml` file based on the `nginx` image.

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: module-14
spec:
  selector:
    matchLabels:
      app: module-14
  replicas: 1
  template:
    metadata:
      labels:
        app: module-14
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
          - containerPort: 80
```

Run `kubectl apply -f deployment.yaml` to deploy a Pod running the `nginx` image.

Create a `service.yaml` file defining a Service that will provide access to the Pod.

```
# service.yaml
kind: Service
apiVersion: v1
metadata:
  name: module-14
spec:
  type: NodePort
  selector:
    app: module-14
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Run `kubectl apply -f service.yaml` to deploy the Service.

After each deployment, we need to run `kubectl port-forward service/module-14 7080:80` to link up our minikube Service with a port on `localhost`.

Open http://localhost:7080/ in a browser to view the Nginx app.

If you run into any issues then you might find these commands helpful:

- `kubectl get pods`
  List out the cluster's Pods, along with their names
- `kubectl logs my-pod`
  Retrieve the logs for a Pod called `my-pod`

Building a Docker image

Before we can get our To-Do app running on minikube, we'll need to create a Docker image for our Pod. Navigate to the folder containing your Module 13 application and run `docker build --target production --tag todo-app:prod .` to build the image.

On Windows, Docker Desktop has to be run by a user with administrator privileges. If you're on Windows then you'll need to open Docker Desktop and your terminal using the same administrator account.

Setting up a database

Our application will also need a MongoDB database. We'll host our database on MongoDB Atlas, as in Module 9. You can either reuse your cluster from Module 9 or create a new one.

Find your cluster in Atlas and note down the connection string for later.

Using our Docker image in minikube

Before we can swap over our Pod to use the new To-Do app image, we'll need to load it into minikube's local image store using the `minikube image load` command.

We can then update the `deployment.yaml` manifest to use our new image, along with the required environment variables. We also need to set the `imagePullPolicy` to `Never` so minikube doesn't try to pull the image from a registry.

Now we can run `kubectl apply -f deployment.yaml` to deploy our To-Do app.

We'll move the sensitive environment variables into Secrets later in the exercise.

Viewing logs in Loggly

We've deployed our To-Do app and provided it with a Loggly token, so you should be able to see logs coming through in Loggly as you use the application.

Moving sensitive information to Secrets

At this point we have a fully functional To-Do app running on minikube. However, we're using environment variables to store sensitive information:

- `LOGGLY_TOKEN`
- `SECRET_KEY`
- `MONGODB_CONNECTION_STRING`
- `GITHUB_CLIENT_ID`
- `GITHUB_CLIENT_SECRET`

We should move these into Secrets instead.

For each sensitive environment variable, create a Secret and reference it in `deployment.yaml`. Redeploy the Deployment to swap out the environment variables with Secrets.

Stretch goals

Using a local MongoDB database instead provides some interesting challenges, so let's try that too!

Start out by installing MongoDB Community Edition.

If you don't already have a MongoDB client, then include `MongoDB Compass` during the installation.

Setting up a local database

If you installed MongoDB as a Windows Service, then you should already have a running MongoDB server on `localhost`. Otherwise, you can run `mongod` in an admin terminal to start up the server.

Use MongoDB Compass, or an alternative MongoDB client, to connect to the MongoDB server on `localhost` and create a database. Note down the connection string for later.

If you get an error like `mongod command not found`, then make sure that `mongod` is on your `PATH` and restart your terminal.

If you get an error like `Data directory C:\data\db\ not found`, then manually create those directories and run `mongod` again. Alternatively, create those directories in a different location and use the `--dbpath` argument, e.g. `mongod --dbpath="C:\Program Files\MongoDB\Server\4.4\data\db"`.

Connecting to a local database from minikube

Update the `MONGODB_CONNECTION_STRING` Secret with your new connection string. Run `kubectl rollout restart` to restart the Pod so it uses the updated secret.

If you used `localhost` in your connection string, then you should see a database timeout when you open the To-Do app. This is because the Pod is using its own internal `localhost`, rather than the host machine's `localhost` (i.e. your laptop's `localhost`).

We'll need to find out the host machine's IP address and use that instead. You should be able to find this IP mapped to `host.minikube.internal` in the minikube VM's `/etc/hosts` file.

After replacing `localhost` with the host machine's IP address, redeploy the Deployment to swap over to the local database.

You can run `minikube ssh` to access the VM.

If you're using Docker as your minikube driver, then you should be able to use `host.docker.internal` too.

Moving the host IP address into hostAliases

Ideally, we would use `host.minikube.internal` in the MongoDB connection string, keeping the IP address logic separate. However, Pods don't have `host.minikube.internal` entries in their `/etc/hosts` files. This means that Pods don't resolve `host.minikube.internal` to the host machine's IP address by default.

We can solve this by using hostAliases) to add a `host.minikube.internal` entry in each Pod's `/etc/hosts` file.

Add a `hostAliases` section to the Deployment, mapping `host.minikube.internal` to the host machine's IP address. Update the connection string to use `host.minikube.internal`, then redeploy the Deployment to finish setting up.