

Ceng495

Homework 2 Report

Berke Sina Ahlatci 2468502

Introduction

Among the provided demo projects, I chose to work with [ACME Air](#). The project's relatively small size seemed suitable for a hands-on experience, and I assumed that being listed first might have provided more resources for the project.

Installation

This section provides a step-by-step guide on installing and configuring the ACME Air application. The prerequisites for this task are as follows:

Pre-requisites

- Docker
- Kubernetes
- Skaffold
- Kubectl

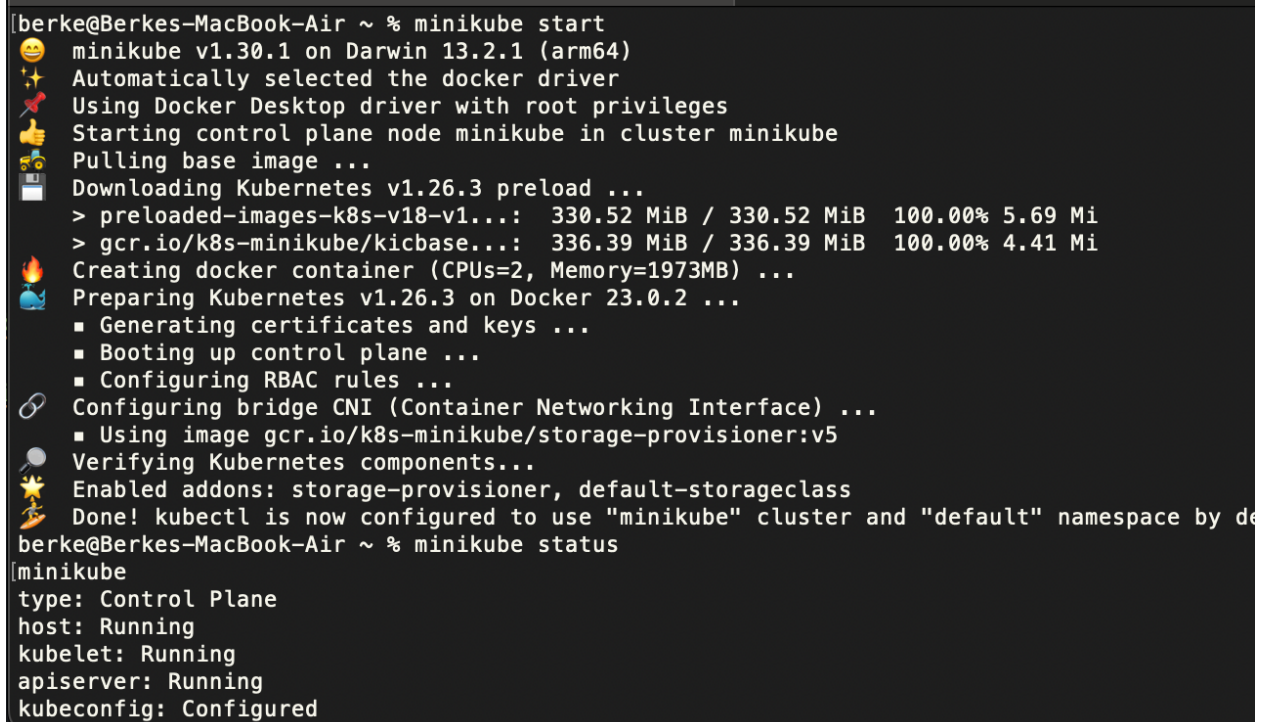
I started with installing **Minikube** and **Skaffold**, followed by cloning the [acmeair-nodejs](#) repository.

After successful installation, I used the following command to start **Minikube**:

```
> minikube start
```

The image below showcases the terminal. Following command provides additional properties

```
> minikube status
```

A terminal window screenshot showing the execution of minikube commands. The first command is 'minikube start', which initiates the setup of a Kubernetes cluster. The output shows various steps including downloading the base image, Kubernetes components, and configuring the control plane. The second command is 'minikube status', which shows the current state of the cluster components.

```
[berke@Berkes-MacBook-Air ~ % minikube start
🐹 minikube v1.30.1 on Darwin 13.2.1 (arm64)
🌟 Automatically selected the docker driver
🔧 Using Docker Desktop driver with root privileges
👍 Starting control plane node minikube in cluster minikube
📦 Pulling base image ...
📦 Downloading Kubernetes v1.26.3 preload ...
> preloaded-images-k8s-v18-v1...: 330.52 MiB / 330.52 MiB 100.00% 5.69 Mi
> gcr.io/k8s-minikube/kicbase...: 336.39 MiB / 336.39 MiB 100.00% 4.41 Mi
🔥 Creating docker container (CPUs=2, Memory=1973MB) ...
🐳 Preparing Kubernetes v1.26.3 on Docker 23.0.2 ...
  ▪ Generating certificates and keys ...
  ▪ Booting up control plane ...
  ▪ Configuring RBAC rules ...
🔗 Configuring bridge CNI (Container Networking Interface) ...
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🔍 Verifying Kubernetes components...
🌟 Enabled addons: storage-provisioner, default-storageclass
🏁 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
berke@Berkes-MacBook-Air ~ % minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

Configure the Acme Air Application

After successfully installing the Acme Air application, I've continued with configuring the project to my specific needs. I created configuration files for **kubernetes** and **skaffold**.

Create a Kubernetes Deployment Configuration

Creating a Kubernetes deployment involves writing a YAML configuration file that represents the desired state of your application. I've created a file named *acme-air-deployment.yaml*.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: acme-air-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: acme-air
  template:
    metadata:
      labels:
        app: acme-air
    spec:
      containers:
        - name: acme-air-container
          image: acme-air-image
          ports:
            - containerPort: 9080
```

In this configuration;

- **metadata.name:** This is the name of the deployment that you'll be creating.
- **spec.replicas:** This is the number of pod instances you want to run in the cluster.
- **spec.selector:** This is how the deployment identifies the pods that it manages.
- **spec.template:** The template that will be used to create the pods.
- **spec.template.spec.containers:** A list of containers that should be launched in each pod.
- **spec.template.spec.containers.image:** The image that will be used to create the containers.
- **spec.template.spec.containers.ports:** The ports that the container will expose.

Deploy the Application using Skaffold

Skaffold is a tool designed to facilitate continuous development for Kubernetes applications. I set up Skaffold by creating a YAML file named *skaffold.yaml*

```
apiVersion: skaffold/v2beta21
kind: Config
metadata:
  name: acme-air
build:
  artifacts:
    - image: acme-air-image
deploy:
  kubectl:
    manifests:
      - acme-air-deployment.yaml
```

In this configuration;

- **apiVersion:** This indicates the Skaffold version.
- **kind:** This indicates the kind of the object, which is Config in this case.
- **metadata.name:** This is the name of the Skaffold configuration.
- **build.artifacts:** This is a list of artifacts to be built, with each artifact being built from a Dockerfile.
- **deploy:** This section contains all the configuration needed for the deployment phase.
- **deploy.kubectl.manifests:** This is a list of Kubernetes manifest files that Skaffold can use to deploy your application.

With the Skaffold configuration in place, I deployed the application using the following command in the terminal:

```
> skaffold run
```

Visible Changes in the Frontend

After successfully deploying the ACME Air application, I made modifications to the **main logo**, **catchphrase** and **actions** sections on the frontend.

Expose the Frontend

In order to make the application accessible, we need to expose it as a service within Kubernetes. This step involves creating a service configuration file named *acme-air-service.yaml*

```
apiVersion: v1
kind: Service
metadata:
  name: acmeair-service
spec:
  type: NodePort
  ports:
    - port: 9080
      targetPort: 9080
      protocol: TCP
  selector:
    app: acme-air
```

In this configuration;

- **apiVersion:** The version of Kubernetes API used.
- **kind:** Indicates creating a Service.
- **metadata.name:** Name of the service.
- **spec.type:** Service type, exposing it outside the cluster.
- **spec.ports:** List of ports the service exposes.
- **spec.ports.port:** Port accessible by other components in the cluster.
- **spec.ports.targetPort:** The port the application listens to inside the pod.
- **spec.ports.protocol:** Protocol for the port.
- **spec.selector:** Determines which pods receive traffic. Here, it's any pods labeled acme-air. This should match the labels in the Deployment.

In the final step, I exposed the frontend of the application to make it accessible from my browser. I used the command

```
> minikube service acme-air-service
```

```
|-----|
| NAMESPACE | | NAME | | TARGET PORT | | URL |
|-----|
| default | | acme-air-service | | 80 | | http://192.168.49.2:32491 |
|-----|
🔗 Starting tunnel for service acme-air-service.
|-----|
| NAMESPACE | | NAME | | TARGET PORT | | URL |
|-----|
| default | | acme-air-service | | | | http://127.0.0.1:53593 |
|-----|
🔗 Opening service default/acme-air-service in default browser...
! Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

I then checked the status of the deployments, pods, and services using

```
> kubectl get deployments/pods/services
```

```
[berke@Berkes-MacBook-Air acmeair-nodejs % kubectl get deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
acme-air-deployment 1/1     1            1           8m12s
[berke@Berkes-MacBook-Air acmeair-nodejs % kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
acme-air-deployment-667c886657-k5d9q 1/1     Running   0          5m6s
[berke@Berkes-MacBook-Air acmeair-nodejs % kubectl get services
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
acme-air-deployment LoadBalancer  10.105.159.62 <pending>    32491:31670/TCP  114m
acme-air-service    LoadBalancer  10.103.237.78 <pending>    80:32491/TCP     3h13m
kubernetes          ClusterIP     10.96.0.1     <none>       443/TCP          4h1m
```

Visuals



Conclusion

In conclusion, this report presents an overview of deploying the **ACME Air** application on a **Minikube Kubernetes** cluster using **Scaffold**. The process includes setting up the environment, configuring the application, creating a Kubernetes deployment configuration, and deploying the application. By following these steps, we can automate the build, push, and deploy stages of the development workflow seamlessly.