Test your Ruby on Rails deployment

Table of contents

Prerequisites

- Complete all the previous sections of this guide, starting with Containerize a Ruby on Rails application.
- Turn on Kubernetes in Docker Desktop.

Overview

In this section, you'll learn how to use Docker Desktop to deploy your application to a fully-featured Kubernetes environment on your development machine. This lets you to test and debug your workloads on Kubernetes locally before deploying.

Create a Kubernetes YAML file

In your `docker-ruby-on-rails` directory, create a file named `docker-ruby-on-rails-kubernetes.yaml`. Open the file in an IDE or text editor and add the following contents. Replace `DOCKER_USERNAME/REPO_NAME` with your Docker username and the name of the repository that you created in Configure CI/CD for your Ruby on Rails application.

Give feedback

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: docker-ruby-on-rails-demo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      service: ruby-on-rails
  template:
    metadata:
      labels:
        service: ruby-on-rails
    spec:
      containers:
        - name: ruby-on-rails-container
          image: DOCKER_USERNAME/REPO_NAME
          imagePullPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: docker-ruby-on-rails-demo
  namespace: default
spec:
  type: NodePort
  selector:
    service: ruby-on-rails
  ports:
    - port: 3000
      targetPort: 3000
      nodePort: 30001
```

In this Kubernetes YAML file, there are two objects, separated by the `---`:

- A Deployment, describing a scalable group of identical pods. In this case, you'll get just one replica, or copy of your pod. That pod, which is described under `template`, has just one container in it. The container is created from the image built by GitHub Actions in Configure CI/CD for your Ruby on Rails application.

- A NodePort service, which will route traffic from port 30001 on your host to port 8001 inside the pods it routes to, allowing you to reach your app from the network.

To learn more about Kubernetes objects, see the [Kubernetes documentation](#).

Deploy and check your application

1. In a terminal, navigate to `docker-ruby-on-rails` and deploy your application to Kubernetes.

   ```
   $ kubectl apply -f docker-ruby-on-rails-kubernetes.yaml
   ```

   You should see output that looks like the following, indicating your Kubernetes objects were created successfully.

   ```
   deployment.apps/docker-ruby-on-rails-demo created
   service/docker-ruby-on-rails-demo created
   ```

2. Make sure everything worked by listing your deployments.

   ```
   $ kubectl get deployments
   ```

   Your deployment should be listed as follows:

   ```
   NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
   docker-ruby-on-rails-demo   1/1     1            1           15s
   ```

   This indicates all one of the pods you asked for in your YAML are up and running. Do the same check for your services.

   ```
   $ kubectl get services
   ```

   You should get output like the following.

   ```
   NAME                        TYPE        CLUSTER-IP      EXTERNAL-IP   POF
   kubernetes                  ClusterIP   10.96.0.1       <none>        443
   docker-ruby-on-rails-demo   NodePort    10.99.128.230   <none>        30€
   ```

In addition to the default `kubernetes` service, you can see your `docker-ruby-on-rails-demo` service, accepting traffic on port 30001/TCP.

3. T[illegible]ing on Kubernetes, you need to follow these steps.

   Get the Current Pods: First, you need to identify the pods running in your Kubernetes cluster. Execute the following command to list the current pods in the `default` namespace:

   ```
   # Get the current pods in the cluster in the namespace default
   $ kubectl get pods
   ```

   This command will display a list of all pods in the `default` namespace. Look for the pod with the prefix `docker-ruby-on-rails-demo-`. Here is an example output:

   ```
   NAME                                         READY   STATUS    RESTARTS
   docker-ruby-on-rails-demo-7cbddb5d6f-qh44l   1/1     Running   2 (22h agc
   ```

   Execute the Migration Command: Once you've identified the correct pod, use the `kubectl exec` command to run the database migration inside the pod.

   ```
   $ kubectl exec -it docker-ruby-on-rails-demo-7cbddb5d6f-qh44l -- rails dt
   ```

   This command opens an interactive terminal session (`-it`) in the specified pod and runs the `rails db:migrate` command with the environment set to development (`RAILS_ENV=development`).

   By following these steps, you ensure that your database is properly migrated within the Ruby on Rails application running in your Kubernetes cluster. This process helps maintain the integrity and consistency of your application's data structure during deployment and updates.

4. Open the browser and go to http://localhost:30001, you should see the ruby on rails application working.

5. Run the following command to tear down your application.

```
$ kubectl delete -f docker-ruby-on-rails-kubernetes.yaml
```

Summary

In this section, you learned how to use Docker Desktop to deploy your application to a fully-featured Kubernetes environment on your development machine.

Related information:

- Kubernetes documentation
- Deploy on Kubernetes with Docker Desktop
- Swarm mode overview