Use containers for Ruby on Rails development

Table of contents

Prerequisites

Complete Containerize a Ruby on Rails application.

Overview

In this section, you'll learn how to set up a development environment for your containerized application. This includes:

- Adding a local database and persisting data
- Configuring Compose to automatically update your running Compose services as you edit and save your code

Add a local database and persist data

You can use containers to set up local services, like a database. In this section, you'll update the `compose.yaml` file to define a database service and a volume to p

In the cloned repository's directory, open the `compose.yaml` file in an IDE or text editor. You need to add the database password file as an environment variable to the server service and specify the secret file to use.

The following is the updated `compose.yaml` file.

```
services:
  web:
    build: .
    command: bundle exec rails s -b '0.0.0.0'
    ports:
      - "3000:3000"
    depends_on:
      - db
    environment:
      - RAILS_ENV=test
    env_file: "webapp.env"
  db:
    image: postgres:latest
    secrets:
      - db-password
    environment:
      - POSTGRES_PASSWORD_FILE=/run/secrets/db-password
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
secrets:
  db-password:
    file: db/password.txt
```

> **Note**
>
> To learn more about the instructions in the Compose file, see Compose file reference.

Before you run the application using Compose, notice that this Compose file specifies a `password.txt` file to hold the database's password. You must create this file as it's not included in the source repository.

In the cloned repository's directory, create a new directory named `db` and inside that directory create a file named `password.txt` that contains the password for the database. Using your favorite IDE or text editor, add the following contents to the `password.txt` file.

```
mysecretpassword
```

Save and close the `password.txt` file. In addition, in the file `webapp.env` you can change the password to connect to the database.

You should now have the following contents in your `docker-ruby-on-rails` directory.

```
.
├── Dockerfile
├── Gemfile
├── Gemfile.lock
├── README.md
├── Rakefile
├── app/
├── bin/
├── compose.yaml
├── config/
├── config.ru
├── db/
│   ├── development.sqlite3
│   ├── migrate
│   ├── password.txt
│   ├── schema.rb
│   └── seeds.rb
├── lib/
├── log/
├── public/
├── storage/
├── test/
├── tmp/
└── vendor
```

Now, run the following `docker compose up` command to start your application.

```
$ docker compose up --build
```

In Ruby on Rails, `db:migrate` is a Rake task that is used to run migrations on the database. Migrations are a way to alter the structure of your database schema over time in a consistent and easy way.

```
$ docker exec -it docker-ruby-on-rails-web-1 rake db:migrate RAILS_ENV=test
```

You will see a similar message like this:

```
console == 20240710193146 CreateWhales: migrating
==================================== -- create_table(:whales) -> 0.0126s ==
20240710193146 CreateWhales: migrated (0.0127s) ============================
```

Refresh http://localhost:3000 in your browser and add the whales.

Press `ctrl+c` in the terminal to stop your application and run `docker compose up` again, the whales are being persisted.

Automatically update services

Use Compose Watch to automatically update your running Compose services as you edit and save your code. For more details about Compose Watch, see Use Compose Watch.

Open your `compose.yaml` file in an IDE or text editor and then add the Compose Watch instructions. The following is the updated `compose.yaml` file.

```
services:
  web:
    build: .
    command: bundle exec rails s -b '0.0.0.0'
    ports:
      - "3000:3000"
    depends_on:
      - db
    environment:
      - RAILS_ENV=test
    env_file: "webapp.env"

    develop:
      watch:
        - action: rebuild
```

```
          path: .
  db:
    image: postgres:latest
    secrets:
      - db-password
    environment:
      - POSTGRES_PASSWORD_FILE=/run/secrets/db-password
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
secrets:
  db-password:
    file: db/password.txt
```

Run the following command to run your application with Compose Watch.

```
$ docker compose watch
```

Any changes to the application's source files on your local machine will now be immediately reflected in the running container.

Open `docker-ruby-on-rails/app/views/whales/index.html.erb` in an IDE or text editor and update the `Whales` string by adding a exclamation marks.

```
-    <h1>Whales</h1>
+    <h1>Whales!</h1>
```

Save the changes to `index.html.erb` and then wait a few seconds for the application to rebuild. Go to the application again and verify that the updated text appears.

Press `ctrl+c` in the terminal to stop your application.

Summary

In this section, you took a look at setting up your Compose file to add a local database and persist data. You also learned how to use Compose Watch to automatically rebuild and run your container when you update your code.

Related information:

- Compose file reference
- Compose file watch
- Multi-stage builds

Next steps

In the next section, you'll take a look at how to set up a CI/CD pipeline using GitHub Actions.

Configure CI/CD for your Ruby on Rails application »