

[Home](#) / [Guides](#) / [Ruby](#) / Containerize your app

Containerize a Ruby on Rails application

Table of contents

[Prerequisites](#)

[Overview](#)

[Initialize Docker assets](#)


[Run the application](#)

[Run the application in the background](#)


[Summary](#)


[Next steps](#)

Prerequisites

- You have installed the latest version of [Docker Desktop](#).
- You have a [Git client](#) . The examples in this section show the Git CLI, but you can use any client.

Overview

This section walks you through containerizing and running a [Ruby on Rails](#)  application.

Starting from Rails 7.1 [Docker is supported out of the box](#) . This means that you will get a `Dockerfile`, `.dockerignore` and `bin/docker-entrypoint` files generated for you when you create a new Rails application.

If you have an existing Rails application, you will need to create the Docker assets manually. Unfortunately `docker init` command does not yet support Rails. Thi

[Give feedback](#)

working with Rails, you'll need to copy Dockerfile and other related configurations manually from the examples below.

Initialize Docker assets

Rails 7.1 generates multistage Dockerfile out of the box, below is an example of such file generated from a Rails template.

Multistage Dockerfiles help create smaller, more efficient images by separating build and runtime dependencies, ensuring only necessary components are included in the final image. Read more in the [Multi-stage builds guide](#).

Although the Dockerfile is generated automatically, understanding its purpose and functionality is important. Reviewing the following example is highly recommended.

Dockerfile

```
# syntax=docker/dockerfile:1
# check=error=true

# This Dockerfile is designed for production, not development.
# docker build -t app .
# docker run -d -p 80:80 -e RAILS_MASTER_KEY=<value from config/master.key> .

# For a containerized dev environment, see Dev Containers: https://guides.rubyonrails.org/dev_containers.html

# Make sure RUBY_VERSION matches the Ruby version in .ruby-version
ARG RUBY_VERSION=3.3.6
FROM docker.io/library/ruby:$RUBY_VERSION-slim AS base

# Rails app lives here
WORKDIR /rails

# Install base packages
# Replace libpq-dev with sqlite3 if using SQLite, or libmysqlclient-dev if using MySQL
RUN apt-get update -qq && \
    apt-get install --no-install-recommends -y curl libjemalloc2 libvips libpq-dev \
    rm -rf /var/lib/apt/lists /var/cache/apt/archives

# Set production environment
ENV RAILS_ENV="production" \
```

```

BUNDLE_DEPLOYMENT="1" \
BUNDLE_PATH="/usr/local/bundle" \
BUNDLE_WITHOUT="development"

# Throw-away build stage to reduce size of final image
FROM base AS build

# Install packages needed to build gems
RUN apt-get update -qq && \
    apt-get install --no-install-recommends -y build-essential curl git pkg-config \
    rm -rf /var/lib/apt/lists /var/cache/apt/archives

# Install JavaScript dependencies and Node.js for asset compilation
#
# Uncomment the following lines if you are using NodeJS need to compile assets
#
# ARG NODE_VERSION=18.12.0
# ARG YARN_VERSION=1.22.19
# ENV PATH=/usr/local/node/bin:$PATH
# RUN curl -sL https://github.com/nodenv/node-build/archive/master.tar.gz | \
#     /tmp/node-build-master/bin/node-build "${NODE_VERSION}" /usr/local/node \
#     npm install -g yarn@$YARN_VERSION && \
#     npm install -g mjml && \
#     rm -rf /tmp/node-build-master

# Install application gems
COPY Gemfile Gemfile.lock ./
RUN bundle install && \
    rm -rf ~/.bundle/ "${BUNDLE_PATH}"/ruby/*/cache "${BUNDLE_PATH}"/ruby/*/lib \
    bundle exec bootsnap precompile --gemfile

# Install node modules
#
# Uncomment the following lines if you are using NodeJS need to compile assets
#
# COPY package.json yarn.lock ./
# RUN --mount=type=cache,id=yarn,target=/rails/.cache/yarn YARN_CACHE_FOLDER=/rails/.cache/yarn \
#     yarn install --frozen-lockfile

# Copy application code
COPY . .

```

```

# Precompile bootsnap code for faster boot times
RUN bundle exec bootsnap precompile app/ lib/

# Precompiling assets for production without requiring secret RAILS_MASTER_KEY
RUN SECRET_KEY_BASE_DUMMY=1 ./bin/rails assets:precompile

# Final stage for app image
FROM base

# Copy built artifacts: gems, application
COPY --from=build "${BUNDLE_PATH}" "${BUNDLE_PATH}"
COPY --from=build /rails /rails

# Run and own only the runtime files as a non-root user for security
RUN groupadd --system --gid 1000 rails && \
    useradd rails --uid 1000 --gid 1000 --create-home --shell /bin/bash && \
    chown -R rails:rails db log storage tmp
USER 1000:1000

# Entrypoint prepares the database.
ENTRYPOINT ["/rails/bin/docker-entrypoint"]

# Start server via Thruster by default, this can be overwritten at runtime
EXPOSE 80
CMD [ "./bin/thrust", "./bin/rails", "server" ]

```

The Dockerfile above assumes you are using Thruster together with Puma as an application server. In case you are using any other server, you can replace the last three lines with the following:

```

# Start the application server
EXPOSE 3000
CMD [ "./bin/rails", "server" ]

```

This Dockerfile uses a script at `./bin/docker-entrypoint` as the container's entrypoint. This script prepares the database and runs the application server. Below is an example of such a script.

docker-entrypoint

```
#!/bin/bash -e

# Enable jemalloc for reduced memory usage and latency.
if [ -z "${LD_PRELOAD+x}" ]; then
    LD_PRELOAD=$(find /usr/lib -name libjemalloc.so.2 -print -quit)
    export LD_PRELOAD
fi

# If running the rails server then create or migrate existing database
if [ "${@: -2:1}" == "./bin/rails" ] && [ "${@: -1:1}" == "server" ]; then
    ./bin/rails db:prepare
fi

exec "${@}"
```

Besides the two files above you will also need a `.dockerignore` file. This file is used to exclude files and directories from the context of the build. Below is an example of a `.dockerignore` file.

`.dockerignore`

Show more

The last optional file that you may want is the `compose.yaml` file, which is used by Docker Compose to define the services that make up the application. Since SQLite is being used as the database, there is no need to define a separate service for the database. The only service required is the Rails application itself.

`compose.yaml`

```
services:
  web:
    build: .
    environment:
      - RAILS_MASTER_KEY
    ports:
      - "3000:80"
```

You should now have the following files in your application folder:

- `.dockerignore`
- `compose.yaml`
- `Dockerfile`
- `bin/docker-entrypoint`

To learn more about the files, see the following:

- [Dockerfile](#)
- [.dockerignore](#)
- [compose.yaml](#)
- [docker-entrypoint](#)

Run the application

To run the application, run the following command in a terminal inside the application's directory.

```
$ RAILS_MASTER_KEY=<master_key_value> docker compose up --build
```

Open a browser and view the application at <http://localhost:3000> [↗](#). You should see a simple Ruby on Rails application.

In the terminal, press `ctrl + c` to stop the application.

Run the application in the background

You can run the application detached from the terminal by adding the `-d` option. Inside the `docker-ruby-on-rails` directory, run the following command in a terminal.

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

Open a browser and view the application at <http://localhost:3000> [↗](#).

You should see a simple Ruby on Rails application.

In the terminal, run the following command to stop the application.

```
$ docker compose down
```

For more information about Compose commands, see the [Compose CLI reference](#).

Summary

In this section, you learned how you can containerize and run your Ruby application using Docker.

Related information:

- [Docker Compose overview](#)

Next steps

In the next section, you'll learn how you can develop your application using containers.

[Use containers for Ruby on Rails development »](#)

Product offerings

Pricing

About us

Support

Contribute

Copyright © 2013-2025 Docker Inc. All rights reserved.



[Cookies Settings](#)

Theme: [Light](#)