Overview of Docker Compose

Estimated reading time: 5 minutes

> **Looking for Compose file reference?** Find the latest version here.

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. To learn more about all the features of Compose, see the list of features.

Compose works in all environments: production, staging, development, testing, as well as CI workflows. You can learn more about each case in Common Use Cases.

Using Compose is basically a three-step process:

1. Define your app's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.
3. Run `docker-compose up` and Compose starts and runs your entire app.

A `docker-compose.yml` looks like this:

```
version: '2.0'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
```

```
      - .:/code
      - logvolume01:/var/log
      links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

For more information about the Compose file, see the Compose file reference.

Compose has commands for managing the whole lifecycle of your application:

- Start, stop, and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

Compose documentation

- Installing Compose
- Getting started with Compose
- Get started with Django
- Get started with Rails
- Get started with WordPress
- Frequently asked questions
- Command line reference
- Compose file reference

Features

The features of Compose that make it effective are:

- Multiple isolated environments on a single host

- Preserve volume data when containers are created
- Only recreate containers that have changed
- Variables and moving a composition between environments

Multiple isolated environments on a single host

Compose uses a project name to isolate environments from each other. You can make use of this project name in several different contexts:

- on a dev host, to create multiple copies of a single environment, such as when you want to run a stable copy for each feature branch of a project
- on a CI server, to keep builds from interfering with each other, you can set the project name to a unique build number
- on a shared host or dev host, to prevent different projects, which may use the same service names, from interfering with each other

The default project name is the basename of the project directory. You can set a custom project name by using the `-p` command line option or the `COMPOSE_PROJECT_NAME` environment variable.

Preserve volume data when containers are created

Compose preserves all volumes used by your services. When `docker-compose up` runs, if it finds any containers from previous runs, it copies the volumes from the old container to the new container. This process ensures that any data you've created in volumes isn't lost.

If you use `docker-compose` on a Windows machine, see Environment variables and adjust the necessary environment variables for your specific needs.

Only recreate containers that have changed

Compose caches the configuration used to create a container. When you restart a service that has not changed, Compose re-uses the existing containers. Re-using containers means that you can make changes to your environment very quickly.

Variables and moving a composition between environments

Compose supports variables in the Compose file. You can use these variables to customize your composition for different environments, or different users. See Variable substitution for more details.

You can extend a Compose file using the `extends` field or by creating multiple Compose files. See extends for more details.

Common use cases

Compose can be used in many different ways. Some common use cases are outlined below.

Development environments

When you're developing software, the ability to run an application in an isolated environment and interact with it is crucial. The Compose command line tool can be used to create the environment and interact with it.

The Compose file provides a way to document and configure all of the application's service dependencies (databases, queues, caches, web service APIs, etc). Using the Compose command line tool you can create and start one or more containers for each dependency with a single command (`docker-compose up`).

Together, these features provide a convenient way for developers to get started on a project. Compose can reduce a multi-page "developer getting started guide" to a single machine readable Compose file and a few commands.

Automated testing environments

An important part of any Continuous Deployment or Continuous Integration process is the automated test suite. Automated end-to-end testing requires an environment in which to run tests. Compose provides a convenient way to create and destroy isolated testing environments for your test suite. By defining the full environment in a Compose file, you can create and destroy these environments in just a few commands:

```
$ docker-compose up -d
$ ./run_tests
$ docker-compose down
```

Single host deployments

Compose has traditionally been focused on development and testing workflows, but with each release we're making progress on more production-oriented features. You can use Compose to deploy to a remote Docker Engine. The Docker Engine may be a single instance provisioned with Docker Machine or an entire Docker Swarm cluster.

For details on using production-oriented features, see compose in production in this documentation.

Release notes

To see a detailed list of changes for past and current releases of Docker Compose, refer to the CHANGELOG.

Getting help

Docker Compose is under active development. If you need help, would like to contribute, or simply want to talk about the project with like-minded individuals, we have a number of open channels for communication.

- To report bugs or file feature requests: use the issue tracker on Github.
- To talk about the project with people in real time: join the `#docker-compose` channel on the Docker Community Slack.
- To contribute code or documentation changes: submit a pull request on Github.

documentation, docs, docker, compose, orchestration, containers