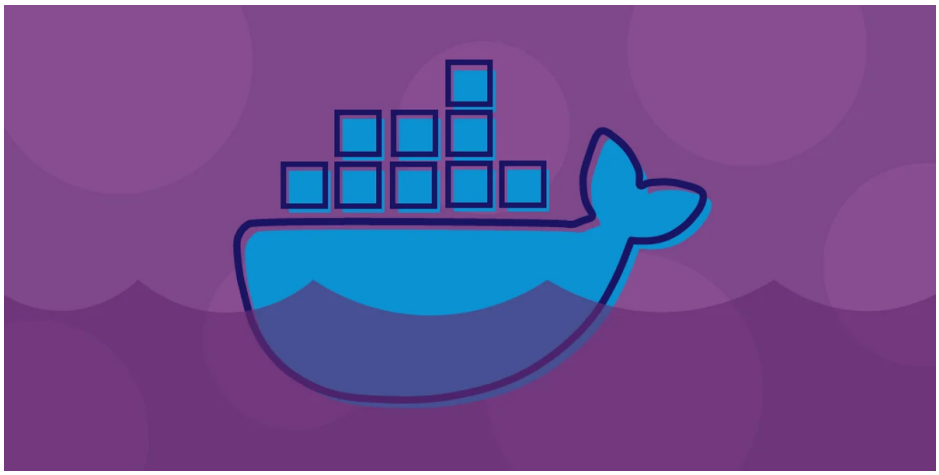


Docker Compose Tutorial: advanced Docker made simple

Jul 20, 2020 - 13 min read

Amanda Fawcett

Docker is an in-demand, DevOps technology sought after in the tech industry. This powerful tool allows you to set up and deploy applications using containers. With the consistent environment of Docker, the development lifecycle of applications is easy and seamless. Docker Compose, a more advanced Docker tool, can be used to simplify your workflow.

In this article, we will refresh your knowledge of Docker and show you how to get started with Docker Compose. To be most successful with this article, you should already have some experience with Docker.

> *If you're new to DevOps, check out our beginner's guide to Docker and Kubernetes (https://www.educative.io/blog/docker-kubernetes-beginners-guide) before proceeding with this tutorial.*

Today, we will go over:

- Brief refresher on Docker
- Fundamentals of Docker
- Getting started with Docker Compose
- What to learn next
- Wrapping up and resources

Learn advanced Docker and Docker Compose.

Learn the fundamentals of Docker with hands-on practice.

Working with Containers: Docker & Docker Compose (https://www.educative.io/courses/working-with-containers-docker-docker-compose)

Brief refresher on Docker

Docker is an open-source containerization tool used to simplify the creation and deployment of applications by using the concept of **containers**. Containers allow us to package all the parts of an application and deploy it as one entity.

This tool makes it easy for different developers to work on the same project in the same environment without any dependencies or OS issues. Docker is sort of like a virtual machine, but Docker enables applications to access the same Linux kernel.

Docker offers many advantages for developers and DevOps teams. Docker...

- is highly demanded by companies large and small
- offers isolation from the main system
- simplifies configuration
- provides access to thousands of configured images with Docker Hub
- supports many CI tools like Travis and Jenkins
- allows developers to focus just on writing code
- simplifies deployment management for operations teams

Docker is commonly used alongside **Kubernetes**, a powerful container management tool that automates the deployment of your Docker containers. While Docker is used to isolate, pack and ship your application into containers, Kubernetes is like the container scheduler for deploying and scaling the application.

The two technologies are designed to work together and make app deployment a breeze.

Fundamentals of Docker

Before diving into advanced Docker concepts, like Docker Compose, we want to make sure to refresh the fundamentals of Docker as a whole. Let's define and explore the basics of Docker.

Docker Architecture

The Docker Architecture is made of layers, as we will discuss below. The bottom layer is the physical server that we use to host virtual machines. This is the same as a traditional virtualization architecture.

The second layer is the Host OS, which is the base machine (i.e. Windows or Linux). Next, is the Docker Engine, which we use to run the operating system. Above that is are the Apps which run as Docker containers. Those Docker Objects are made up of images and containers.

Containers and images

The basic structure of Docker relies on **images** and **containers**. Think of images and containers as two different states of the same underlying concept. A container is like an object, and an image is like its class.

Think of a container as an isolated system that contains everything needed to run a certain application. It is an instance of an image that simulates the required environment. Below is an example command when we run a ubuntu Docker container:

```
docker run –i –t ubuntu /bin/bash
```

Images, on the other hand, are used to start-up containers. From running containers, we can get images, which can be composed together to form a system-agnostic way of packaging applications.

Images can be pre-built, retrieved from registries, created from already existing ones, or combined together via a common network.

Dockerfiles

Dockerfiles are how we containerize our application, or how we build a new container from an already pre-built image and add custom logic to start our application. From a Dockerfile, we use the Docker build command to create an image.

Think of a Dockerfile as a text document that contains the commands we call on the command line to build an image.

Below is an example of a Dockerfile:

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD [ "python", "./your-daemon-or-script.py" ]
```

Layers

A Dockerfile works in layers. These are the building blocks of Docker. The first layer starts with the `FROMkeyword` and defines which pre-built image we will use build an image. We can then define user permissions and startup scripts.

In Docker, a container is an image with a readable layer build on top of a read-only layers. These layer are called intermediate images, and they are generated when we execute the commands in our Dockerfile during the build stage.

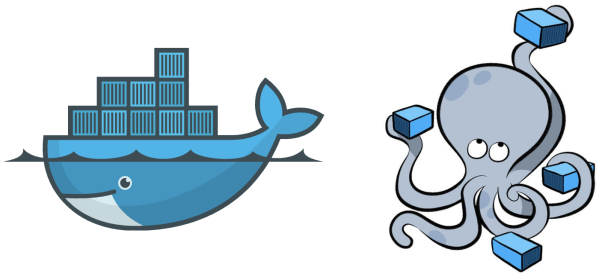
Docker Hub

Docker Hub is a Docker Registry that provides unlimited storage of public images and paid plans for hosting private images. A public image may be accessed by anyone. You can publish and access images on Docker Hub once you make an account.

You can do this using the docker build command or docker tags to generate an image ID. You can publish an image on Docker Hub using the following commands:

```
docker login
docker push learnbook/webserver
```

That was just an overview of the fundamentals of Docker before we move onto move advanced concepts. Keep in mind that there's a lot more to Docker than what we discussed above.

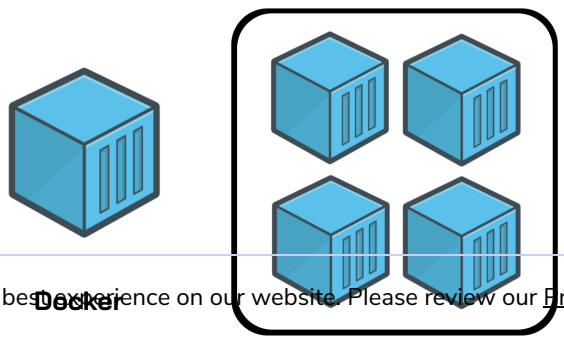


Getting started with Docker Compose

Now for the advanced stuff. **Docker Compose** is a Docker tool used to define and run multi-container applications. With Compose, you use a `YAML` file to configure your application's services and create all the app's services from that configuration.

Think of `docker-compose` as an **automated multi-container workflow.** Compose is an excellent tool for development, testing, CI workflows, and staging environments. According to the Docker documentation, the most popular features of Docker Compose 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
- Orchestrate multiple containers that work together



> ***Enjoying the article?*** *Scroll down to sign up (https://www.educative.io/blog/blog-newsletter-annoucement) for our free, bi-monthly newsletter.*

How to use and install Docker Compose

Compose uses the Docker Engine, so you'll need to have the Docker Engine installed on your device. You can run Compose on Windows, Mac, and 64-bit Linux. Installing Docker Compose is actually quite easy.

On desktop systems, such as Docker Desktop for Mac and Windows, Docker Compose is already included. No additional steps are needed. On Linux systems, you'll need to:

1. Install the Docker Engine
2. Run the following command to download Docker Compose

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.26.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

3. Apply permissions to the binary, like so:

```
sudo chmod +x /usr/local/bin/docker-compose
```

4. Test the installation to check it worked properly

```
$ docker-compose --version
docker-compose version 1.26.2, build 1110ad01
```

Regardless of how you chose to install it, once you have Docker Compose downloaded and running properly, you can start using it with your Dockerfiles. This process requires three basic steps:

1. Define your app's environment using a Dockerfile. This way, it can be reproduced.
2. Define the services for your app in a `docker-compose.yml` file. This way, they can run in an isolated environment.
3. Run `docker-compose` to start your app.

You can easily add Docker Compose to a pre-existing project. If you already have some Dockerfiles, add Docker Compose files by opening the Command Palette. Use the **Docker: Docker Compose**

Files to the Workspace command, and, when promoted, choose the Dockerfiles you want to include.

You can also add Docker Compose files to your workspace when you add a Dockerfile. Similarly, open the Command Palette and use the **Docker: Add Docker Files to Workspace** command.

You'll then be asked if you want to add any Docker Compose files. In both cases, Compose extension will add the `docker-compose.yml` file to your workspace.

Docker Compose file structure

Now that we know how to download Docker Compose, we need to understand how Compose files work. It's actually simpler than it seems. In short, Docker Compose files work by applying mutiple commands that are declared within a single `docker-compose.yml` configuration file.

The basic structure of a Docker Compose YAML file looks like this:

```
version: 'X'

services:
  web:
    build: .
    ports:
     - "5000:5000"
    volumes:
     - .:/code
  redis:
    image: redis
```

Now, let's look at real-world example of a Docker Compose file and break it down step-by-step to understand all of this better. Note that all the clauses and keywords in this example are commonly used keywords and industry standard.

With just these, you can start a development workflow. There are some more advanced keywords that you can use in production, but for now, let's just get started with the necessary clauses.

```
      - "/usercode/:/code"
    # Link database container to app container
    # for reachability.
    links:
      - "database:backenddb"
```

```
21
22   database:
23
24     # image to fetch from docker hub
25     image: mysql/mysql-server:5.7
26
27     # Environment variables for startup script
28     # container will use these variables
29     # to start the container with these define variables.
30     environment:
31       - "MYSQL_ROOT_PASSWORD=root"
32       - "MYSQL_USER=testuser"
33       - "MYSQL_PASSWORD=admin123"
34       - "MYSQL_DATABASE=backend"
35     # Mount init.sql file to automatically run
36     # and create tables for us.
37     # everything in docker-entrypoint-initdb.d folder
38     # is executed as soon as container is up nd running.
39     volumes:
40       - "/usercode/db/init.sql:/docker-entrypoint-initdb.d/init.sq
41
```

- **`version '3'`:** This denotes that we are using version 3 of Docker Compose, and Docker will provide the appropriate features. At the time of writing this article, version 3.7 is latest version of Compose.
- **`services`:** This section defines all the different containers we will create. In our example, we have two services, web and database.
- **`web`:** This is the name of our Flask app service. Docker Compose will create containers with the name we provide.
- **`build`:** This specifies the location of our Dockerfile, and `.` represents the directory where the `docker-compose.yml` file is located.
- **`ports`:** This is used to map the container's ports to the host machine.
- **`volumes`:** This is just like the `-v` option for mounting disks in Docker. In this example, we attach our code files directory to the containers' `./code` directory. This way, we won't have to rebuild the images if changes are made.
- **`links`:** This will link one service to another. For the bridge network, we must specify which container should be accessible to which container using links.
- **`image`:** If we don't have a Dockerfile and want to run a service using a pre-built image, we specify the image location using the `image` clause. Compose will fork a container from that image.

- **`environment`:** The clause allows us to set up an environment variable in the container. This is the same as the `-e` argument in Docker when running a container.

Congrats! Now you know a bit about Docker Compose and the necessary parts you'll need to get started with your workflow.

Docker Compose commands

Now that we know how to create a `docker-compose` file, let's go over the most common Docker Compose commands that we can use with our files. Keep in mind that we will only be discussing the most frequently-used commands.

`docker-compose`: Every Compose command starts with this command. You can also use `docker-compose <command> --help` to provide additional information about arguments and implementation details.

```
$ docker-compose --help
Define and run multi-container applications with Docker.
```

`docker-compose build`: This command builds images in the `docker-compose.yml` file. The job of the `build` command is to get the images ready to create containers, so if a service is using the prebuilt image, it will skip this service.

```
$ docker-compose build
database uses an image, skipping
Building web
Step 1/11 : FROM python:3.9-rc-buster
 ---> 2e0edf7d3a8a
Step 2/11 : RUN apt-get update && apt-get install -y docke
r.io
```

`docker-compose images`: This command will list the images you've built using the current `docker-compose` file.

```
$ docker-compose images
          Container                  Repository        Tag
       Image Id       Size
-------------------------------------------------------------
----------------------------
7001788f31a9_docker_database_1   mysql/mysql-server   5.7
      2a6c84ecfcb2   333.9 MB
docker_database_1                mysql/mysql-server   5.7
      2a6c84ecfcb2   333.9 MB
docker_web_1                     <none>               <non
e>   d986d824dae4   953 MB
```

`docker-compose stop`: This command stops the running containers of specified services.

```
$ docker-compose stop
Stopping docker_web_1      ... done
Stopping docker_database_1 ... done
```

`docker-compose run`: This is similar to the `docker run` command. It will create containers from images built for the services mentioned in the compose file.

```
$ docker-compose run web
Starting 7001788f31a9_docker_database_1 ... done
 * Serving Flask app "app.py" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 116-917-688
```

`docker-compose up` : This command does the work of the `docker-compose build` and `docker-compose run` commands. It builds the images if they are not located locally and starts the containers. If images are already built, it will fork the container directly.

```
$ docker-compose up
Creating docker_database_1 ... done
Creating docker_web_1      ... done
Attaching to docker_database_1, docker_web_1
```

`docker-compose ps` : This command list all the containers in the current `docker-compose` file. They can then either be running or stopped.

```
$ docker-compose ps
      Name                 Command             State
               Ports
------------------------------------------------------------
-----------------------
docker_database_1   /entrypoint.sh mysqld   Up (healthy)
   3306/tcp, 33060/tcp
docker_web_1        flask run               Up
             0.0.0.0:5000->5000/tcp

$ docker-compose ps
      Name                 Command          State    Ports
------------------------------------------------------------
docker_database_1   /entrypoint.sh mysqld   Exit 0
docker_web_1        flask run               Exit 0
```

`docker-compose down` : This command is similar to the `docker system prune` command. However, in Compose, it stops all the services and cleans up the containers, networks, and images.

```
$ docker-compose down
Removing docker_web_1      ... done
Removing docker_database_1 ... done
Removing network docker_default
(django-tuts) Venkateshs-MacBook-Air:Docker venkateshachin
talwar$ docker-compose images
Container   Repository   Tag   Image Id   Size
------------------------------------------------
(django-tuts) Venkateshs-MacBook-Air:Docker venkateshachin
talwar$ docker-compose ps
Name   Command   State   Ports
------------------------------
```

Congrats! You've now learned most of the basic commands for Docker Compose. Why stop there? Check out the documentation (https://docs.docker.com/compose/reference/) of other commands and keep learning!

What to learn next

I hope this has familiarized you with Docker Compose and all it has to offer. There's still a lot to explore and learn to be a true Docker Compose master. Once you are comfortable making `docker-compose` files and working with the necessary commands, you can move onto the following advancements:

- Working with multiple Dockerfiles in Compose (common for microservices (https://www.educative.io/blog/microservices-architecture-tutorial-all-you-need-to-get-started))
- Docker Compose environment variables (`.env` files)
- Docker Swarm (for scaling and monitoring clusters)
- Automated deployments with Docker Stack
- and more

Educative's advanced Docker course **Working with Containers: Docker & Docker Compose (https://www.educative.io/courses/working-with-containers-docker-docker-compose)** is an ideal place to learn these concepts and beyond. Not only will you get a refresher on Docker fundamentals, but you'll also progress to advanced concepts like connecting to a database container and hands-on practice with Docker Compose.

At the end, you'll even learn how to monitor clusters and scale Docker services with Swarm. Jumpstart your career and become an in-demand Docker developer!

Continue reading

- Kubernetes Tutorial: Getting Started with Container Orchestration (https://www.educative.io/blog/kubernetes-tutorial)
- Getting started with Docker and Kubernetes: a beginners guide (https://www.educative.io/blog/docker-kubernetes-beginners-guide)

- Why (and when) you should use Kubernetes (https://www.educative.io/blog/why-and-when-you-should-use-kubernetes)

WRITTEN BY

Amanda Fawcett

Join a community of 625,000 monthly readers. A free, bi-monthly email with a roundup of Educative's top articles and coding tips.

Full Name

E-mail

Subscribe

More from Educative:

The 11 best C++ IDEs (and code editors) for 2022

When choosing the best C++ IDE for your needs, where should you start? This complete list of the be...

Zach Milkis
Sep 2 · 2021
(/blog/best-cpp-ides-code-editors)

NumPy matrix multiplication: Get started in 5 minutes

Matrix multiplication can help give us quick approximations of very complicated calculations. In...

Erin Schaffer
Sep 3 · 2021
(/blog/numpy-matrix-multiplication)

Coding for social good: 7 companies you should know

about

Today, we'll explore seven companies where you can code for social good.

Erin Schaffer
Aug 27 · 2021
(/blog/coding-for-social-good)

Become an Author

(/authors)

Become an Affiliate

(/affiliate)

LEGAL

Privacy Policy

(/privacy)

Terms of Service

(/terms)

Business Terms of Service

(/enterprise-terms)

MORE

Our Team

(/team)

Careers (//jobs.lever.co/educative) Hiring

For Bootcamps (//try.educative.io/bootcamps)

Blog for Business

(/blog/enterprise)

Quality Commitment

(/quality)

FAQ

(/courses/educative-faq)

Press

(/press)

Contact Us

(/contactUs)

(//linkedin.com/company/educative-inc/) (//twitter.com/educativeinc) (//www.youtube.com/channel/UCT_8FqzTlr2Q1BOtvX_DPPw/?sub_confirmation=1) (//educativesess

educativeinc)