How to Set up Docker Compose: Step-By-Step Tutorial

KaaIoT Technologies, LLC

.

Follow

9 min read

.

Mar 19, 2024

--

Listen

Share

Docker Compose is like a helpful assistant for organizing and running complex applications with multiple containers. By using a straightforward YAML file, you can easily define all the services, networks, and storage needed for your containers to work smoothly together.
Whether you're handling microservices or interconnected applications, Docker Compose simplifies the setup and upkeep, fostering teamwork and efficient Docker environment management.
For developers aiming to streamline their containerized app development, mastering Docker Compose setup is key. This guide will lead you through the installation and configuration steps, empowering you to efficiently manage multi-container applications.

Docker Compose use cases

Image source: kaaiot.com

Prototyping

Prototyping with Docker Compose offers developers a streamlined approach to testing and validating new ideas. Developers can swiftly refine ideas into tangible implementations by specifying containerized services and their interactions within a Docker Compose file. This enables a fast cycle of testing and refining, accelerating the journey from conceptualization to actualization. This isolated environment allows for safe experimentation without risking the stability of production systems.

Also, Docker Compose enables efficient resource utilization, making it cost-effective to prototype on local machines or in cloud environments. Collaborative development is facilitated through the easy sharing of Docker Compose files among team members, promoting teamwork and knowledge sharing. These prototypes can also be integrated with existing systems for comprehensive integration testing, ensuring compatibility and functionality.

Local development

Local development with Docker Compose streamlines the setup of development environments by encapsulating all necessary dependencies and configurations within a single YAML file. It allows developers to replicate production-like environments on their local machines effortlessly. With Docker Compose, developers can launch and manage multi-container applications locally with a single command, facilitating rapid development iterations and debugging processes.

Additionally, Docker Compose provides a consistent development environment across teams, ensuring that all developers work with identical setups, minimizing compatibility issues, and streamlining collaboration.

Demo environments

Demo environments showcase applications to stakeholders or potential users. Docker Compose simplifies the creation and management of demo environments by encapsulating the entire application stack, including services, networks, and volumes, into a single YAML file. It enables developers to swiftly create demo environments on their local computers or cloud servers, requiring little effort.

With Docker Compose, developers can ensure demo environments accurately reflect production configurations, enabling more effective demonstrations and evaluations of the application's features and functionalities.

Additionally, Docker Compose facilitates the sharing of demo environments among team members or with external parties, promoting collaboration and feedback exchange.

Microservices developing

Microservices development often involves working with multiple interconnected components, which can make local development and testing challenging. Docker Compose simplifies this process by allowing developers to define and manage all the necessary services in a single configuration file. It streamlines the setup of

microservice architectures and facilitates efficient local development and testing workflows.

5 Key benefits of Docker Compose

Image source: kaaiot.com

Simple management

Docker Compose makes managing multi-container apps a breeze by simplifying the setup in a single YAML file. Rather than juggling individual container configurations and dependencies, you can define all services, networks, and storage in one go. This centralized setup streamlines management, encapsulating your entire app stack.
Using Docker Compose, you can easily handle your app with straightforward commands, eliminating the hassle of managing each container separately.

Development efficiency

Docker Compose brings significant benefits to development efficiency. By letting developers define their app setup in a single YAML file, Docker Compose simplifies local environment configuration. With just one command, developers can swiftly launch their entire app stack, bypassing the need to configure each part individually.
This streamlined process cuts down on the time and energy spent on setting up and tearing down dev environments, allowing developers to concentrate on coding rather than infrastructure management.
Furthermore, Docker Compose ensures that all developers work with consistent setups across environments, promoting a standardized development approach. It helps prevent discrepancies between setups and minimizes the chances of bugs or compatibility issues stemming from environment differences.

Security of internal container network

In Docker Compose, all containers specified in the compose file are connected to the same internal network, shielding them from unauthorized access. It not only enhances security but also streamlines network management for multi-container applications.

Version control

Version control is crucial in software development, and Docker Compose supports it effectively by allowing developers to store and manage their application stack configurations in version-controlled YAML files. This ensures that changes to the development environment are tracked and documented, facilitating collaboration and reproducibility across different environments and team members.

Host isolation

Isolation is paramount in containerized environments, and Docker Compose provides a level of isolation by encapsulating each application component within its container. This prevents conflicts between dependencies and ensures that changes made to one component do not affect others. Docker Compose allows developers to define network and volume configurations, further enhancing isolation and security.

Read also: Docker Compose vs. Kubernetes.

Docker Compose Installation and setup for Mac

To start using Docker Compose swiftly, opt for Docker Desktop. It's bundled with Docker Engine, Docker CLI, and Docker Compose. Available for Windows, Linux, and MacOS, you can download it directly from Docker's official website for a hassle-free setup.

1. First, make sure you have Docker Desktop installed on your Mac. Download Docker Desktop from the official Docker website and follow the installation instructions.
2. Once Docker Desktop is installed, open a terminal window on your Mac.
3. To install Docker Compose, you can use Homebrew, a package manager for macOS: ***brew install docker-compose***

4. Once the installation is complete, you can verify that Docker Compose was installed correctly by running: ***docker-compose — version***

You've successfully installed Docker Compose on your Mac using Homebrew. You're now ready to utilize Docker Compose to craft and execute multi-container Docker applications.

Docker Compose file

YAML, a versatile data serialization language, finds utility across diverse programming scenarios like internet communications, object persistence, and configuration file composition. Its syntax allows for the construction of hierarchical structures and key-value pairs, with indentation visually depicting relationships between components. Notably, YAML permits inline comments, enhancing its usability for describing Compose files, unlike JSON, which lacks this feature.

To create a docker-compose.yml file in your first project directory, you can follow these steps:

1. Create a project directory. Open Terminal and navigate to the location where you want to create your Docker Compose project directory. Use the ***mkdir*** command to create a new directory for your project

mkdir my_project

2. Navigate to the project directory
3. Create a new file named ***docker-compose.yml***.
4. Open the ***docker-compose.yml*** file in your text editor.
5. Define your services, volumes, networks, and other configurations using YAML syntax.
6. Create additional files

Depending on your project's requirements, you may need to create additional files, such as Dockerfiles, for building custom images or environment configuration files. Verify the project structure

Confirm that your project directory structure looks like this:

my_project/

■■■ docker-compose.yml

■■■ Dockerfile (if required)

■■■ scripts/

■ ■■■ setup.sh

■■■ src/

■ ■■■ app.py

■■■ data/

■■■ db.sqlite

Here's a basic example of a docker-compose.yml file:

version: '3.8'

services:

web:

image: nginx:latest

ports:

- "80:80"

volumes:

- ./html:/usr/share/nginx/html

This example defines a single service named "web" using the latest Nginx image. It maps port 80 on the host to port 80 on the container and mounts the HTML directory from the current project directory to /usr/share/nginx/html in the container.

Once you've defined your services and configurations in the docker-compose.yml file, you can proceed to use Docker Compose commands to manage your multi-container applications.

Another example of a docker-compose.yml file for a simple web application:

version: '3.8'

services:

web:

image: nginx:latest

ports:

- "8080:80"

volumes:

- ./html:/usr/share/nginx/html

networks:

```
    - mynetwork

    networks:

    mynetwork:
```

“version”: Specifies the version of the Docker Compose file format being used.

“services”: Defines the services that make up the application.

“web”: Name of the service.

“image”: Specifies the Docker image to use for this service (here, it’s the latest version of Nginx).

“ports”: Maps the host machine’s port 8080 to the container’s port 80, allowing access to the web server.

“volumes”: Mounts the local ./html directory to the container’s /usr/share/nginx/html directory, allowing the webserver to serve files from the local directory.

“networks”: Associates the service with a specific network.

“networks”: Defines the network configuration for the services.

mynetwork: Name of the network.

This configuration defines a service called “web” using the Nginx Docker image. It exposes port 8080 on the host machine, forwards requests to port 80 on the container, mounts the local ./html directory to serve web content, and connects the service to a custom network named “mynetwork.”

Adjust the structure according to your project’s needs.

Once you’ve set up the directory structure and defined your docker-compose.yml file, you can start building and running your Docker Compose project by executing the appropriate Docker Compose commands within the project directory.

Docker Compose popular commands

1. docker-compose up

When you run the command ***docker-compose up*** in the Mac terminal within your Docker Compose project directory, Docker Compose will start the defined services and containers according to the configurations specified in the docker-compose.yml file. Here’s an example of the output you might see in the terminal:

```
    Creating network “my_project_default” with the default driver

    Creating my_project_web_1 … done

    Creating my_project_db_1 … done

    Attaching to my_project_web_1, my_project_db_1

    web_1 | Starting web server…

    db_1 | Initializing database…

    web_1 | Web server started on port 8080

    db_1 | Database initialized successfully
```

In this example, Docker Compose creates a default network for the project.

It starts with the defined services web and db, which correspond to containers defined in the docker-compose.yml file.

Output from each service/container is displayed, such as log messages or startup information.

The containers web_1 and db_1 are created and started with the services’ defined configurations applied.

You’ll see additional output depending on the specifics of your Docker Compose project and the services it defines.

2. docker-compose down

When you run the command ***docker-compose down*** in the Mac terminal within your Docker Compose project directory, the tool will stop and remove all containers, networks, and volumes created for the project. Here’s an example of the output you might see in the terminal:

```
    Stopping my_project_web_1 … done

    Stopping my_project_db_1 … done
```

Removing my_project_web_1 … done

Removing my_project_db_1 … done

Removing network my_project_default

Docker Compose stops and removes the containers my_project_web_1 and my_project_db_1.

It also removes the default network my_project_default that was created for the project.

You'll see this output as confirmation that the containers and associated resources have been successfully stopped and removed.

3. docker-compose build

When you run the command ***docker-compose build*** in the Mac terminal within your Docker Compose project directory, the tool will build the images defined in your docker-compose.yml file. Here's an example of the output you might see in the terminal:

Building web

Step 1/4 : FROM python:3.8-alpine

3.8-alpine: Pulling from library/python

…

Step 4/4 : CMD ["python", "app.py"]

Successfully built 1234567890

Docker Compose builds the image for the service named web, as specified in the docker-compose.yml file.

It pulls the base image python:3.8-alpine from the Docker Hub.

It executes the steps defined in the Dockerfile for the web service, such as installing dependencies and setting the command to run the application.

Finally, it confirms that the image has been successfully built and assigns it a unique identifier (e.g., 1234567890).

4. docker-compose start and docker-compose stop

When you run the command ***docker-compose start*** in the Mac terminal within your Docker Compose project directory, Docker Compose will start the containers defined in your ***docker-compose.yml*** file. Similarly, running ***docker-compose stop*** will stop the running containers. Here's an example of the output you might see in the terminal:

For docker-compose start:

Starting myapp_web_1 … done

Starting myapp_db_1 … done

Starting myapp_db_1 … done

For docker-compose stop:

Stopping myapp_web_1 … done

Stopping myapp_db_1 … done

Docker Compose starts or stops the containers named myapp_web_1 and myapp_db_1, as defined in the docker-compose.yml file.

The 'done' message indicates that the operation was successful.

Conclusion

Docker Compose offers a streamlined solution for managing multi-container Docker applications, enabling developers to define and execute complex application stacks with ease. Its simple installation process and intuitive setup make it a valuable tool for improving development efficiency, promoting consistency across environments, and facilitating collaboration among team members.

With Docker Compose, developers can accelerate the deployment of applications, enhance version control practices, and streamline the development lifecycle, ultimately leading to faster innovation and the delivery of high-quality software solutions.