

Table of Contents

- [Table of Contents](#)
 - [Docker Compose](#)
 - [Installing Compose on Linux](#)
 - [Compose V2](#)
 - [Compose files](#)
 - [Compose files structure](#)
 - [docker run vs docker-compose](#)
 - [Environment Variables in Compose](#)
 - [Deploying an app with Compose - Step 1: Setup - Step 3: Define services in a Compose file - Step 4: Build and run your app with Compose](#)
 - [Reference](#)
-

Docker Compose

- 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.
- Using a Single command we can **create and start** all the services from your configuration.
- 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
- 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** to start and runs your entire application.

--

services in compose will launch containers

You define multi-container (multi-service) apps in a YAML file, pass the YAML file to the docker compose binary, and Compose deploys it via the Docker Engine API.

Installing Compose on Linux

Compose V2

- The new Compose V2, which supports the **compose** command as part of the Docker CLI, is now available.
- **docker-compose** was previously used in the Original Docker Compose Setup.
- **docker compose** will be used in Docker Compose V2 and Compose V2 will be included with the latest version of the Docker CLI.
- Installing Docker Compose on Linux involves:
 - Download the binary using the **curl** command. Then you make it executable using **chmod**.
 - For Docker Compose to work on Linux, you'll need a working version of the **Docker Engine**.

--

- You can install Compose V2 by downloading the appropriate binary for your system from the release page and copying it into **\$HOME/.docker/cli-plugins** as **docker-compose**
- Run the following command to download the current stable release of Docker Compose, The following command will download version **2.14.2** of Docker Compose. Replace the **2.2.2** in the URL with the version you want to install. [Compose Version Reference](#)

```
mkdir -p ~/.docker/cli-plugins/
curl -SL https://github.com/docker/compose/releases/download/v2.2.2/docker-
compose-linux-x86_64 -o ~/.docker/cli-plugins/docker-compose
```

- This command installs Compose V2 for the active user under **\$HOME** directory.
- To install Docker Compose for all users on your system, replace **~/.docker/cli-plugins** with **/usr/local/lib/docker/cli-plugins**
- Apply executable permissions to the binary:

```
chmod +x ~/.docker/cli-plugins/docker-compose
```

--

- Test the installation and check **docker compose** version

```
docker compose version
# Docker Compose version v2.2.2
```

```
[ec2-user@ip-172-31-0-125 ~]$ curl -SL https://github.com/docker/compose/releases/download/v2.2.2/docker-compose-linux-x86_64
-o ~/.docker/cli-plugins/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 664    100 664    0    0  2432      0  --:--:-- --:--:-- --:--:-- 2432
100 23.5M  100 23.5M    0    0 6451k      0  0:00:03 0:00:03 --:--:-- 10.4M
[ec2-user@ip-172-31-0-125 ~]$ chmod +x ~/.docker/cli-plugins/docker-compose
[ec2-user@ip-172-31-0-125 ~]$ docker compose version
Docker Compose version v2.2.2
[ec2-user@ip-172-31-0-125 ~]$
```

Compose files structure

- Compose uses YAML files for definition
- The Compose file is a YAML file defining **services**, **networks** and **volumes**. The default path for a Compose file is `./docker-compose.yml`. However, you can use the `-f` flag to specify custom filenames.
- It is used to define **multi-service** applications.

--

docker run vs docker-compose

- Docker run command to start a container looks like :

```
docker run -d --name=nginx-run -p 8080:80 nginx
```

- A simple Docker Compose file for above run command looks like : **docker-compose.yml**.
- **docker-compose.yml**

```
version: '3'
services:
  web:
    container_name: nginx-compose
    image: nginx
    ports:
      - "8888:80"
```

- Commands to run the above docker compose file:

```
# Start the services specified inside the docker-compose.yml file and -d option is
used to run containers in the background
docker compose up -d
docker compose up -d -f docker-compose-nginx.yml
# Stop the services specified inside the docker-compose.yml file
docker compose down
docker compose down -f docker-compose-nginx.yml
```

--

Environment Variables in Compose

- Below is the content of the **docker-compose.yml** file which will run **mysql** database inside a container.

```
version: '3'
services:
  db:
    image: mysql:5.7
    container_name: mysql
    ports:
      - "3306:3306"
    environment:
      - MYSQL_ROOT_PASSWORD=12345678
      - AWS_ACCESS_ID=AWS_ACCESS_ID_VALUE
```

- execute **docker compose up -d** command to start the above services in compose file.
- Validate the environment variables:

```
docker inspect mysql
```

- Use **docker compose logs** to view logs.

--

- To connect interactively to the container shell, use:

```
docker ps
docker exec -it mysql bash
#Execute below commands in the Container:
printenv
#To connect to mysql database inside the container and enter the Password that is
set in the environment variable.
mysql -u root -p

#Run basic Mysql Commands
show databases;
use information_schema;
show tables;
```

- To use all the environment variables provided in a file:
- Use **env_file: variables.env** in the **docker-compose.yml** file instead of **environment**:
- Here we need to create **variables.env** file in the same directory of **docker-compose.yml** file with environment variables values.

--

- Environment file **variables.env** will be:

```
SDLC_ENVIRONMENT=dev
MYSQL_ROOT_PASSWORD=12345678
```

Updated **docker-compose.yml** will be:

```
version: '3'
services:
  db:
    image: mysql:5.7
    container_name: mysql
    ports:
      - "3306:3306"
    env_file:
      - variables.env
```

- After executing similar commands as above, the environment variables can be setup from a file inside the container.

```
docker compose up -d
# Login inside the docker container and validate the environment variables set
from this file.

docker compose down
```

Deploying an app with Compose

- Here, lets build a simple Python Web Application running on Docker Compose.
- The application uses the Flask framework and maintains a hit counter in Redis.

Step 1: Setup

- Code Base

```
git clone https://github.com/aws-containers/demo-app-for-docker-compose.git
cd demo-app-for-docker-compose/application
```

```
[ec2-user@ip-172-31-0-125 demo-app-for-docker-compose]$ tree application/
application/
├── docker-compose.yml
└── frontend
    ├── Dockerfile
    ├── myweb
    │   ├── app.py
    │   ├── static
    │   │   ├── blue.png
    │   │   ├── green.png
    │   │   └── style.css
    └── templates
```

```
├── health.html
├── index.html
└── requirements.txt
```

4 directories, 9 files

- The below **Dockerfile** that builds a Docker image. The application code and required python packages are installed in the Docker Image.
- `application/frontend/Dockerfile`

```
FROM public.ecr.aws/docker/library/python:3.8-slim

WORKDIR /app

ADD requirements.txt /app/requirements.txt

RUN apt-get update && \
    apt-get install --no-install-recommends curl -y && \
    rm -rf /var/lib/apt/lists/* && \
    pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt

COPY ./myweb /app/

EXPOSE 80

HEALTHCHECK --interval=30s --timeout=5s \
    CMD curl -sf http://localhost/health || exit 1

ENTRYPOINT ["python"]

CMD ["app.py"]
```

This tells Docker to:

- Build an image starting with the **public.ecr.aws/docker/library/python:3.8-slim** image.
- Set the working directory to `/app`.
- Add the requirements.txt file inside the `/app` directory in the image and install the pip packages.
- Copy application code directory **myweb**.
- Set the ENTRYPOINT to execute the program file **app.py**.

Step 3: Define services in a Compose file

Create a file called `docker-compose.yml` in your project directory and paste the following:

- `application/docker-compose.yml`

```

services:
  frontend:
    image: ${IMAGE_URI:-frontend}:${IMAGE_TAG:-latest}
    build: ./frontend
    environment:
      REDIS_URL: "backend"
    networks:
      - demoapp
    ports:
      - 80:80

  backend:
    image: public.ecr.aws/docker/library/redis:6.2
    volumes:
      - redisdata:/data
    networks:
      - demoapp

volumes:
  redisdata:

networks:
  demoapp:

```

This Compose file defines two services: **frontend** and **backend**.

- **Web service** The **frontend** service uses an image that's built from the **Dockerfile** in the current directory. It then binds the container and the host machine to the exposed port, **80**. This example service uses the default port for the Flask web server, **80**.
- **Redis service** The **backend** service uses a public **Redis** image pulled from the Public ECR registry.

Step 4: Build and run your app with Compose

1. From your project directory, start up your application by running **docker compose up**.

```

cd application/

docker compose up -d

WARN[0000] The "AWS_ECS_CLUSTER" variable is not set. Defaulting to a blank
string.
WARN[0000] The "AWS_ELB" variable is not set. Defaulting to a blank string.
WARN[0000] The "AWS_VPC" variable is not set. Defaulting to a blank string.
[+] Running 3/3
  :: Network application_demoapp      Created
0.1s
  :: Container application-frontend-1 Started
1.5s
  :: Container application-backend-1  Started

```

1.5s

```
[ec2-user@ip-172-31-0-125 application]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND
CREATED	STATUS	PORTS
	NAMES	
d55d7feb6a3d	public.ecr.aws/docker/library/redis:6.2	"docker-entrypoint.s..."
36 minutes ago	Up About a minute	6379/tcp
	application-backend-1	
42238d43d76d	frontend:latest	"python app.py"
43 minutes ago	Up About a minute (healthy)	0.0.0.0:80->80/t
cp, ::80->80/tcp	application-frontend-1	

- Here, Compose pulls a Redis image, builds an image for your application code, and starts the services you defined.
- Enter `http://HOST-IP:8000/` in a browser to see the application running.

▲ Not secure | 13.233.146.76



Welcome to the Demo App!



No of Clicks Today: 112

Served by Container 42238d43d76d

Previous Days Results

Date	Clicks
2022-12-26	112

- View Docker Compose Service Logs

```
docker compose logs frontend
docker compose logs backend
```

Reference

- Get started with Docker Compose : <https://docs.docker.com/compose/gettingstarted/>