DigitalOcean

Subscribe

How To Install Docker Compose on Debian 9

0

Posted September 6, 2018 5.1k DOCKER DEBIAN DEBIAN 9

By: Kathleen Juell By: Melissa Anderson

Introduction

Docker is a great tool for automating the deployment of Linux applications inside software containers, but to take full advantage of its potential each component of an application should run in its own individual container. For complex applications with a lot of components, orchestrating all the containers to start up, communicate, and shut down together can quickly become unwieldy.

The Docker community came up with a popular solution called Fig, which allowed you to use a single YAML file to orchestrate all of your Docker containers and configurations. This became so popular that the Docker team decided to make Docker Compose based on the Fig source, which is now deprecated. Docker Compose makes it easier for users to orchestrate the processes of Docker containers, including starting up, shutting down, and setting up intra-container linking and volumes.

In this tutorial, we'll show you how to install the latest version of Docker Compose to help you manage multi-container applications on a Debian 9 server.

Prerequisites

To follow this article, you will need:

- A Debian 9 server and a non-root user with sudo privileges. This initial server setup with Debian 9 tutorial explains how to set this up.
- Docker installed with the instructions from **Step 1** and **Step 2** of How To Install and Use Docker on Debian 9

Note: Even though the Prerequisites give instructions for installing Docker on Debian 9, the `docker` commands in this article should work on other operating systems as long as Docker is installed.

Step 1 — Installing Docker Compose

Although we can install Docker Compose from the official Debian repositories, it is several minor versions behind the latest release, so we'll install it from Docker's GitHub repository. The command below is slightly different than the one you'll find on the Releases page. By using the `-o` flag to specify the output file first rather than redirecting the output, this syntax avoids running into a permission denied error caused when using `sudo`.

We'll check the current release and, if necessary, update it in the command below:

```
$ sudo curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-compose-`
```

Next we'll set the permissions:

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

Then we'll verify that the installation was successful by checking the version:

```
$ docker-compose --version
```

This will print out the version we installed:

```
Output
docker-compose version 1.22.0, build f46880fe
```

Now that we have Docker Compose installed, we're ready to run a "Hello World" example.

Step 2 — Running a Container with Docker Compose

The public Docker registry, Docker Hub, includes a *Hello World* image for demonstration and testing. It illustrates the minimal configuration required to run a container using Docker Compose: a YAML file that calls a single image. We'll create this minimal configuration to run our `hello-world` container.

First, we'll create a directory for the YAML file and move into it:

```
$ mkdir hello-world
$ cd hello-world
```

Then, we'll create the YAML file:

```
$ nano docker-compose.yml
```

Put the following contents into the file, save the file, and exit the text editor:

docker-compose.yml

```
my-test:
 image: hello-world
```

The first line in the YAML file is used as part of the container name. The second line specifies which image to use to create the container. When we run the `docker-compose up` command, it will look for a local image by the name we specified, `hello-world`. With this in place, we'll save and exit the file.

We can look manually at images on our system with the `docker images` command:

```
$ docker images
```

When there are no local images at all, only the column headings display:

```
Output
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

Now, while still in the `~/hello-world` directory, we'll execute the following command:

```
$ docker-compose up
```

The first time we run the command, if there's no local image named `hello-world`, Docker Compose will pull it from the Docker Hub public repository:

```
Output
Pulling my-test (hello-world:)...
latest: Pulling from library/hello-world
```

```
latest: Pulling from library/hello-world
9db2ca6ccae0: Pull complete
Digest: sha256:4b8ff392a12ed9ea17784bd3c9a8b1fa3299cac44aca35a85c90c5e3c7afacdc
Status: Downloaded newer image for hello-world:latest
. . .
```

After pulling the image, `docker-compose` creates a container, attaches, and runs the hello program, which in turn confirms that the installation appears to be working:

Output

```
. . .
Creating helloworld_my-test_1...
Attaching to helloworld_my-test_1
my-test_1 |
my-test_1 | Hello from Docker.
my-test_1 | This message shows that your installation appears to be working correctly.
my-test_1 |
. . .
```

Then it prints an explanation of what it did:

Output

```
 To generate this message, Docker took the following steps:
my-test_1  |  1. The Docker client contacted the Docker daemon.
my-test_1  |  2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
my-test_1  |     (amd64)
my-test_1  |  3. The Docker daemon created a new container from that image which runs the
my-test_1  |     executable that produces the output you are currently reading.
my-test_1  |  4. The Docker daemon streamed that output to the Docker client, which sent it
my-test_1  |     to your terminal.
```

Docker containers only run as long as the command is active, so once `hello` finished running, the container stopped. Consequently, when we look at active processes, the column headers will appear, but the `hello-world` container won't be listed because it's not running:

```
$ docker ps
```

Output

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

We can see the container information, which we'll need in the next step, by using the `-a` flag. This shows all containers, not just active ones:

```
$ docker ps -a
```

Output

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
06069fd5ca23        hello-world         "/hello"            35 minutes ago      Exited (0)
```

This displays the information we'll need to remove the container when we're done with it.

Step 3 — Removing the Image (Optional)

To avoid using unnecessary disk space, we'll remove the local image. To do so, we'll need to delete all the containers that reference the image using the `docker rm` command, followed by either the `CONTAINER ID` or the `NAME`. Below, we're using the `CONTAINER ID` from the `docker ps -a` command we just ran. Be sure to substitute the ID of your container:

```
$ docker rm 06069fd5ca23
```

Once all containers that reference the image have been removed, we can remove the image:

```
$ docker rmi hello-world
```

Conclusion

We've now installed Docker Compose, tested our installation by running a Hello World example, and removed the test image and container.

While the Hello World example confirmed our installation, the simple configuration does not show one of the main benefits of Docker Compose — being able to bring a group of Docker containers up and down all at the same time. To see the power of Docker Compose in action, you might like to check out this practical example, How To Configure a Continuous Integration Testing Environment with Docker and Docker Compose on Ubuntu 16.04.

By: Kathleen Juell By: Melissa Anderson ♡ Upvote (0) Subscribe

Related Tutorials

How To Install Docker Compose on Ubuntu 18.04

How To Remove Docker Images, Containers, and Volumes

Naming Docker Containers: 3 Tips for Beginners

How to Use Traefik as a Reverse Proxy for Docker Containers on Ubuntu 18.04

How To Provision and Manage Remote Docker Hosts with Docker Machine on Ubuntu 18.04

0 Comments

Leave a comment...

Log In to Comment