**Step 1 — Installing Docker Compose**

In order to get the latest release, take the lead of the Docker docs and install Docker Compose from the binary in Docker's GitHub repository.

Check the current release and if necessary, update it in the command below:

* sudo curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

Next, set the permissions to make the binary executable:

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

Then, verify that the installation was successful by checking the version:

* docker-compose --version

This will print out the version you installed:

Output

docker-compose version 1.23.2, build 1110ad01

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

**Step 2 — Running a Container with Docker Compose**

The public Docker registry, Docker Hub, includes a simple "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.

First, create a directory for our YAML file:

* mkdir hello-world

Then change into the directory:

* cd hello-world

Now create the YAML file using your favorite text editor. This tutorial will use Vi:

* vi docker-compose.yml

Enter insert mode, by pressing i, then put the following contents into the file:

docker-compose.yml

my-test:

image: hello-world

The first line will be part of the container name. The second line specifies which image to use to create the container. When you run the command docker-compose up it will look for a local image by the name specified, hello-world.

With this in place, hit ESC to leave insert mode. Enter :x then ENTER to save and exit the file.

To look manually at images on your system, use 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, execute the following command to create the container:

* 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

1b930d010525: Pull complete

. . .

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 |

. . .

It will then print an explanation of what it did:

Output

. . .

my-test\_1 | 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 stops. Consequently, when you 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 PORTS NAMES

Use the -a flag to show all containers, not just the active ones:

* docker ps -a

Output

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

50a99a0beebd hello-world "/hello" 3 minutes ago Exited (0) 3 minutes ago hello-world\_my-test\_1

Now that you have tested out running a container, you can move on to exploring some of the basic Docker Compose commands.

**Step 3 — Learning Docker Compose Commands**

To get you started with Docker Compose, this section will go over the general commands that the docker-compose tool supports.

The docker-compose command works on a per-directory basis. You can have multiple groups of Docker containers running on one machine — just make one directory for each container and one docker-compose.yml file for each directory.

So far you've been running docker-compose up on your own, from which you can use CTRL-C to shut the container down. This allows debug messages to be displayed in the terminal window. This isn't ideal though; when running in production it is more robust to have docker-compose act more like a service. One simple way to do this is to add the -d option when you up your session:

* docker-compose up -d

docker-compose will now fork to the background.

To show your group of Docker containers (both stopped and currently running), use the following command:

* docker-compose ps -a

If a container is stopped, the State will be listed as Exited, as shown in the following example:

Output

Name Command State Ports

------------------------------------------------

hello-world\_my-test\_1 /hello Exit 0

A running container will show Up:

Output

Name Command State Ports

---------------------------------------------------------------

nginx\_nginx\_1 nginx -g daemon off; Up 443/tcp, 80/tcp

To stop all running Docker containers for an application group, issue the following command in the same directory as the docker-compose.yml file that you used to start the Docker group:

* docker-compose stop

**Note:** docker-compose kill is also available if you need to shut things down more forcefully.

In some cases, Docker containers will store their old information in an internal volume. If you want to start from scratch you can use the rm command to fully delete all the containers that make up your container group:

* docker-compose rm

If you try any of these commands from a directory other than the directory that contains a Docker container and .yml file, it will return an error:

Output

ERROR:

Can't find a suitable configuration file in this directory or any

parent. Are you in the right directory?

Supported filenames: docker-compose.yml, docker-compose.yaml

This section has covered the basics of how to manipulate containers with Docker Compose. If you needed to gain greater control over your containers, you could access the filesystem of the Docker container and work from a command prompt inside your container, a process that is described in the next section.

**Step 4 — Accessing the Docker Container Filesystem**

In order to work on the command prompt inside a container and access its filesystem, you can use the docker exec command.

The "Hello World" example exits after it runs, so to test out docker exec, start a container that will keep running. For the purposes of this tutorial, use the [Nginx image](https://hub.docker.com/_/nginx/) from Docker Hub.

Create a new directory named nginx and move into it:

* mkdir ~/nginx
* cd ~/nginx

Next, make a docker-compose.yml file in your new directory and open it in a text editor:

* vi docker-compose.yml

Next, add the following lines to the file:

~/nginx/docker-compose.yml

nginx:

image: nginx

Save the file and exit. Start the Nginx container as a background process with the following command:

* docker-compose up -d

Docker Compose will download the Nginx image and the container will start in the background.

Now you will need the CONTAINER ID for the container. List all of the containers that are running with the following command:

* docker ps

You will see something similar to the following:

Output of `docker ps`

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

b86b6699714c nginx "nginx -g 'daemon of…" 20 seconds ago Up 19 seconds 80/tcp nginx\_nginx\_1

If you wanted to make a change to the filesystem inside this container, you'd take its ID (in this example b86b6699714c) and use docker exec to start a shell inside the container:

* docker exec -it b86b6699714c /bin/bash

The -t option opens up a terminal, and the -i option makes it interactive. /bin/bash opens a bash shell to the running container.

You will then see a bash prompt for the container similar to:

root@b86b6699714c:/#

From here, you can work from the command prompt inside your container. Keep in mind, however, that unless you are in a directory that is saved as part of a data volume, your changes will disappear as soon as the container is restarted. Also, remember that most Docker images are created with very minimal Linux installs, so some of the command line utilities and tools you are used to may not be present.

