Run your image as a container

Estimated reading time: 9 minutes

Prerequisites

Work through the steps to build a Python image in Build your Python image (/language/python/build-images/).

Overview

In the previous module, we created our sample application and then we created a Dockerfile that we used to produce an image. We created our image using the docker command docker build. Now that we have an image, we can run that image and see if our application is running correctly.

A container is a normal operating system process except that this process is isolated in that it has its own file system, its own networking, and its own isolated process tree separate from the host.

To run an image inside of a container, we use the `docker run` command. The `docker run` command requires one parameter which is the name of the image. Let's start our image and make sure it is running correctly. Run the following command in your terminal.

```
$ docker run python-docker
```

After running this command, you'll notice that you were not returned to the command prompt. This is because our application is a REST server and runs in a loop waiting for incoming requests without returning control back to the OS until we stop the container.

Let’s open a new terminal then make a `GET` request to the server using the `curl` command.

```
$ curl localhost:5000
curl: (7) Failed to connect to localhost port 5000: Connection refused
```

As you can see, our `curl` command failed because the connection to our server was refused. This means, we were not able to connect to the localhost on port 5000. This is expected because our container is running in isolation which includes networking. Let’s stop the container and restart with port 5000 published on our local network.

To stop the container, press ctrl-c. This will return you to the terminal prompt.

To publish a port for our container, we’ll use the `--publish flag` (`-p` for short) on the `docker run` command. The format of the `--publish` command is `[host port]:[container port]` . So, if we wanted to expose port 5000 inside the container to port 3000 outside the container, we would pass `3000:5000` to the `--publish` flag.

We did not specify a port when running the flask application in the container and the default is 5000. If we want our previous request going to port 5000 to work we can map the host’s port 5000 to the container’s port 5000:

```
$ docker run --publish 5000:5000 python-docker
```

Now, let’s rerun the curl command from above. Remember to open a new terminal.

```
$ curl localhost:5000
Hello, Docker!
```

Success! We were able to connect to the application running inside of our container on port 5000. Switch back to the terminal where your container is running and you should see the GET request logged to the console.

```
[31/Jan/2021 23:39:31] "GET / HTTP/1.1" 200 -
```

Press ctrl-c to stop the container.

Run in detached mode

This is great so far, but our sample application is a web server and we don’t have to be connected to the container. Docker can run your container in detached mode or in the background. To do this, we can use the `--detach` or `-d` for short. Docker starts your container the same as before but this time will “detach” from the container and return you to the terminal prompt.

```
$ docker run -d -p 5000:5000 python-docker
ce02b3179f0f10085db9edfccd731101868f58631bdf918ca490ff6fd223a93b
```

Docker started our container in the background and printed the Container ID on the terminal.

Again, let’s make sure that our container is running properly. Run the same curl command from above.

```
$ curl localhost:5000
Hello, Docker!
```

List containers

Since we ran our container in the background, how do we know if our container is running or what other containers are running on our machine? Well, we can run the `docker ps` command. Just like on Linux to see a list of processes on your machine, we would run the `ps` command. In the same spirit, we can run the `docker ps` command which displays a list of containers running on our machine.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
ce02b3179f0f        python-docker         "python3 -m flask ru…"   6 minutes ago
```

The `docker ps` command provides a bunch of information about our running containers. We can see the container ID, The image running inside the container, the command that was used to start the container, when it was created, the status, ports that exposed and the name of the container.

You are probably wondering where the name of our container is coming from. Since we didn’t provide a name for the container when we started it, Docker generated a random name. We’ll fix this in a minute, but first we need to stop the container. To stop the container, run the `docker stop` command which does just that, stops the container. You need to pass the name of the container or you can use the container ID.

```
$ docker stop wonderful_kalam
wonderful_kalam
```

Now, rerun the `docker ps` command to see a list of running containers.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
```

Stop, start, and name containers

You can start, stop, and restart Docker containers. When we stop a container, it is not removed, but the status is changed to stopped and the process inside the container is stopped. When we ran the `docker ps` command in the previous module, the default output only shows running containers. When we pass the `--all` or `-a` for short, we see all containers on our machine, irrespective of their start or stop status.

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
ce02b3179f0f        python-docker         "python3 -m flask ru…"   16 minutes ago
ec45285c456d        python-docker         "python3 -m flask ru…"   28 minutes ago
fb7a41809e5d        python-docker         "python3 -m flask ru…"   37 minutes ago
```

You should now see several containers listed. These are containers that we started and stopped but have not been removed.

Let's restart the container that we just stopped. Locate the name of the container we just stopped and replace the name of the container below in the restart command.

```
$ docker restart wonderful_kalam
```

Now list all the containers again using the `docker ps` command.

```
$ docker ps --all
CONTAINER ID        IMAGE               COMMAND                  CREATED
ce02b3179f0f        python-docker         "python3 -m flask ru…"   19 minutes ago
ec45285c456d        python-docker         "python3 -m flask ru…"   31 minutes ago
fb7a41809e5d        python-docker         "python3 -m flask ru…"   40 minutes ago
```

Notice that the container we just restarted has been started in detached mode and has port 5000 exposed. Also, observe the status of the container is "Up X seconds". When you restart a container, it starts with the same flags or commands that it was originally started with.

Now, let's stop and remove all of our containers and take a look at fixing the random naming issue. Stop the container we just started. Find the name of your running container and replace the name in the command below with the name of the container on your system.

```
$ docker stop wonderful_kalam
wonderful_kalam
```

Now that all of our containers are stopped, let's remove them. When you remove a container, it is no longer running, nor it is in the stopped status, but the process inside the container has been stopped and the metadata for the container has been removed.

```
$ docker ps --all
CONTAINER ID        IMAGE               COMMAND                   CREATED
ce02b3179f0f        python-docker         "python3 -m flask ru…"   19 minutes ago
ec45285c456d        python-docker         "python3 -m flask ru…"   31 minutes ago
fb7a41809e5d        python-docker         "python3 -m flask ru…"   40 minutes ago
```

To remove a container, simple run the `docker rm` command passing the container name. You can pass multiple container names to the command using a single command. Again, replace the container names in the following command with the container names from your system.

```
$ docker rm wonderful_kalam agitated_moser goofy_khayyam
wonderful_kalam
agitated_moser
goofy_khayyam
```

Run the `docker ps --all` command again to see that all containers are removed.

Now, let's address the random naming issue. Standard practice is to name your containers for the simple reason that it is easier to identify what is running in the container and what application or service it is associated with.

To name a container, we just need to pass the `--name` flag to the `docker run` command.

```
$ docker run -d -p 5000:5000 --name rest-server python-docker
1aa5d46418a68705c81782a58456a4ccdb56a309cb5e6bd399478d01eaa5cdda
$ docker ps
CONTAINER ID        IMAGE               COMMAND                   CREATED
1aa5d46418a6        python-docker         "python3 -m flask ru…"   3 seconds ago
```

That's better! We can now easily identify our container based on the name.

Next steps

In this module, we took a look at running containers, publishing ports, and running containers in detached mode. We also took a look at managing containers by starting, stopping, and, restarting them. We also looked at naming our containers so they are more easily identifiable. In the next module, we'll learn how to run a database in a container and connect it to our application. See:

How to develop your application (/language/python/develop/)

Feedback

Help us improve this topic by providing your feedback. Let us know what you think by creating an issue in the Docker Docs (https://github.com/docker/docker.github.io/issues/new?title=[Python%20docs%20feedback]) GitHub repository. Alternatively, create a PR (https://github.com/docker/docker.github.io/pulls) to suggest updates.

Python (/search/?q=Python), run (/search/?q=run), image (/search/?q=image), container (/search/?q=container)