
Django Docker Intro Documentation

James Wettenhall

Mar 07, 2019

Contents

| | | |
|----------|--|-----------|
| 1 | Prerequisites | 3 |
| 1.1 | macOS | 3 |
| 1.2 | Windows | 3 |
| 1.3 | Linux | 4 |
| 2 | Getting Docker | 5 |
| 2.1 | Docker Community Edition | 5 |
| 2.2 | Docker on Linux | 5 |
| 2.3 | Testing your Docker installation | 6 |
| 3 | Docker Hello World | 7 |
| 3.1 | What Is Docker? | 7 |
| 3.2 | Running Docker's Hello World | 7 |
| 3.3 | Listing Docker Images Stored Locally | 7 |
| 3.4 | Listing Running and Stopped Docker Containers | 8 |
| 3.5 | Tagging Docker Images and Pushing to DockerHub | 8 |
| 4 | Django In Docker | 9 |
| 4.1 | Why Run Django in Docker? | 9 |
| 4.2 | Containerizing the Django Polls Application | 10 |
| 5 | The Django Polls Application | 11 |
| 5.1 | Security Disclaimer | 11 |
| 5.2 | Repository Implementing Polls Application | 11 |
| 5.3 | Polls Application Without Docker | 11 |
| 5.4 | Adding a RESTful API | 11 |
| 5.5 | Serializing Settings With YAML | 12 |
| 5.6 | Docker Implementation of Polls Application | 12 |
| 6 | Exercises and Further Reading | 13 |
| 6.1 | Exercise: Separate Front-End from Polls application | 13 |
| 6.2 | Kubernetes in Under 3 Hours | 13 |
| 6.3 | Exercise: Create a Kubernetes (Minikube) deployment of the Polls application | 13 |
| 6.4 | Learn How To Run Kubernetes In Your Favorite Cloud Environment | 13 |
| 6.5 | Domain-Driven Design | 14 |
| 6.6 | Docker CI/CD Workflows | 14 |

| | | |
|----------|---|-----------|
| 7 | Docker Cheat Sheet | 15 |
| 7.1 | Listing images, containers and volumes | 15 |
| 7.2 | Building a Docker image from a Dockerfile | 16 |
| 7.3 | Running an interactive shell inside a container | 16 |
| 7.4 | Pulling in images required in your docker-compose.yml | 16 |
| 7.5 | Starting up services defined in docker-compose.yml | 17 |
| 7.6 | Deleting all images, containers and volumes | 17 |

This tutorial aims to provide an introduction to using Docker with Django for developers who already have some familiarity with Django.

Contents:

CHAPTER 1

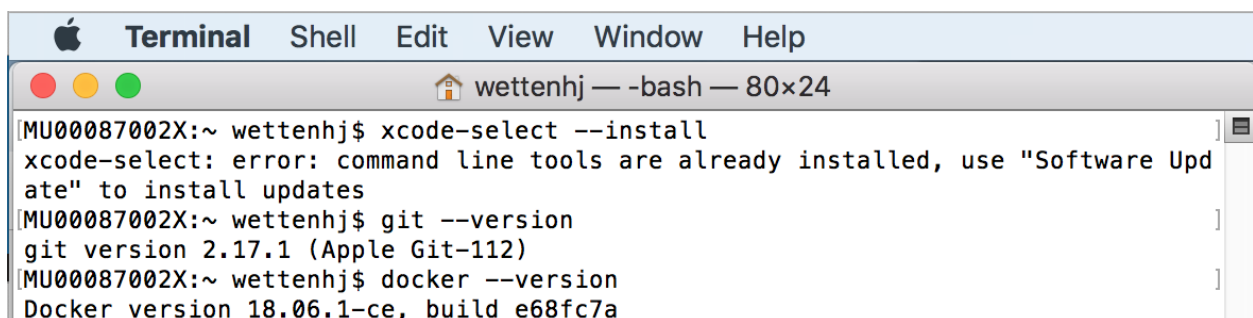
Prerequisites

You should ensure that you have at least 5 GB of free disk space to install and build the required Docker images.

You will need `git`, which may already be installed, e.g. if you have installed the developer tools on macOS by typing `xcode-select --install` into a Terminal window (see below). If you don't already have `git` installed, you can download it from <https://git-scm.com/>

1.1 macOS

On macOS, you can use the Terminal application, which can be found in macOS's Spotlight Search Tool:



```
Terminal  Shell  Edit  View  Window  Help
wettenhj — -bash — 80x24
[MU00087002X:~ wettenhj$ xcode-select --install
xcode-select: error: command line tools are already installed, use "Software Update" to install updates
[MU00087002X:~ wettenhj$ git --version
git version 2.17.1 (Apple Git-112)
[MU00087002X:~ wettenhj$ docker --version
Docker version 18.06.1-ce, build e68fc7a
```

And you will need `Docker`, described further in the *Getting Docker* section.

1.2 Windows

On Windows, Docker advises that you can use either Command Prompt or Power Shell, *but not Power Shell ISE*: <https://docs.docker.com/docker-for-windows/>

If running Windows 10, you could use Windows Subsystem for Linux (WSL):

- <https://docs.microsoft.com/en-us/windows/wsl/about>

- <https://docs.microsoft.com/en-us/windows/wsl/install-win10>

1.3 Linux

If you don't already have `git` installed, you can following the instructions at <https://git-scm.com/download/linux>. Instructions for installing `Docker` will be provided in the *Getting Docker* section.

2.1 Docker Community Edition

macOS and Windows 10 users can download a Docker Community Edition (CE) installer from:

- <https://store.docker.com/editions/community/docker-ce-desktop-mac>
- <https://store.docker.com/editions/community/docker-ce-desktop-windows>

Registration is required and can be done at <https://store.docker.com/signup>

Users of older operating systems may be able to use Docker Toolbox instead:

- https://docs.docker.com/toolbox/toolbox_install_mac/
- https://docs.docker.com/toolbox/toolbox_install_windows/

2.2 Docker on Linux

Digital Ocean has some excellent guides for installing Docker on Linux:

- <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04>
- <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04>
- <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-centos-7>
- <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-debian-9>

Linux users should note that *docker-compose* is required, but may not be installed automatically by following the guides above. On Ubuntu/Debian, after following the appropriate guide above, you can run:

```
sudo apt install docker-compose
```

2.3 Testing your Docker installation

Depending on your operating system and the way you install Docker, you may need to manually launch the Docker application / service after installation, before being able to run `docker` commands.

Try running pulling and running Docker's hello-world image:

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Docker Hello World

3.1 What Is Docker?

According to Wikipedia, Docker is a computer program that performs operating-system-level virtualization, also known as “containerization”.

Docker images are analogous to virtual machine images, but they can be much smaller than traditional virtual machine images, because instances of Docker images (which are called “containers”) usually only need to run one particular type of service or process.

3.2 Running Docker’s Hello World

Depending on your operating system and the way you install Docker, you may need to manually launch the Docker application / service after installation, before being able to run `docker` commands.

Try running pulling and running Docker’s hello-world image:

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cab9fde470971e499788
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

3.3 Listing Docker Images Stored Locally

We can run the `docker image ls` command to list the Docker images stored locally:

```
$ docker image ls
```

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|-------------|--------|--------------|--------------|--------|
| hello-world | latest | fce289e99eb9 | 2 months ago | 1.84kB |

In this case, we only have one Docker image. It was pulled from DockerHub (i.e. downloaded from DockerHub), which we could have done by typing:

```
$ docker pull hello-world
```

However the `docker run` command above pulled the image automatically, and then created a container (instance) of that image and ran the container, resulting in the `Hello from Docker!` message shown above.

3.4 Listing Running and Stopped Docker Containers

So if a container was created, is it still running? We can find out by running:

```
$ docker ps
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | |
|--------------|-------|---------|---------|--|
| ↪STATUS | PORTS | NAMES | | |

In this case, `docker ps` shows that no containers are currently running. To list all containers, included those which have stopped running, we can run:

```
$ docker ps -a
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | |
|--------------|-------------------|----------------------|-------------------|--|
| ↪STATUS | PORTS | NAMES | | |
| 636651c57ab7 | hello-world | "/hello" | About an hour ago | |
| ↪Exited (0) | About an hour ago | inspiring_goldwasser | | |

which shows us that a container was created from the `hello-world` image, and that it exited with a `(0)` (success) status code.

As we no longer need this stopped container, we can remove it, using its `CONTAINER ID`:

```
$ docker rm 636651c57ab7
```

3.5 Tagging Docker Images and Pushing to DockerHub

Given an existing image, e.g. this one:

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|-------------|--------|--------------|--------------|--------|
| hello-world | latest | fce289e99eb9 | 2 months ago | 1.84kB |

we can create one or more new tags for it, and push the tags to DockerHub:

```
$ docker tag fce289e99eb9 jameswettenhall/hello-world:1.0
$ docker tag fce289e99eb9 jameswettenhall/hello-world:latest
$ docker login -u jameswettenhall
$ docker push jameswettenhall/hello-world:1.0
$ docker push jameswettenhall/hello-world:latest
```

Now the image tags are available on DockerHub at: <https://cloud.docker.com/repository/docker/jameswettenhall/hello-world>

4.1 Why Run Django in Docker?

Django is a web application framework which can be used to develop full-stack web applications (using Django templates to render the front-end pages), or it can be used to develop a backend API (e.g. using the Django REST framework) which a front-end application can interact with.

Docker can help with CI (Continuous Integration) and CD (Continuous Deployment) of Django applications.

4.1.1 Simulating a Production Environment in Local Testing

It is easy to test a Django application locally using your local operating system (e.g. macOS) and a simple database engine (e.g. SQLite), but what if you want to run your tests in the same environment that will be used in production? As you become more advanced with Docker, you will want to use an orchestration tool like Kubernetes to run Docker containers in production. But if your production environment is not yet containerized, you can run your Django application's tests in a simulated version of your production server environment, by using a Docker image matching your server's operating system version. For example, if your server is running Ubuntu 18.04, you could run `docker pull ubuntu:18.04` to download a Docker image for Ubuntu 18.04, and run your tests in a container built from that image. You can also download an image for the database engine you use on your production server (e.g. MySQL or PostgreSQL) with `docker pull mysql` or `docker pull postgres`.

4.1.2 Continuous Integration

Each time you push a commit to your remote repository (e.g. GitHub / GitLab / BitBucket), you can set up a web hook so that your tests are automatically run in a Docker container matching your production environment, using a CI tool like Travis CI, Semaphore CI, CircleCI or Jenkins.

4.1.3 Continuous Deployment

If you containerize your production environment, and use a container orchestration tool like Kubernetes or Docker Swarm, then you can define replica sets (sets of Docker container instances) which can be updated potentially with zero downtime by replacing each instance within a replica set with a newer instance created from a new image pulled from DockerHub which has passed CI test before being pushed to DockerHub.

4.2 Containerizing the Django Polls Application

If you are familiar with Django, then you are probably familiar with the “Polls” application, implemented in the *Writing your first Django app* tutorial at <https://docs.djangoproject.com/en/2.1/intro/tutorial01/>

This tutorial demonstrates how to run the “Polls” application in a Docker container. It also demonstrates how to add a RESTful API to the “Polls” application, so that it can be split into separate front-end and back-end applications which could run in separate Docker containers.

The Django Polls Application

We'll use the result of running through Django's "Writing your first Django app" tutorial as the starting point for this tutorial. The Django tutorial can be found at <https://docs.djangoproject.com/en/2.1/intro/tutorial01/>

5.1 Security Disclaimer

Please note that we will not be listing all of the steps required to secure the application for a production environment. See: <https://docs.djangoproject.com/en/2.1/howto/deployment/checklist/>

5.2 Repository Implementing Polls Application

We'll use the following repository to illustrate how to run the "Polls" application in a Docker container: <https://github.com/jameswettenhall/django-polls-tutorial>

Each step of the tutorial will use a different branch of the repository:

5.3 Polls Application Without Docker

Firstly, we'll run the Polls application without Docker, using the `master` branch: <https://github.com/jameswettenhall/django-polls-tutorial/blob/master/README.md>

5.4 Adding a RESTful API

Then we'll add a RESTful API to the Polls application (still without Docker), using the `api` branch: <https://github.com/jameswettenhall/django-polls-tutorial/blob/api/README.md>

5.5 Serializing Settings With YAML

Then we'll discuss how to serialize the application's settings in YAML and make them available to the containerized application, using the `yaml` branch: <https://github.com/jameswettenhall/django-polls-tutorial/blob/yaml/README.md>

5.6 Docker Implementation of Polls Application

Then we'll discuss how to run the application in a Docker container, with a MySQL database server provided by another Docker container, using the `docker` branch: <https://github.com/jameswettenhall/django-polls-tutorial/blob/docker/README.md>

Exercises and Further Reading

6.1 Exercise: Separate Front-End from Polls application

Re-write the front-end of the Polls application in a front-end framework of your choice, e.g. React, Angular or Vue.js, instead of using Django templates. You can use the REST API we created as the data source for your front-end application.

6.2 Kubernetes in Under 3 Hours

Read the “Learn Kubernetes in Under 3 Hours” guide here: <https://medium.freecodecamp.org/learn-kubernetes-in-under-3-hours-a-detailed-guide-to-orchestrating-containers-114ff420e882> and learn how to install Minikube to run Kubernetes locally.

6.3 Exercise: Create a Kubernetes (Minikube) deployment of the Polls application

Using Minikube, create a local Kubernetes deployment of the Polls application and experiment with changing the number of replicas required of each container (front-end, back-end, MySQL).

6.4 Learn How To Run Kubernetes In Your Favorite Cloud Environment

- <https://kubernetes.io/docs/setup/custom-cloud/kubespray/>
- <https://cloud.google.com/kubernetes-engine/>
- <https://aws.amazon.com/kubernetes/>

- <https://www.digitalocean.com/products/kubernetes/>
- <https://azure.microsoft.com/en-au/services/kubernetes-service/>
- <https://www.openstack.org/assets/containers/openstack-containers-12.4.pdf>
- <https://github.com/infraly/k8s-on-openstack>
- etc.

6.5 Domain-Driven Design

Domain-Driven Design (DDD) is about finding ways to split a large application into logical components which could be implemented by microservices running in Docker containers.

- <https://stackoverflow.com/a/1222488>
- <https://medium.com/the-coding-matrix/ddd-101-the-5-minute-tour-7a3037cf53b8>
- <https://www.thoughtworks.com/insights/blog/domain-driven-design-services-architecture>
- <https://techbeacon.com/app-dev-testing/get-your-feet-wet-domain-driven-design-3-guiding-principles>
- <https://www.mirkosertic.de/blog/2013/04/domain-driven-design-example/>
- etc.

6.6 Docker CI/CD Workflows

- <https://success.docker.com/article/dev-pipeline>
- <https://github.com/mozilla-services/Dockerflow>
- etc.

7.1 Listing images, containers and volumes

List all of your images:

```
docker image ls
```

or to list images used by the created containers:

```
docker-compose images
```

List all of your running containers:

```
docker ps
```

List all of your containers (including stopped containers):

```
docker ps -a
```

or:

```
docker-compose ps
```

List services launched from your *docker-compose.yml*:

```
docker-compose config --services
```

List all of your volumes:

```
docker volume ls
```

Inspect a volume, including its Mountpoint:

```
docker volume inspect <volume_name>
```

List named volumes referenced in your *docker-compose.yml*:

```
docker-compose config --volumes
```

7.2 Building a Docker image from a Dockerfile

To build an image from a Dockerfile:

```
docker build
```

or if you have a *docker-compose.yml*:

```
docker-compose build
```

7.3 Running an interactive shell inside a container

You can either launch a new container to run an interactive shell allowing you to browse the contents of an image, or you can launch a shell inside an already running container.

To launch a new container which runs an interactive bash shell from an existing image, the method depends on whether the image has an entrypoint, and whether you want to override it.

If the image (e.g. *monashmerc/mytardis_django*) has an entrypoint, you can run */bin/bash* instead of the image's default entrypoint script and then look at the entrypoint script from the container you just created:

```
$ docker run -it --entrypoint /bin/bash monashmerc/mytardis_django
root@47d3e3917ed2:/usr/src/app# cat /docker-entrypoint.sh
```

If the image (e.g. *nginx*) doesn't have an entrypoint, you can create a container to run bash as follows:

```
docker run -it IMAGE /bin/bash
# e.g. docker run -it nginx /bin/bash
```

To launch a bash shell inside an already running container:

```
docker exec -it CONTAINER /bin/bash
# e.g. docker exec -it mytardis_celery_1 /bin/bash
```

To launch a bash shell inside a service (listed by *docker-compose config --services*):

```
docker-compose exec SERVICE /bin/bash
# e.g. docker-compose exec celery /bin/bash
```

7.4 Pulling in images required in your docker-compose.yml

To pull images (from DockerHub) required in your *docker-compose.yml*:

```
docker-compose pull
```

7.5 Starting up services defined in docker-compose.yml

To start up the services / containers defined in `docker-compose.yml`:

```
docker-compose up -d
```

where `-d` is “Detached mode” for running containers in the background.

To check the logs of services / containers launched with *docker-compose up -d*:

```
docker-compose logs -f
```

where `-f` means “Follow log output” and will continue displaying updates to logs until you press Control-C.

To stop services / containers launched with *docker-compose up -d*:

```
docker-compose down
```

7.6 Deleting all images, containers and volumes

WARNING: The commands below are destructive and are intended to be used when you want to clean up a test Docker environment and you don’t have any valuable Docker images or volumes you want to keep.

Remove all stopped containers, and receive an error for each container which can’t be removed because it is still running:

```
docker rm $(docker ps -aq)
```

Remove all containers, even those which are still running:

```
docker rm -f $(docker ps -aq)
```

Remove all volumes not being used by a container:

```
docker volume prune
```

Remove an image:

```
docker rmi <image_id>
```

Remove all images:

```
docker rmi $(docker image ls -q)
```

If attempting to remove images gives an error like `image is referenced in multiple repositories` (because you have created multiple tags from an image), you can add `-f` to remove them forcefully:

```
docker rmi -f $(docker image ls -q)
```