Building and running Docker images

Contents

Docker is a set of tools and services used to build, manage and deploy containers.

As many cloud platforms offer Docker-based container hosting services, creating Docker containers for a given service is a common task when building reproducible software. In this tutorial, you will learn how to build Docker containers using Nix.

Prerequisites

You will need both Nix and Docker installed. Docker is available in `nixpkgs`, which is the preferred way to install it on NixOS. However, you can also use the native Docker installation of your OS, if you are on another Linux distribution or macOS.

Build your first container

Nixpkgs provides `dockerTools` to create Docker images:

```
1 { pkgs ? import <nixpkgs> { }
2 , pkgsLinux ? import <nixpkgs> { system = "x86_64-linux"; }
3 }:
4
5 pkgs.dockerTools.buildImage {
6   name = "hello-docker";
```

Skip to main content

```
  9   };
 10 }
```

> **Note**
>
> If you're running **macOS** or any platform other than `x86_64-linux`, you'll need to either:
>
> - Set up a remote builder to build on Linux
> - Cross compile to Linux by replacing `pkgsLinux.hello` with `pkgs.pkgsCross.musl64.hello`

We call the `dockerTools.buildImage` and pass in some parameters:

- a `name` for our image
- the `config` including the command `Cmd` that should be run inside the container once the image is started. Here we reference the GNU hello package from `nixpkgs` and run its executable in the container.

Save this in `hello-docker.nix` and build it:

```
$ nix-build hello-docker.nix
these derivations will be built:
  /nix/store/qpgdp0qpd8ddi1ld72w02zkmm7n87b92-docker-layer-hello-docker.drv
  /nix/store/m4xyfyviwbi38sfplq3xx54j6k7mccfb-runtime-deps.drv
  /nix/store/v0bvy9qxa79izc7s03fhpq5nqs2h4sr5-docker-image-hello-docker.tar.gz.drv
warning: unknown setting 'experimental-features'
building '/nix/store/qpgdp0qpd8ddi1ld72w02zkmm7n87b92-docker-layer-hello-docker.drv
No contents to add to layer.
Packing layer...
Computing layer checksum...
Finished building layer 'hello-docker'
building '/nix/store/m4xyfyviwbi38sfplq3xx54j6k7mccfb-runtime-deps.drv'...
building '/nix/store/v0bvy9qxa79izc7s03fhpq5nqs2h4sr5-docker-image-hello-docker.tar
Adding layer...
tar: Removing leading `/' from member names
Adding meta...
Cooking the image...
Finished.
/nix/store/y74sb4nrhxr975xs7h83izgm8z75x5fc-docker-image-hello-docker.tar.gz
```

The image tag (`y74sb4nrhxr975xs7h83izgm8z75x5fc`) refers to the Nix build hash and makes sure that the Docker image corresponds to our Nix build. The store path in the last line of the output references the Docker image.

Skip to main content

Run the container

To work with the container, load this image into Docker's image registry from the default `result` symlink created by `nix-build`:

```
$ docker load < result
Loaded image: hello-docker:y74sb4nrhxr975xs7h83izgm8z75x5fc
```

You can also use the store path to load the image in order to avoid depending on the presence of `result`:

```
$ docker load < /nix/store/y74sb4nrhxr975xs7h83izgm8z75x5fc-docker-image-hello-dock
Loaded image: hello-docker:y74sb4nrhxr975xs7h83izgm8z75x5fc
```

Even more conveniently, you can do everything in one command. The advantage of this approach is that `nix-build` will rebuild the image if there are any changes and pass the new store path to `docker load`:

```
$ docker load < $(nix-build hello-docker.nix)
Loaded image: hello-docker:y74sb4nrhxr975xs7h83izgm8z75x5fc
```

Now that you have loaded the image into Docker, you can run it:

```
$ docker run -t hello-docker:y74sb4nrhxr975xs7h83izgm8z75x5fc
Hello, world!
```

Working with Docker images

A general introduction to working with Docker images is not part of this tutorial. The official Docker documentation is a much better place for that.

Note that when you build your Docker images with Nix, you will probably not write a `Dockerfile` as Nix replaces the Dockerfile functionality within the Docker ecosystem. Nonetheless, understanding the anatomy of a Dockerfile may still be useful to understand how Nix replaces each of its functions. Using the Docker CLI, Docker Compose, Docker Swarm or Docker Hub on the other hand may still be relevant, depending on your use case.

Skip to main content

Next steps

- More details on how to use `dockerTools` can be found in the reference documentation.
- You might like to browse through more examples of Docker images built with Nix.
- Take a look at Arion, a `docker-compose` wrapper with first-class support for Nix.
- Build docker images on a CI with GitHub Actions.