# Using Nix to generate Docker images

## and why prefer it over Dockerfiles

https://functional.cafe/@yvan

What's Nix again?

Nix is a package manager and a configuration language (pure and lazy) that describes a package or a developer environment as the complete list of its dependencies.

By complete list of its dependencies, it means it fetches (and locks) recursively the whole tree of anything you need to build a given package.

Nix puts everything in a big `/nix/store`, so packages can share dependencies, and at the same time you could have, e.g., 2 conflicting `glibc` versions on your machine.

Nix introduces `patchelf`[1] to control the path of every dynamic library that would depend on a binary!

[1] https://github.com/NixOS/patchelf

So, one `.nix` file (hermetic) means one perfectly reproducible developer environment.

```
let hash = "9402c27069da5c5217648ec9cfe6d437aeadb79d";
in { pkgs ? import (fetchTarball
  "https://github.com/NixOS/nixpkgs/archive/${hash}.tar.gz"
  { } }:

pkgs.mkShell {
  buildInputs = [
    pkgs.python3
  ];
}
```

It will ALWAYS output `Python 3.10.9`!

Docker (I will not introduce it) on the other hand has a `Dockerfile` format:

```
FROM alpine:3.14

RUN apk add --no-cache python3

CMD [ "python" "--version" ]
```

You can build one image with a `Dockerfile`! But running `docker build` later may produce a different image.

So, you have to distribute the image (that could be heavy) rather than the `Dockerfile` to ensure reproducibility ...

Do you know there is a Nix docker image? [2]

You could enter a Nix environment in it!

[2] https://hub.docker.com/r/nixos/nix

```
FROM nixos/nix

RUN nix-shell --pure -p python3 -I \
    nixpkgs="https://github.com/NixOS/nixpkgs/archive/\
9402c27069da5c5217648ec9cfe6d437aeadb79d.tar.gz"

CMD [ "python" "--version" ]
```

But, the size of our image is 1.2Gb ...

Docker images have a state, that could be mutated, sometimes nobody knows how to rebuild a modified image ...

However, we can do better: generate docker images out of Nix files!

The image will not contain Nix or any package manager, it will actually contain only what it really needs to run, so it could be smaller than Alpine ;)

```
let hash = "9402c27069da5c5217648ec9cfe6d437aeadb79d";
in { pkgs ? import (fetchTarball
  "https://github.com/NixOS/nixpkgs/archive/${hash}.tar.gz"
  { system = "x86_64-linux"; } }:

pkgs.dockerTools.buildImage {
  name = "my-image";
  config = { Cmd = [ "${pkgs.python3}/bin/python" ]; };
}
```

Now our image size is 53.5Mb! (VS 52.4Mb for Alpine Linux)

N.b. Nix also support layered images, and can even stream them!

Is it hard to turn an existing `Dockerfile` into a Nix expression?

Yes, it's switching from an imperative model to a declarative one ...

But I'm starting to experiment with an interactive CLI that would help user to step by step do the rewriting.

Please ping me if that would be something that interests you!

Thank you

`impressive` is the ignite talk CLI tool suggested by CfgMgtCamp organizer:

```
nix-shell -p impressive
error: impressive has been removed due to lack of
released python 2 support and maintainership in nixpkgs
```

It's so easy to contribute to `nixpkgs` ($\sim$ 5000 contributors), give it a try!