Reproducible builds and deployments.

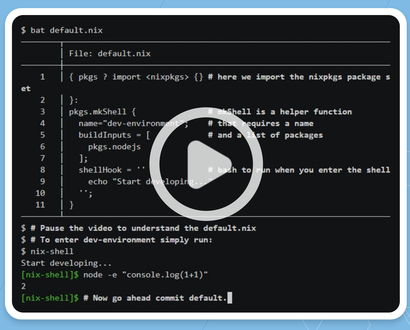
Nix

Powerful package manager for Linux and other Unix systems that makes package management reliable and reproducible. Share your development and build environments across different machines.

NixOS

Linux distribution with a unique approach to package and configuration management. Built on top of the Nix package manager, it is completely declarative, makes upgrading systems reliable, and has many other advantages.

Get started Download



Why choose Nix or NixOS?

It`s Reproducible...

Nix builds packages in isolation from each other. This ensures that they are reproducible and don't have undeclared dependencies, so if a package works on one machine, it will also work on another.

It`s Declarative...

Nix makes it trivial to share development and build environments for your projects, regardless of what programming languages and tools you're using.

It`s Reliable...

Nix ensures that installing or upgrading one package cannot break other packages. It allows you to roll back to previous versions, and ensures that no package is in an inconsistent state during an upgrade.

Choose from Thousands of Packages

The Nix Packages collection **(nixpkgs)** is a set of over 60.000 packages for the Nix package manager.

Search for a package Download

or search among many NixOS options.

Examples

Try new tools without fear

Don't clutter your system with tools that you use only now and then.

```
$ python --version
python: command not found
$ nix-shell -p python3
[nix-shell]$ python --version
Python 3.7.7
```

Multiple languages, one tool

```
$ nix-shell -p python3 nodejs go rustc
[nix-shell]$ node --version
v10.20.1
[nix-shell]$ go version
go version go1.14.1 linux/amd64
[nix-shell]$ rustc --version
rustc 1.42.0
```

Isolated development environments

After you get familiar with nix-shell -p you can take the next step and learn some Nix. To setup a more persistent environment you can also write a simple shell.nix file:

```
{ pkgs ? import <nixpkgs> {}
}:
pkgs.mkShell {
  name = "dev-shell";
  buildInputs = [
    pkgs.python3
    pkgs.python3Packages.virtualenv
    pkgs.nodejs
    pkgs.yarn
  ];
}
```

Then enter development environment with:

```
$ nix-shell
[nix-shell]$ virtualenv --version
16.7.9
[nix-shell]$ yarn --version
1.22.4
```

Commit the above shell.nix file and let you coworkers have easier time setting their development environment.

Minimal docker image

Using a Dockerfile, you are responsible for:
- cleaning up everything that is not needed at runtime
- deciding how to split into layers for better caching

Writing a Dockerfile that would produce a minimal image is at best a very error prone process.

With Nix only packages you define are included in the docker image. No cleaning up needed. There are no build tools left in your docker image, making it as minimal as you need.

Nix also knows how to layer your resulting docker image, automatically. The resulting layers are optimized for caching as much as possible.

The following Nix expression (default.nix) defines a docker image with only hello package in it.

```
{ pkgs ? import <nixpkgs> {}
}:
pkgs.dockerTools.buildLayeredImage {
  name = "only-hello";
  contents = [ pkgs.hello ];
}
```

To build and run the image you need to:

```
$ nix-build default.nix -o ./result
...
/nix/store/...-docker-image-only-hello.tar.gz
$ docker load -i ./result
1c31fbac2eb1: Loading layer [==================>]  1.649MB/1.649MB
03b22f638054: Loading layer [==================>]    256kB/256kB
29c350a9c392: Loading layer [==================>]  31.61MB/31.61MB
6a87e4d71e07: Loading layer [==================>]  266.2kB/266.2kB
c09c43a6b918: Loading layer [==================>]  71.68kB/71.68kB
Loaded image: only-hello:qn5x1pnk7d467js181jng7168qsks421
$ docker run only-hello:qn5x1pnk7d467js181jng7168qsks421 hello"
Hello, world
```

Learn more how to build docker images.