# Devbox Nix dev environments

**START HERE**

`devbox init`
Create devbox.json

`devbox search <pkg>`
Find packages

`devbox add <pkg>`
Add a package

`devbox update`
Refresh versions

`devbox shell`
Enter the environment

`devbox services up`
Start services

`devbox run <script>`
Run a script

`devbox rm <pkg>`
Remove a package

**devbox.json** packages, env, scripts → `devbox add` → **devbox.lock** pinned versions → `devbox shell` → **Nix store** installed packages → `PATH` → **Shell** isolated tools

## CONCEPTS

devbox.json lists the packages a project needs; devbox.lock pins exact versions. Commit both so everyone gets the same tools.

Devbox wraps Nix: you get reproducible packages without writing Nix code. Nix itself is installed for you on first use.

Flags of devbox run must come before the script or command name.

## PROJECT SETUP

`devbox init`
Create devbox.json in this directory

`devbox add <pkg>`
Add a package to the project

`devbox add <pkg>@<version>`
Add a specific version

`devbox rm <pkg>`
Remove a package

`devbox install`
Install all packages without a shell

`devbox search <pkg>`
List matching packages and versions

`devbox info <pkg>`
Show package and plugin info

`devbox update`
Update all packages within their version tags

`devbox update <pkg>`
Update only the named packages
*python@3.11 moves to the newest 3.11.x patch release.*

`devbox add <pkg> -p <platform>`
Install only on given platforms

`devbox add <pkg> --allow-insecure`
Allow a package flagged insecure

## SHELL AND RUN

`devbox shell`
Start a shell with your packages

`devbox shell -- <cmd>`
Run one command in the environment

## SHELL AND RUN (cont.)

`devbox shell --pure`
Isolated shell, almost no inherited env
*HOME, USER and DISPLAY are kept.*

`devbox shell -c <dir>`
Use the devbox.json in another directory

`devbox shell -e <KEY>=<val>`
Set a variable inside the environment

`devbox shell --env-file <file>`
Load variables from a file

`devbox shell --print-env`
Print a script that sets up the environment

`devbox run <script>`
Run a script from devbox.json

`devbox run <cmd>`
Run any command inside the environment

`devbox run -e <KEY>=<val> <script>`
Run with extra variables

`exit`
Leave the devbox shell

`devbox version`
Print the devbox version

## DEVBOX.JSON

**Packages (list form)**

```
{
  "packages": ["go@latest",
"nodejs@20"]
}
```

**Packages (map form)**

```
"packages": {
  "busybox": {
    "version": "latest",
    "platforms": ["x86_64-
linux"]
  }
}
```

**Environment variables**

```
"env": {
  "APP_ENV": "dev",
  "BIN": "$PWD/bin"
}
```

`"env_from": "<path>.env"`
Load variables from a .env file
*Values set in devbox.json win over the .env file.*

**Init hook**

```
"shell": {
  "init_hook": ["echo Welcome",
"export FOO=bar"]
}
```

## DEVBOX.JSON (cont.)

**Scripts**

```
"shell": {
  "scripts": {
    "test": "go test ./...",
    "ci": ["go vet ./...", "go
test ./..."]
  }
}
```

`"include": ["plugin:<name>"]`
Merge plugins or a local plugin.json
*Later plugins override earlier ones; your config overrides all plugins.*

`"github:<owner>/<repo>"`
Use a Nix flake as a package source

Valid platforms: x86_64-linux, aarch64-linux, x86_64-darwin, aarch64-darwin. Set platforms or excluded_platforms on a package, not both.

init_hook runs every time a shell starts, so keep it fast and idempotent.

## SERVICES

`devbox services ls`
List available services

`devbox services up`
Start all services with process-compose

`devbox services up <svc>`
Start only the named services

`devbox services up -b`
Run services in the background

`devbox services start <svc>`
Start a service (all if none named)

`devbox services stop <svc>`
Stop a service (all if none named)

`devbox services restart <svc>`
Restart a service

`devbox services up --process-compose-file <path>`
Use your own process-compose file

Packages with a plugin (postgresql, redis, nginx, ...) expose ready-made services; others need a process-compose.yaml next to devbox.json.

## GENERATE AND GLOBAL

`devbox generate direnv`
Write an .envrc that loads the environment

`devbox generate dockerfile`
Dockerfile that replicates devbox shell

`devbox generate devcontainer`
Create .devcontainer/ files for VS Code

`devbox generate devcontainer --root-user`
Use root and single-user Nix in the image

`devbox global add <pkg>`
Install a package for all projects

`devbox global list`
List global packages

`devbox global rm <pkg>`
Remove a global package

`devbox global shellenv`
Print shell code adding global packages to PATH

Add eval "$(devbox global shellenv)" to your shell rc file to use global packages everywhere.

## RECIPES

**Start a project**

```
devbox init
devbox add python@3.12
postgresql@16
devbox shell
python --version
```

**Auto-load with direnv**

```
devbox generate direnv
direnv allow
# entering the directory now
loads the environment
```

**Scripts and CI**

```
# devbox.json:
shell.scripts.test
devbox run test
devbox run -e CI=1 test
```

**Services in the background**

```
devbox services up -b
devbox services ls
devbox services stop
```

**Containerize**

```
devbox generate dockerfile
docker build -t myapp .
```