

Nix ❤️ Python:

Maybe you don't need devcontainers after all

# A lot of questions

1. What is Nix?
2. What is devenv?
3. Why is this relevant to me as a Python engineer?
4. Do I still need Docker?

# What is Nix?



# What is Nix?



Nix - the build system and package manager

- Declarative
  - Declare inputs and desired output state, Nix handles the rest
- Reproducible
  - $\text{build(inputs)} \rightarrow \text{output}$
  - No accidentally installed system libs, user tools
  - Multiple version of same lib coexist in peace

# What is Nix?



Nix - the purely function language

- Package definitions
- Configuration management

```
1 system.primaryUser = "tomislavmaricevic";
2
3 system.defaults = {
4     dock.autohide = true;
5     dock.show-recents = false;
6     finder.AppleShowAllExtensions = true;
7     finder.AppleShowAllFiles = true;
8     screencapture.location = "~/Pictures/screenshots";
9     screensaver.askForPasswordDelay = 10;
10    controlcenter.Bluetooth = true;
11
12 system.keyboard = {
13     enableKeyMapping = true;
14     remapCapsLockToEscape = true;
15 };
16
17 homebrew = {
18     enable = true;
19     brews = [
20         "ollama"
21         "borders"
22     ];
23     taps = [
24         "FelixKratz/formulae"
25     ];
26     casks = [
27         "google-chrome"
28         "raycast"
29         "chatgpt"
30         "whatsapp"
31         "dropbox"
32         "orcaslicer"
33         "stremio"
34         "claude"
35         "todoist-app"
36         "anki"
37     ];
38 };
39
40 services.aerospace = {
41     enable = true;
42     settings = builtins.fromTOML (builtins.readFile ../config/aerospace/aerospace.toml);
43 };
```

# What is Nix?

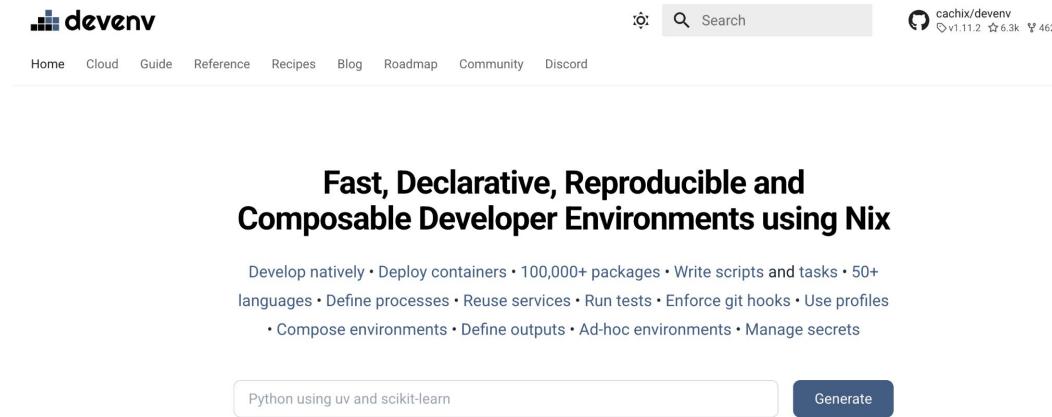


NixOS - Linux distribution based on Nix

- Uses Nix lang for configuration and Nix package manager for packages

Switching atomically between system states as immutable artifacts >> mutating a live system

# What is devenv?

The screenshot shows the devenv website. At the top, there is a navigation bar with links for Home, Cloud, Guide, Reference, Recipes, Blog, Roadmap, Community, and Discord. To the right of the navigation is a search bar with a magnifying glass icon and a placeholder 'Search'. Further right is a GitHub icon followed by the text 'cachix/devenv v1.11.2 ⭐6.3k 462'. The main content area features a large title 'Fast, Declarative, Reproducible and Composable Developer Environments using Nix'. Below the title is a list of features: 'Develop natively • Deploy containers • 100,000+ packages • Write scripts and tasks • 50+ languages • Define processes • Reuse services • Run tests • Enforce git hooks • Use profiles • Compose environments • Define outputs • Ad-hoc environments • Manage secrets'. At the bottom of the main content area is a search bar containing the text 'Python using uv and scikit-learn' and a 'Generate' button.

Nix-based development environment manager

devenv = nix shell + nice abstractions + service management + direnv

# Why is this relevant to me as a Python engineer?

Scenario:

- Monorepo with Django backend and React frontend
- Backend:
  - uv for package management
  - granian web server
  - rabbitmq message broker
  - Celery workers
  - ruff formatting
- Frontend:
  - Node v24
  - pnpm package management
  - Biome formatting

```
1  ./. devenv.nix •
2 {
3   pkgs,
4   lib,
5   config,
6   inputs,
7   ...
8 }
9 {
10  # Helper packages, and optional lib dependencies
11  packages = [
12    pkgs.jujutsu
13    pkgs.jq
14    pkgs.zlib
15  ];
16
17  # Language runtimes and package managers
18  languages = {
19    python = {
20      enable = true;
21      version = "3.12";
22      uv.enable = true;
23    };
24    javascript = {
25      enable = true;
26      package = pkgs.nodejs_24;
27      pnpm.enable = true;
28    };
29  };
30
31  # Predefined processes
32  services = {
33    postgres = {
34      enable = true;
35      package = pkgs.postgresql_18;
36      initialDatabases = [
37        {
38          name = "app_db";
39          user = "app_user";
40          pass = "app_pass";
41        }
42      ];
43      initialScript = ''
44        CREATE USER app_user WITH PASSWORD 'app_pass';
45      '';
46    };
47    redis.enable = true;
48    rabbitmq = {
49      enable = true;
50      managementPlugin.enable = true;
51    };
52  };
53
54  # Ad-hoc processes
55  processes = {
56    api = {
57      exec = "uv run granian --interface rsgi main:app";
58      cwd = "${config.git.root}/backend";
59    };
60    worker = {
61      exec = "uv run celery -A tasks worker --loglevel=info";
62      cwd = "${config.git.root}/backend";
63    };
64    frontend = {
65      exec = "pnpm dev";
66      cwd = "${config.git.root}/frontend";
67    };
68  };
69
70  git-hooks.hooks = {
71    ruff.enable = true;
72    biome.enable = true;
73  };
74 }
```

```
70 git-hooks.hooks = {
71     ruff.enable = true;
72     biome.enable = true;
73 };
74
75 claudie.code = {
76     enable = true;
77     agents = {
78         code-reviewer = {
79             description = "Expert code review specialist that checks for quality, security, and best practices";
80             proactive = true; # Claude will use this automatically when appropriate
81             tools = [
82                 "Read"
83                 "Grep"
84                 "TodoWrite"
85             ];
86             prompt = ''
87             You are an expert code reviewer. When reviewing code, check for:
88             - Code readability and maintainability
89             - Proper error handling
90             - Security vulnerabilities
91             - Performance issues
92             - Adherence to project conventions
93
94             Provide constructive feedback with specific suggestions for improvement.
95         '';
96     };
97
98     test-writer = {
99         description = "Specialized in writing comprehensive test suites";
100        proactive = false; # Only invoked explicitly
101        tools = [
102            "Read"
103            "Write"
104            "Edit"
105            "Bash"
106        ];
107        prompt = ''
108        You are a test writing specialist. Create comprehensive test suites that:
109        - Cover edge cases and error conditions
110        - Follow the project's testing conventions
111        - Include unit, integration, and property-based tests where appropriate
112        - Have clear test names that describe what is being tested
113    '';
114 };
115 };
116 hooks = {
117     # Protect sensitive files (PreToolUse hook)
118     protect-secrets = {
119         enable = true;
120         name = "Protect sensitive files";
121         hookType = "PreToolUse";
122         matcher = "^(Edit|MultiEdit|Write)$";
123         command = ''
124             # Read the JSON input from stdin
125             json=$(cat)
126             file_path=$(echo "$json" | jq -r '.file_path // empty')
127
128             if [[ "$file_path" =~ \.(env|secret)$ ]]; then
129                 echo "Error: Cannot edit sensitive files"
130                 exit 1
131             fi
132         '';
133     };
134 };
135 };
136 }
```

# Do I still need Docker?

Yes.

# Do I still need Docker?

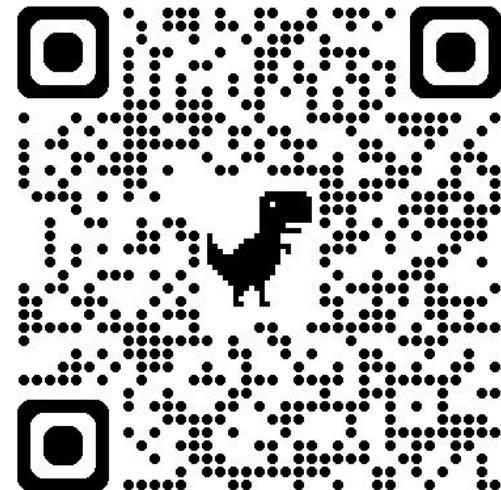
Yes, when:

- You need complex networking (e.g. many local services) – service discovery, compose network
- Testing container-related functionality
- High production parity is necessary
- Toolchain native performance is not crucial
- You like to develop in the cloud (e.g. Github Codespaces)

# Docker Was Too Slow, So We Replaced It: Nix in Production @ Anthropic



<https://www.youtube.com/watch?v=iPoL03tFBtU>



Thank you!

Q & A!

Open to new opportunities, reach out!

Tomislav Maričević | <https://tmarice.dev> | @tmrcv | github.com/tmarice