ChatGPT

# Advanced NixOS & Pixi Configuration Repositories (Late 2025 – Present)

## Agentic Robotics & Embedded Systems

**Botnix (nervosys/Botnix)** – *Last update: active development (major 1.0 release pending)*. **Primary tooling**: NixOS (custom "Botpkgs" package set). **Use case**: Robotics OS for multi-agent autonomous systems (a true "Robot Linux"). **Platforms**: Desktop and embedded (x86_64, ARM boards for robots). **Noteworthy features**: Offers **7 operational modes** (Agent, Orchestrator, Simulator, etc.) tailored to AI and robotics tasks [1] [2]. Built on the NixOS model for declarative, reproducible system configuration, with a curated *Botpkgs* repository for robotics packages [3] [4]. Integrates with a companion simulation engine (AutonomySim) for seamless real-world and simulated agent development [5]. Botnix aims to simplify deploying ROS-like functionality on a full OS, addressing pain points of ROS on traditional systems by providing a *"single-purpose, battle-hardened variant"* of NixOS for intelligent robots [6] [7].

**nix-ros-overlay (lopsided98/nix-ros-overlay)** – *Last update: late 2025 (continuous ROS 2 support)*. **Primary tooling**: Nix package overlay (usable on any Linux distro). **Use case**: Reproducible ROS 1/2 environment setup across Linux distributions. **Platforms**: Linux (distro-agnostic, including NixOS). **Noteworthy features**: Provides Nix expressions to **install ROS packages on any Linux in the same way** [8] – ideal if you *"want to use ROS, but don't want to run Ubuntu"* [9]. Leverages Nix's functional builds for reliable, shareable ROS setups, with binary caches (Cachix) to speed up development [10] [11]. Includes ready-made examples (e.g. ROS 2 desktop demo) and flake integration, so developers can quickly enter a Nix shell with a full ROS workspace or add ROS packages to their own flakes [12] [13]. This overlay lowers the barrier to entry for ROS on NixOS or mixed-OS environments by ensuring environment consistency.

**Pixi (prefix-dev/pixi)** – *Last update: Dec 22, 2025 (v0.62.2)*. **Primary tooling**: Pixi CLI (Conda-based package & workflow manager) used alongside NixOS/Nix. **Use case**: Reproducible multi-language dev environments, increasingly adopted in robotics and AI research. **Platforms**: Cross-platform (NixOS, other Linux, macOS, Windows; supports x86_64 & ARM) [14] [15]. **Noteworthy features**: Pixi introduces **conda/PyPI package integration on Nix** with Cargo-like ergonomics [16] [14]. It captures exact dependencies in project lockfiles for bit-for-bit consistency across machines [17]. A high-performance SAT solver yields dependency resolution *"up to 10× faster"* than comparable tools [18]. Pixi can install project-specific tools or global utilities, and it includes a cross-platform task runner (via Deno shell) for defining build/test tasks once and running on any OS [19]. Notably, Pixi has been embraced by robotics/AI developers – an arXiv report (Nov 2025) notes Pixi was *"adopted in over 5,300 projects since 2023"* due to drastically reducing environment setup time for complex multi-language workflows [20]. *(Pixi also provides shell completions for Nushell, ensuring smooth CLI use in modern shells.)*

## Distributed Compute & DevOps Infrastructure

**Robotnix (nix-community/robotnix)** – *Last update: revived in 2025 (active maintenance)*. **Primary tooling**: NixOS-style modules and Nix builds. **Use case**: Distributed build pipeline for **Android OS images**

(LineageOS, GrapheneOS, AOSP) using Nix. **Platforms**: Desktop Linux (as build host); outputs custom Android images for mobile/embedded devices (ARM). **Noteworthy features**: Robotnix lets you reliably build Android **from source with one Nix command**, orchestrating all required toolchains and sources [21] [22]. It introduces a NixOS-like module system to easily **customize builds** – enabling options for MicroG integration, Chromium/WebView swaps, or even verified boot signing with user keys [23] [24]. By using Nix, Robotnix ensures the entire AOSP build environment is captured declaratively, so anyone can reproduce identical firmware (a big win for Android research or small OEMs). After a period of hiatus, new maintainers in 2025 restored support for latest LineageOS/GrapheneOS releases, though the project is still labeled alpha [25]. It showcases Nix's power in a **cross-compilation, multi-repo build** scenario – a niche where reproducibility is notoriously hard.

**k3s-nix (rorosen/k3s-nix)** – *Last update: late 2025*. **Primary tooling**: NixOS configurations and NixOS VM tests. **Use case**: Declarative **Kubernetes cluster** setup (k3s distro) and app deployments entirely in Nix. **Platforms**: NixOS hosts (tested on x86_64; adaptable to aarch64). **Noteworthy features**: A single Nix flake defines a lightweight **2-node k3s cluster** (one master, one agent) *and* the Kubernetes workloads on it [26] [27]. It uses the NixOS `services.k3s` module to auto-deploy Kubernetes manifests and even pre-load container images at boot [28] – resulting in a self-contained, **air-gappable** cluster environment (no external downloads at runtime) [29]. Example configs include a Node Exporter DaemonSet, Prometheus & Grafana deployments, and an nginx via Helm – all defined in Nix [30] [31]. Secrets are managed with sops-nix, and the repo provides a NixOS test that boots VMs and validates the cluster health end-to-end [32]. This approach ensures that your dev, CI, and prod k3s clusters are identical and easily reproducible – ideal for homelab setups or edge deployments where consistency is critical [29] [33].

**Fleet (deltarocks/fleet)** – *Last update: 2025*. **Primary tooling**: Nix CLI tool (`fleet`) plus NixOS module extensions. **Use case**: **NixOS cluster deployment** and configuration management (modern alternative to NixOps/Morph). **Platforms**: NixOS hosts (multi-architecture). **Noteworthy features**: Fleet introduces the ability for NixOS **modules to configure multiple hosts at once** [34] – for example, a single module can set up inter-node WireGuard or a Kubernetes cluster by applying config across all specified machines. Cluster secrets are handled via built-in **age encryption**: secrets live in Git but only target hosts can decrypt them, and Fleet can auto-regenerate and re-encrypt on changes [35] [36]. Deployments are transactional with automatic **rollbacks on failure**, ensuring cluster stability (as long as the system boots to init) [37]. A flake integration is provided (via Hercules' flake-parts), making it easy to plug Fleet into your monorepo flake and declare `fleetConfigurations` for different clusters [38] [39]. Overall, Fleet brings NixOS deployments closer to a "infrastructure-as-code" ideal, letting you manage multi-host setups with one source of truth and safe updates.

**Standard (divnix/std)** – *Last update: 2025 (beta)*. **Primary tooling**: Nix flakes with a framework ("std") library. **Use case**: **DevOps monorepo framework** for projects using Nix – structure and automate everything from development shells to CI pipelines. **Platforms**: Cross-platform (dev environments on Linux/macOS; outputs can target NixOS, containers, etc.). **Noteworthy features**: *"Standard"* (from the Divnix group) organizes Nix code into a high-level structure of **Cells** (project areas) and **Cell Blocks** (types of outputs, e.g. packages, shells, CI jobs) [40] [41]. It generates a consistent CLI/TUI for your repository, so team members can discover "what can I do with this repo?" intuitively [40]. Under the hood, std integrates a suite of Nix community tools: Devshell for reproducible dev environments, Treefmt for auto-formatting, Nix2Container for container images, Flake Parts for modular config, Cachix for binaries, and more [42]. The framework enforces best practices to prevent your Nix code from turning into "a giant ball of spaghetti" by imposing a clear folder/output convention [43]. In short, Standard is like a **starter kit for Nix-powered**

**startups** – it helps manage everything in one repo (apps, NixOS configs, CI) with sane defaults, so you can *"ship today."* [44] [45]

**NixOS-QChem (Nix-QChem/NixOS-QChem)** – *Last update: 2025*. **Primary tooling**: Nix overlay + modules for HPC software. **Use case**: **Scientific computing/HPC** environment for quantum chemistry, fully reproducible for research. **Platforms**: Linux HPC clusters (Nix on cluster head nodes or user space). **Noteworthy features**: NixOS-QChem provides a curated overlay of **popular quantum chemistry and HPC packages** tuned for Nix/NixOS [46]. Its goal is to make NixOS suitable for HPC by integrating performance optimizations and allowing multiple MPI/BLAS variants, etc., beyond what upstream nixpkgs offers [47]. The project maintains branches aligned with NixOS releases and offers example Nix shells (including Jupyter notebook setups) for scientists to get started quickly [48] [49]. Impressively, the maintainers published a peer-reviewed paper about the design, and they encourage users to **cite the specific Nix environment commit** in publications for true reproducibility [50]. This academic collaboration showcases Nix's strengths in HPC: using one config to deploy identical software stacks on clusters, enabling experimental results to be replicated exactly by others [51].

## Reproducible Cross-Platform Development

**Pixi** – *See above; relevant to this category as well.* Pixi's cross-OS package management and built-in task runner make it a centerpiece for reproducible CLI workflows. It bridges gaps between languages (Python, C++, R, etc.) on different platforms [16] [14]. For example, a robotics team can define their environment once and deploy it on a NixOS robot, a Windows laptop, or a Mac, all using Pixi to resolve dependencies. Pixi ensures environment **lockstep consistency** between developers and CI agents by using a unified lockfile [17]. Its tasks (powered by Deno) run the same on PowerShell or Bash or Nushell, meaning build/test scripts are truly portable [52]. Given these strengths, Pixi is increasingly seen alongside NixOS in projects that demand *"works on my machine"* parity across diverse setups.

**Additional Mentions**: The Nix ecosystem offers other tools for cross-platform dev environments. **Devbox (jetpack-io/devbox)**, for instance, provides container-less, Nix-based development shells on any OS through a simple config (abstracting Nix for ease of use). **Devenv.sh** (by Determinate Systems) similarly lets teams define a unified environment (with editor plugins, language servers, etc.) in a YAML, supporting monorepos and different OSes. And for those experimenting with alternative shells, **Nuenv** (an experimental Nushell-based Nix build environment) demonstrated that even Nix's build scripts can run in modern shells [53] [54]. All these efforts highlight a trend: using Nix's declarative approach to guarantee that whether on a Linux server, a MacBook, or CI runner, developers and "agent" processes share the same CLI tools, dependencies, and configurations. This consistency, combined with task runners (Just, Make, or Nix's own), yields agent-friendly workflows where automation can reliably invoke complex toolchains in a reproducible state – a critical factor for DevOps and AI-driven agents alike.

---

1 2 3 4 5 6 7 GitHub - nervosys/Botnix: The operating system for robotic general intelligence™
https://github.com/nervosys/Botnix

8 9 10 11 12 13 GitHub - lopsided98/nix-ros-overlay: ROS overlay for the Nix package manager
https://github.com/lopsided98/nix-ros-overlay

14 15 16 GitHub - prefix-dev/pixi: Package management made easy
https://github.com/prefix-dev/pixi

17 18 20 [2511.04827] Pixi: Unified Software Development and Distribution for Robotics and AI
https://arxiv.org/abs/2511.04827

19 52 Advanced tasks - Pixi by prefix.dev
https://prefix-dev.github.io/pixi/v0.15.1/advanced/advanced_tasks/

21 22 23 24 25 GitHub - nix-community/robotnix: Build Android (AOSP) using Nix [maintainer=@danielfullmer,@Atemu,@cyclic-pentane]
https://github.com/nix-community/robotnix

26 27 28 29 30 31 32 33 GitHub - rorosen/k3s-nix: Reproducible k3s clusters and deployments in pure Nix
https://github.com/rorosen/k3s-nix

34 35 36 37 38 39 GitHub - deltarocks/fleet: NixOS cluster configuration tool
https://github.com/deltarocks/fleet

40 41 42 43 44 45 GitHub - divnix/std: A DevOps framework for the SDLC with the power of Nix and Flakes. Good for keeping deadlines!
https://github.com/divnix/std

46 47 48 49 50 51 GitHub - Nix-QChem/NixOS-QChem: Nix expressions for HPC/Quantum chemistry software packages
https://github.com/Nix-QChem/NixOS-QChem

53 54 Nuenv: an experimental Nushell environment for Nix
https://determinate.systems/blog/nuenv/