Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

Nix и NixOS: декларативное управление серверами

Переводим Linux-парк на NixOS 26.05: flake-монорепозиторий, атомарный деплой и откат за секунды

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

У заказчиков с парком 10–40 Linux-серверов конфигурация живёт в головах и разрозненных скриптах: ручные правки создают дрейф, apt upgrade в разное время даёт разные версии библиотек, новый сервер «по той же инструкции» работает иначе, а откат неудачного обновления занимает часы. Мы закрываем это архитектурно: вся ОС описывается декларативно, собирается из git и откатывается ат…

Почему это важно бизнесу

- Простой после неудачного обновления сокращается с часов ручного разбора до секунд отката поколения
- Новый сервер или замена сгоревшего — 15 минут сборки из git, а не день ручной настройки
- Инфраструктура не завязана на «инженера, который помнит»: всё состояние — в репозитории
- Каждое изменение проходит через pull request: видно, кто, что и когда поменял на серверах

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

26.05

Стабильный релиз NixOS «Yarara»: пиним в flake.lock, security-фиксы 7 месяцев до 31.12.2026

nixos.org, релиз 30.05.2026

5 с

Откат конфигурации: nixos-rebuild switch --rollback переключает симлинк поколения атомарно

наш стандарт эксплуатации

17

PostgreSQL по умолчанию при stateVersion >= 25.11 — версию всегда пиним явно pkgs.postgresql_17

по докам NixOS 25.11/26.05

30 дней

Глубина хранения поколений: nix.gc.automatic + dates=weekly + --delete-older-than 30d, иначе с…

наш стандарт

10

boot.loader.systemd-boot.configurationLimit: дефолт — без лимита; 10 поколений в boot-меню хва…

наш стандарт

15 мин

Ввод нового узла: nixos-anywhere + disko от пустого VPS до боевой конфигурации из git

наш стандарт внедрения

РЕАЛИЗАЦИЯ 1 ИЗ 3

Flake-монорепозиторий: единый источник истины для парка

Что настраиваем

Scope: git-репо infra на весь парк; hosts/<имя>/configuration.nix плюс общие modules/base.nix, monitoring.nix и роли

Как мы это делаем

1. flake.nix: input nixpkgs = github:NixOS/nixpkgs/nixos-26.05, outputs — nixosConfigurations по хостам через nixpkgs.lib.nixosSystem
2. modules/base.nix: services.openssh.settings.PasswordAuthentication = false, networking.firewall.allowedTCPPorts, users с ssh-ключами, zabbix-агент
3. Гигиена стора: nix.gc.automatic = true, dates = "weekly", options = "--delete-older-than 30d"; nix.settings.auto-optimise-store = true
4. CI на каждый PR: nix flake check + nix build .#nixosConfigurations.<host>.config.system.build.toplevel — битая конфигурация не доходит до прода
5. Деплой на парк: colmena apply или deploy-rs с magic-rollback — узел сам откатится, если после активации потерял связь

РЕЗУЛЬТАТ

Любой инженер собирает бит-в-бит ту же систему из flake.lock; ручной дрейф исключён — всё, что не описано в модулях, следующий switch перезапишет. Изменения проходят ревью в PR, история инфраструктуры равна git log.

КЛЮЧЕВОЙ НЮАНС

Каналы nix-channel на хостах отключаем полностью: два источника пакетов — это снова дрейф. Единственный вход — flake.lock; обновляем его отдельным PR через nix flake update с прогоном CI.

РЕАЛИЗАЦИЯ 2 ИЗ 3

Типовой сервер приложений: nginx, PostgreSQL 17, секреты

Что настраиваем

Scope: роль web/db — reverse-proxy с ACME, СУБД с закреплённой версией, секреты через sops-nix, декларативные бэкапы

Как мы это делаем

1. services.nginx.virtualHosts: forceSSL + enableACME, security.acme.acceptTerms = true — сертификат Let's Encrypt выпускается и продлевается сам
2. services.postgresql: package = pkgs.postgresql_17 явно (дефолт зависит от stateVersion), settings — shared_buffers и max_connections под RAM узла
3. Секреты: sops-nix, шифрование age-ключом, производным от ssh-ключа хоста; расшифровка при активации в /run/secrets с правами 0400 на юзера сервиса
4. Бэкапы данных: services.postgresqlBackup.enable + services.restic.backups с passwordFile из sops — конфиг откатывается, данные защищены отдельно

РЕЗУЛЬТАТ

Роль воспроизводится на любом узле: тот же nginx, та же мажорная версия PostgreSQL, те же лимиты. Мажорный апгрейд СУБД — осознанное изменение в PR с дампом «до», а не сюрприз от пакетного менеджера в случайный момент.

КЛЮЧЕВОЙ НЮАНС

Всё, что попадает в /nix/store, читаемо любым процессом хоста — пароли и ключи туда класть нельзя. В git храним только зашифрованные sops-файлы и проверяем PR на открытые секреты.

РЕАЛИЗАЦИЯ 3 ИЗ 3

Ввод узла с нуля и атомарный откат

Что настраиваем

Scope: голый VPS или железо → боевой узел; установка nixos-anywhere по SSH, разметка диска декларативно через disko

Как мы это делаем

1. disko-конфиг в репозитории: GPT, ESP 512M, ext4/ZFS — разметка описана кодом рядом с configuration.nix, никакого ручного fdisk
2. nixos-anywhere --flake .#new-host root@IP: kexec в инсталлер, разметка, установка и первая активация одним прогоном
3. Проверка до прода: nixos-rebuild build-vm — конфигурация поднимается в локальной QEMU-VM, смоук-тест сервисов
4. Обновления ядра/initrd: nixos-rebuild boot + плановый reboot; switch применяет только userspace — фиксируем это в runbook
5. Откат: nixos-rebuild switch --rollback, а при неживом SSH — выбор предыдущего поколения в меню systemd-boot (configurationLimit = 10)

РЕЗУЛЬТАТ

Замена умершего сервера — 15 минут от чистой машины до узла, идентичного погибшему на уровне closure. Неудачная выкладка откатывается за секунды целиком: версии пакетов, конфиги сервисов и systemd-юниты.

КЛЮЧЕВОЙ НЮАНС

Поколение откатывает систему, но не данные: /var/lib остаётся как есть. Перед мажорными изменениями stateful-сервисов снимаем дамп; откат БД — из бэкапа, а не из загрузчика.

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

✕ Секреты в /nix/store

Стор читаем всеми процессами хоста. Храним через sops-nix/agenix: в git — шифртекст, на хосте — /run/secrets с правами 0400 на юзера сервиса.

✕ stateVersion ≠ версия системы

Это маркер формата данных сервисов, а не апгрейд ОС. Смена меняет дефолты (например, версию PostgreSQL) — на живых хостах не трогаем без release note…

✕ Откат не возвращает данные

Rollback меняет пакеты, конфиги и юниты; /var/lib нетронут. Перед мажором СУБД — pg_dump/restic, миграцию данных планируем отдельным шагом.

✕ /nix/store съедает диск

Каждое поколение держит свой closure. Без nix.gc.automatic weekly с --delete-older-than 30d и auto-optimise-store стор растёт до заполнения раздела.

✕ Flake не видит новый файл

Flakes копируют только tracked-файлы git: свежий модуль даёт «path does not exist». Лечится git add до сборки — закрепляем в привычку и в CI.

✕ switch не обновил ядро

Новое ядро и initrd активируются только после перезагрузки. Для kernel-обновлений используем nixos-rebuild boot и плановое окно reboot.

✕ nix-channel рядом с flakes

Второй источник nixpkgs возвращает дрейф версий между хостами. Каналы удаляем, пин версий — только через flake.lock в репозитории.

✕ Глобальный allowUnfree

Разрешение всего несвободного скрывает, что реально тянется в систему. Используем allowUnfreePredicate со списком конкретных пакетов.

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- Nix поверх текущего дистрибутива: nix shell/develop для окружений инженеров
- configuration.nix в git; деплой nixos-rebuild switch --target-host по SSH
- nix.gc.automatic + configurationLimit 10 — диск и boot-меню под контролем

НОРМАЛЬНО

- Flakes с пином nixos-26.05; flake.lock в git, обновление отдельным PR
- CI-сборка toplevel каждого хоста до деплоя (nix flake check)
- Секреты только через sops-nix — ноль открытых паролей в репозитории
- Деплой парка colmena/deploy-rs с авто-откатом при потере связи

ХОРОШО

- nixos-anywhere + disko: узел с нуля одной командой, разметка кодом
- Свой бинарный кеш (harmonia/attic): собираем раз, узлы тянут готовое
- nixos-rebuild build-vm и NixOS-тесты сервисов до выкладки в прод
- Impermanence: корень в tmpfs, персистентны только /nix и /persist

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Вся конфигурация серверов лежит в git и собирается из flake.lock без ручных шагов?
- [ ] Откат последнего изменения занимает секунды (switch --rollback), а не часы?
- [ ] Секреты зашифрованы (sops-nix/agenix) и не лежат в /nix/store открытым текстом?
- [ ] Настроена автоочистка поколений (nix.gc) и место в /nix/store под контролем?
- [ ] Версии PostgreSQL и других сервисов пинятся явно, а не тянутся дефолтом stateVersion?
- [ ] Есть CI-проверка nix build ...toplevel до деплоя на боевые узлы?
- [ ] Стратегия бэкапов данных в /var/lib отделена от отката конфигурации?
- [ ] Переход на новый релиз NixOS проходит по release notes и сначала на стенде?

Если хотя бы на два вопроса ответ «нет» или «не знаю» — тема требует внимания.

ЧТО МЫ ДЕЛАЕМ

Как поможет ITFresh

ITFresh — ИТ-аутсорсинг для юридических лиц до 50 рабочих мест в Москве и области. 15+ лет практики, собственная инфраструктура в дата-центре МТС (8 серверов Dell Xeon Platinum).

- Аудит парка и пилот: переводим 2–3 сервера на NixOS 26.05 без остановки сервисов
- Проектируем flake-монорепозиторий: модули ролей, секреты sops-nix, CI-сборку
- Внедряем деплой colmena/deploy-rs и ввод узлов через nixos-anywhere + disko
- Обучаем инженеров заказчика языку Nix и передаём runbook эксплуатации
- Берём NixOS-парк на поддержку: обновления релизов, мониторинг, бэкапы

15+
лет в ИТ-поддержке

50
рабочих мест — наш профиль

МТС
дата-центр, Москва

КОНТАКТЫ

Обсудить вашу задачу

Сайт	**itfresh.ru**
Телефон	**+7 903 729-62-41**
Telegram	**@ITfresh_Boss**

Бесплатно посмотрим вашу инфраструктуру по этому чек-листу и скажем, где тонко — без обязательств.

itfresh.ru

ДОКУМЕНТАЦИЯ И НАШ СТЕК

Техническая база

01 NixOS Manual — Configuration, nixos-rebuild (nixos.org — 26.05)

02 NixOS 26.05 «Yarara» Release Notes (nixos.org — 2026)

03 Nix Reference Manual — Flakes, nix flake (nix.dev — 2.32)

04 NixOS Options Search (services.*, nix.gc) (search.nixos.org — 26.05)

05 sops-nix — управление секретами NixOS (github.com/Mic92 — 2026)

06 nixos-anywhere и disko — документация (github.com/nix-community — 2026)

07 Наш шаблон flake-монорепо (base/roles/hosts) (наш стандарт — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.