OSTree в НАЙС.ОС CLOUD

Часть 1. Что такое OSTree и зачем он нужен в НАЙС.ОС

1. Введение

В инфраструктурной ОС важны две вещи: **предсказуемость** и **воспроизводимость**. Сервер, который “вчера работал”, а “сегодня после обновления внезапно стал другим”, — это не приключение, а регулярный источник аварий, простоя и ручных “исцелений на месте”. Подход OSTree в НАЙС.ОС решает эту проблему не “магией”, а дисциплиной: система обновляется как единое целое, а не как поток независимых пакетных изменений.

Данный раздел объясняет, что такое OSTree, какие задачи он закрывает и почему он уместен для НАЙС.ОС — без ухода в низкоуровневые детали сборки и публикации (они будут в технических документах).

Коротко: идея OSTree

OSTree — это способ хранить и доставлять состояние операционной системы как **версионируемые “снимки”**, а развёртывание и обновления выполнять через **deploy** (развёртывание коммита) с возможностью **быстрого отката** на предыдущую рабочую версию.

2. Проблема классических обновлений

В традиционной модели администрирования обновление ОС — это, по сути, множество независимых изменений: обновились пакеты, изменились зависимости, поменялись конфиги по умолчанию, подтянулись новые версии библиотек, изменилось поведение сервисов. В результате “обновление” становится событием со скрытой вариативностью.

2.1 Типовые симптомы (на практике)

- **Невоспроизводимость:** два одинаковых узла, обновлённых “примерно одинаково”, через месяц оказываются разными.
- **Трудный откат:** откатывать приходится пакеты “по одному”, часто без уверенности, что вернётся исходное состояние.
- **Неявные зависимости:** изменение одной библиотеки может сломать приложение, которое формально “не трогали”.
- **Дрейф конфигурации:** даже при аккуратной работе руками система постепенно уходит от “эталона”.

Важно понимать границу ответственности

Конфигурационный менеджмент (Ansible/Salt и т.п.) решает вопрос “как привести конфиги к желаемому виду”. OSTree решает другую задачу: **как гарантировать, что базовая ОС как набор файлов и версий — именно та, что ожидали**. Эти подходы не конкурируют — они дополняют друг друга.

3. OSTree простыми словами

OSTree удобно представлять как механизм, который хранит систему в виде **коммитов** — фиксированных наборов файлов (в первую очередь системной части), а обновление превращает в действие вида: “получить новый коммит” → “развернуть” → “переключиться на него при следующей загрузке”.

3.1 Аналогия, которая помогает (и где она ломается)

Часто говорят “OSTree — это Git для /usr”. Эта аналогия полезна, чтобы почувствовать идею коммитов и истории, но нужно помнить: OSTree не предназначен для ежедневного ручного редактирования “как рабочей директории”. Он про **поставку и управление версиями базовой ОС**.

3.2 Основные термины (минимально необходимый набор)

- **commit** — зафиксированная версия дерева файлов системы (состояние ОС в конкретный момент).
- **ref** — “ветка/канал”, указывающая на коммит (пример: niceos/5.2/x86_64/minimal).
- **remote** — удалённый источник, откуда можно получать коммиты (репозиторий OSTree).

- **deploy** — развёртывание выбранного коммита в sysroot так, чтобы его можно было загрузить.
- **rollback** — переключение на предыдущий deploy (обычно через выбор в загрузке или команду администрирования).

Что “атомарно” в этой модели

В атомарной модели важна не “магическая неубиваемость”, а отсутствие полу-состояний: либо система работает на версии A, либо на версии B. Обновление не превращает узел в смесь “половина старого, половина нового” — оно готовит новую версию как отдельный deploy.

4. Где тут rpm-ostree и зачем он нужен

OSTree отвечает за хранение и доставку “снимков” (коммитов) и управление deploy/rollback. Но сами “снимки” нужно где-то брать. В экосистемах, где базой являются RPM-пакеты, роль “сборщика” выполняет rpm-ostree.

В практическом смысле rpm-ostree позволяет описать систему декларативно через **treefile** (JSON): какие репозитории пакетов использовать, какие пакеты включать, какие units включить, какие параметры применить. Результат — готовый OSTree-коммит.

```
{
  "osname": "niceos",
  "releasever": "5.2",
  "ref": "niceos/5.2/x86_64/minimal",
  "repos": ["niceos", "niceos-updates", "niceos-extras"],
  "packages": ["bash", "systemd", "openssh", "rpm-ostree"]
}
```

Дисциплина treefile

Treefile — это контракт состава системы. Чем стабильнее и прозрачнее этот контракт, тем проще поддерживать узлы в одинаковом состоянии. “Добавим пакет руками на одном сервере” — это уже отклонение от контракта и потенциальная причина расхождений между узлами.

5. Практическая польза: что выигрывает администратор

5.1 Предсказуемость обновлений

Обновление превращается из “цепочки пакетных изменений” в получение готового коммита из репозитория. Если два узла получают один и тот же ref, они получают **одинаковую базовую систему**. Это снижает вероятность “у меня работает, у тебя нет” без объективной причины.

5.2 Быстрый откат

При проблеме после обновления не требуется разбирать “какой пакет сломал”. В регламентируемом контуре чаще важнее быстро восстановить сервис. OSTree даёт возможность откатиться на предыдущую рабочую версию как на отдельный deploy.

5.3 Контроль и аудит

Для эксплуатации и аудита важно уметь ответить на вопрос: **что именно было развёрнуто**. В OSTree-модели ответом является ref и коммит, на который он указывает. Это даёт основу для отчётности, повторяемости выпусков и упрощает разбор инцидентов.

6. Минимальный “мысленный” сценарий применения в НАЙС.ОС

В типовом жизненном цикле “сервер клиент” происходит следующее: на сервере сборки формируется OSTree-репозиторий и коммит по ref, после чего репозиторий публикуется. На стороне клиента выполняется pull выбранного ref и deploy в sysroot, а загрузка настроена так, чтобы система стартовала с развёрнутого deploy.

```
# концептуально (без привязки к конкретным скриптам):
# сервер: compose -> repo + summary
# клиент: remote add -> pull ref -> deploy -> reboot
```

Часть 2. Как OSTree устроен в НАЙС.ОС: архитектура и

жизненный цикл

1. Модель поставки НАЙС.ОС через OSTree

В НАЙС.ОС OSTree используется как “магистраль поставки” базовой системы: есть **репозиторий** (на сервере), в нём есть **refs** (каналы), а клиентская сторона выполняет **pull** и **deploy** выбранного ref. В результате получаем управляемую модель: выпуск публикация развёртывание обновление откат.

Ключевая идея: ref как “канал релиза”

В OSTree моделью управления становится ref. Это удобнее, чем “версия пакета X”, потому что ref описывает **согласованный набор** компонентов, который уже проверен как целое. Клиенту не нужно угадывать, какие зависимости совместимы — он получает готовую версию системы.

2. Что считается “образом” в этой модели

В практической эксплуатации часто смешивают понятия “установочный ISO” и “образ системы”. В OSTree-модели “образ” — это чаще всего: **дисковый образ** (RAW/QCOW2/VMDK), содержащий разметку и загрузчик, и внутри — выполненный deploy нужного ref. Такой образ удобно импортировать в виртуализацию или использовать как эталон для тиражирования.

При этом “содержимое системы” определяется не тем, что “поставили руками”, а тем, какой ref был развёрнут. Это и есть механизм обеспечения одинаковости узлов.

3. Поток “сервер клиент” в НАЙС.ОС

3.1 Сервер: сборка дерева (compose) как декларация состава

На серверной стороне ключевым входом является **treefile** (JSON). Он описывает, какая именно система должна получиться: источники пакетов (repos), набор пакетов (packages), параметры сборки и сервисы (units). Из этого описания сборщик (rpm-ostree) строит OSTree-коммит.

```
{
  "osname": "niceos",
```

```
  "releasever": "5.2",
  "ref": "niceos/5.2/x86_64/minimal",
  "repos": ["niceos", "niceos-updates", "niceos-extras"],
  "packages": ["systemd", "openssh", "rpm-ostree"],
  "units": ["sshd.service"]
}
```

На уровне “механики” сервер выполняет: rpm-ostree compose tree запись коммита в OSTree-репо обновление summary. Именно summary делает репозиторий “удобоваримым” для клиентов: refs обнаруживаются корректно и быстро.

```
# концептуально (серверная сторона):
rpm-ostree compose tree --unified-core --repo=/srv/ostree/niceos/repo /srv/ostree/niceos/niceos-base.json
ostree summary --repo=/srv/ostree/niceos/repo --update
```

Почему “summary” — обязательный шаг

Клиенту недостаточно “где-то лежат коммиты”. Репозиторий должен быть пригоден для обнаружения refs и получения метаданных. Поэтому после каждого compose обновление summary должно считаться частью регламента выпуска.

3.2 Публикация репозитория

Для потребления репозитория на клиенте нужно опубликовать данные репозитория по HTTP(S) (обычно каталог вида .../repo). Важно понимать, что публикуется именно **OSTree-данные**, а не “весь рабочий каталог сборки”.

- **Публикуется:** ${REPOPATH}/repo
- **Не публикуется:** ${REPOPATH}/cache (кэш сборки — внутренний артефакт)

3.3 Клиент: pull deploy загрузка

На клиентской стороне жизненный цикл простой и управляемый: добавляется remote (источник), выполняется pull выбранного ref, затем выполняется deploy, который создаёт готовое дерево файлов и записи для загрузчика. После перезагрузки система стартует в новой версии.

```
# концептуально (клиентская сторона):
ostree remote add --set=gpg-verify=false niceos http://OSTREE_SERVER/repo
ostree pull niceos niceos/5.2/x86_64/minimal
```

```
ostree admin deploy --os=niceos niceos:niceos/5.2/x86_64/minimal
```

4. Из чего состоит sysroot и почему это важно

В OSTree-модели существует “контейнер” для развёртываний — **sysroot**. Это структура на диске, где хранится: (а) репозиторий OSTree, (б) развёрнутые деплои, (в) элементы загрузки. Администратору не нужно помнить все внутренности, но полезно знать “опорные точки”.

- /ostree — служебные данные OSTree и развёртывания (deployments).
- /boot — место хранения элементов загрузки; в OSTree-схеме там живут loader entries.
- /boot/loader/entries/ — записи загрузки (каждая соответствует deploy).

Почему это полезно администратору

Если узел не грузится после обновления, чаще всего проблема не “в OSTree”, а в том, что загрузчик не находит root, или неверно выбран deploy. Понимание, где лежат loader entries и как связан deploy с загрузкой, резко ускоряет диагностику.

5. Как НАЙС.ОС оформляет “ветки” и профили (refs)

Практически удобная схема — использовать ref как комбинацию: **версия релиза архитектура профиль**. Например: niceos/5.2/x86_64/minimal, niceos/5.2/x86_64/base, niceos/5.2/x86_64/security.

Такой подход позволяет иметь несколько “каналов” с разным назначением, не смешивая их в одном “плавающем” наборе пакетов. Клиент выбирает ref осознанно, а обновления происходят внутри выбранного канала.

6. Безопасность поставки: что важно заложить в процесс

OSTree делает поставку системы управляемой, но безопасность зависит от того, **как именно** организована цепочка доверия: источники RPM, верификация подписей, транспорт публикации, контроль доступа к репозиторию.

- **Источники пакетов:** *.repo должны быть контролируемыми и стабильными

(минимизировать “дрейф”).
- **Подписи:** при gpgcheck=1 в RPM-репозиториях обеспечить ключи и доверие; для OSTree — применять политику gpg-verify на remote.
- **Транспорт:** предпочтительно HTTPS или закрытый контур (VPN/приватная сеть) для доступа клиентов к репозиторию.
- **Права на репозиторий:** запись в каталог данных репо — только у сборочного процесса; чтение — у веб-сервера/клиентов.

Смысл “атомарности” не спасает от плохой гигиены

Если репозиторий доступен на запись посторонним или источники пакетов не контролируются, OSTree не может “волшебно” гарантировать происхождение коммитов. Атомарность — это про предсказуемость результата, а не про безопасность без регламента.

Часть 3. Когда OSTree — лучший выбор, а когда нужен другой подход

1. Зачем вообще выбирать “атомарную” модель

Выбор OSTree — это выбор в пользу **управляемого жизненного цикла базовой ОС**. Он особенно хорошо работает там, где важны повторяемость, возможность быстро откатиться и отсутствие “дрейфа” между узлами. При этом OSTree не отменяет администрирование, конфигурацию и прикладные сервисы — он фиксирует именно “скелет” системы.

Опорный критерий

OSTree уместен там, где вопрос “что именно развёрнуто?” должен иметь точный и проверяемый ответ: **ref** + **commit** вместо “примерно такие-то пакеты”.

2. Сценарии, где OSTree в НАЙС.ОС даёт максимальный эффект

2.1 Облачные ВМ и шаблоны (тиражирование и однообразие)

В облаках и виртуализации часто нужно быстро создавать много одинаковых узлов:

NAT-инстансы, сенсоры, лог-агенты, минимальные серверы под сервисы. В классической модели “установка + обновления + ручные настройки” приводит к тому, что узлы начинают отличаться уже через несколько недель.

В OSTree-модели узлы получают один и тот же ref. Это означает, что базовая ОС одинаковая “побайтно” в тех частях, которые определяются коммитом. Администратор получает стандартизируемую основу для дальнейшей конфигурации.

2.2 Edge и удалённые площадки (нужен быстрый откат)

На удалённых площадках цена ошибки выше: физический доступ ограничен, окна обслуживания редкие, а восстановление “ручной магией по SSH” не всегда возможно. OSTree снижает риск за счёт простой стратегии: обновление готовит новый deploy, а при проблеме можно вернуться на предыдущий.

2.3 Регламентированные контуры и аудит

В регламентированных средах часто важнее не “обновиться любой ценой”, а иметь трассируемость: что установлено, когда выпущено, на каком основании. OSTree удобно ложится на контроль: выпуск оформляется как commit в конкретном ref, а на узле можно проверить, какая версия активна.

2.4 Системы безопасности и наблюдаемости (стабильная база)

Сенсоры, IDS/NDR, сборщики логов и агентские компоненты любят стабильность: меньше внезапных несовместимостей библиотек, меньше “починилось само и сломалось само”. OSTree даёт предсказуемую основу, поверх которой разворачиваются профильные сервисы НАЙС.ОС.

Практическая формула

Чем больше у вас узлов и чем больше стоимость простоя, тем сильнее окупается модель “выпуск как единое целое” и “откат как штатная операция”.

3. Когда OSTree может быть излишним или требует дисциплины

3.1 “Я хочу постоянно менять базовые пакеты руками”

Если эксплуатационная модель — это постоянная ручная доустановка/удаление

пакетов на каждом узле, OSTree будет восприниматься как “ограничение”. Это не баг — это свойство: базовая система должна быть определена выпуском, а не импровизацией на каждом сервере.

Правильный подход в OSTree-модели — менять состав системы через treefile и выпускать новый commit, а не “допиливать” каждый узел отдельно.

3.2 Большие локальные кастомизации: где граница

На практике всегда есть данные, конфиги и сервисы, которые живут “поверх” базовой ОС. Важно провести границу: **база** — через OSTree (что должно быть одинаковым), **конфигурация** — через отдельный механизм (Ansible/скрипты/политики), **данные** — в отдельных каталогах/томах.

Антипаттерн

Если разные узлы начинают отличаться “внутри базы” (разный набор системных пакетов/библиотек), вы теряете главный смысл OSTree: одинаковость и проверяемость состояния.

3.3 Миграции между средами: BIOS/UEFI, virtio/SATA, fstab

OSTree управляет поставкой системы, но загрузка всё равно зависит от дисковой схемы и среды. При переносе образов между платформами важно учитывать:

- **BIOS vs UEFI:** профили образов должны быть отдельными (разная разметка, ESP, параметры установки GRUB).
- **virtio vs SATA/SCSI:** изменяется имя устройства (/dev/vda vs /dev/sda), а значит требуется согласованность root= и записей загрузчика.
- **fstab:** если в /etc/fstab указаны конкретные устройства, при смене типа диска потребуется корректировка (или переход на UUID/labels как правило эксплуатации).

4. Рекомендованный “путь внедрения” OSTree в НАЙС.ОС

4.1 Начать с пилота (минимальный риск)

Для внедрения рекомендуется начать с пилотного стенда: один ref (например, minimal), один репозиторий, одна-две ВМ. Цель пилота — подтвердить

предсказуемость обновлений и откатов, и отработать публикацию репозитория.

4.2 Зафиксировать правила (до масштабирования)

До масштабирования важно зафиксировать эксплуатационные правила: какие refs существуют, как именуются, кто и как выпускает коммиты, как публикуется репозиторий, как проверяются подписи и кто имеет право записи в репозиторий.

Что должно быть “в регламенте”

- Схема refs: версия архитектура профиль.
- Процесс выпуска: compose тест публикация контроль summary.
- Процесс обновления узлов: кто, когда, как откатывает.
- Политика доверия: подписи, ключи, транспорт (HTTPS/закрытый контур).
- Разделение ответственности: база (OSTree) vs конфиги (CM) vs данные (тома/каталоги).

5. Следующие шаги (переход к техническим документам)

После понимания концепции переходят к технической реализации: (1) создание и сопровождение OSTree-репозитория на сервере; (2) подготовка дискового образа/установки на клиенте. В НАЙС.ОС эти шаги удобно оформлены отдельными регламентами и автоматизированы скриптами.

```
# связка документов/скриптов:
# 1) сервер: mkostreerepo — создает/обновляет OSTree repo и summary
# 2) клиент/образ: mk-ostree-host.sh — готовит дисковый образ и выполняет deploy выбранного ref
```