Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

Drone CI в Docker: свой CI/CD для команды из 5-10 человек

Как мы поднимаем сервер и docker-раннер, закрываем секреты и собираем пайплайн сборки

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

Команда из 5-10 разработчиков выкатывает руками: сборка на ноутбуке, деплой по SSH, «у меня локально работает». Jenkins обслуживать некому, облачный CI упирается в минуты и вынос кода наружу. Мы ставим self-hosted Drone: сервер и раннер — два контейнера на Docker-хосте, пайплайн описан файлом .drone.yml в репозитории. Пока этого нет, релиз зависит от одного человека.

Почему это важно бизнесу

- Ручная выкатка по SSH — это простой сервиса и откат «по памяти»: клиент видит ошибку раньше, чем вы.
- Знание «как собрать и задеплоить» живёт в голове одного инженера — отпуск или увольнение останавливают релизы.
- Без автотестов в конвейере баг доезжает до прода и стоит в разы дороже, чем найденный на коммите.
- Self-hosted CI держит исходники и креды в вашем контуре — не в чужом облаке и не в личных ноутбуках.
- Два контейнера на существующей виртуалке дешевле подписки на облачный CI с оплатой за минуты сборки.

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

2.28.2

тег drone/drone, который мы фиксируем в compose вместо latest: обновляем осознанно

drone/drone 2.28.2, апрель 2026

1.8.5

тег drone/drone-runner-docker: раннер версионируем отдельно от сервера, откат — сменой тега

drone-runner-docker 1.8.5, январь 2026

2

DRONE_RUNNER_CAPACITY по умолчанию — столько пайплайнов раннер крутит параллельно

по докам docker runner

60 мин

таймаут пайплайна на репозиторий по умолчанию; меняет только администратор Drone

по докам drone server

24h

DRONE_CLEANUP_INTERVAL по умолчанию: реапер добивает задачи, зависшие в pending и running

по докам drone server

9.6+

минимальная версия Postgres под базу Drone; на ней держим историю сборок и секреты

по докам drone server

РЕАЛИЗАЦИЯ 1 ИЗ 3

Сервер Drone: OAuth, Postgres, шифрование секретов

Что настраиваем

Один Docker-хост Ubuntu 24.04 или Debian 12: контейнер drone/drone:2.28.2 за nginx или Traefik, домен ci.company.ru

Как мы это делаем

1. OAuth2-приложение на Git-хостинге (в Gitea: Site Administration -> Applications), отмечаем Confidential Client, callback — ровно https://ci.company.ru/login.
2. Окружение: DRONE_GITEA_SERVER, DRONE_GITEA_CLIENT_ID и _SECRET, DRONE_SERVER_HOST=ci.company.ru, DRONE_SERVER_PROTO=https, DRONE_RPC_SECRET из openssl rand -hex 16.
3. Уходим со встроенной sqlite: DRONE_DATABASE_DRIVER=postgres (по доке 9.6 и выше), DRONE_DATABASE_DATASOURCE=postgres://user:pass@host:5432/drone?s
4. DRONE_DATABASE_SECRET задаём ДО первого использования: по докам шифрование секретов в БД выключено по умолчанию и настраивается только заранее.
5. Периметр: DRONE_REGISTRATION_CLOSED=true, белый список DRONE_USER_FILTER, первый админ через DRONE_USER_CREATE=username:<логин>,admin:true, TLS на прокси.

РЕЗУЛЬТАТ

Сервер поднимается за один сеанс и переживает перезапуск хоста: конфигурация в docker-compose.yml и .env, история сборок и секреты в Postgres, а не в томе контейнера. Логин — только для сотрудников из белого списка, снаружи открыт лишь 443 через прокси, порт панели не проброшен.

КЛЮЧЕВОЙ НЮАНС

Два решения нельзя отложить «на потом»: шифрование секретов включается до первого запуска, а внешний адрес зашит в DRONE_SERVER_HOST и в callback OAuth — их меняют только вместе.

РЕАЛИЗАЦИЯ 2 ИЗ 3

Docker-раннер: ёмкость, лимиты и уборка образов

Что настраиваем

Тот же хост или отдельная VM под сборки: drone/drone-runner-docker:1.8.5 с монтированием /var/run/docker.sock

Как мы это делаем

1. Связь с сервером: DRONE_RPC_HOST, DRONE_RPC_PROTO и тот же DRONE_RPC_SECRET; при расхождении секрета раннер просто не берёт задачи, а сборки висят в pending.
2. Ёмкость: DRONE_RUNNER_CAPACITY (по умолчанию 2) выставляем из расчёта один пайплайн на 2 vCPU, площадку помечаем DRONE_RUNNER_NAME и DRONE_RUNNER_LABELS.
3. Тормозим прожорливые шаги: DRONE_MEMORY_LIMIT (байты на контейнер шага) и DRONE_CPU_QUOTA (мкс CFS-периода), чтобы npm build не выдавил соседние контейнеры.
4. Ставим на хост cron с docker system prune -af --filter until=72h: слои каждой сборки иначе съедают /var/lib/docker и роняют весь Docker.
5. Флаг trusted у репозитория выдаём точечно — только он открывает host-volume и privileged-шаги; для остальных репозиториев это остаётся закрытым.

РЕЗУЛЬТАТ

Раннер перестаёт быть источником сюрпризов: сборки не конкурируют за CPU и память с прод-сервисами, диск не заканчивается ночью в пятницу, привилегированные шаги разрешены лишь паре доверенных репозиториев. Масштабирование — второй раннер с тем же RPC-секретом.

КЛЮЧЕВОЙ НЮАНС

Раннеру отдан сокет Docker-демона, то есть фактически права на хост. Поэтому сборочный узел мы держим отдельно от прод-нагрузки, а trusted-флаг раздаём поштучно.

РЕАЛИЗАЦИЯ 3 ИЗ 3

Пайплайн .drone.yml: тесты, образ, деплой, кэш

Что настраиваем

Репозиторий проекта: kind pipeline, type docker, шаги tests -> build-image -> deploy с условиями по ветке и событию

Как мы это делаем

1. Workspace — общий том, который монтируется в каждый шаг в /drone/src: артефакты между шагами передаём файлами, без внешних хранилищ и хаков.
2. Сборка образа — плагин plugins/docker: settings registry, repo, tags; ускоряем через cache_from и build_args, для проверки без публикации есть dry_run.
3. Все креды только через from_secret; флаг Allow Pull Requests у секрета оставляем выключенным — по умолчанию Drone в PR секреты не отдаёт, и это правильно.
4. Кэш: temp-volume живёт лишь внутри одного пайплайна, поэтому .venv и node_modules кладём в host-volume доверенного репозитория или в meltwater/drone-cache.
5. Разводим контуры условиями when по branch и event: тесты гоняем на pull_request, публикацию образа и деплой — только с main.

РЕЗУЛЬТАТ

Пайплайн лежит в репозитории и версионируется вместе с кодом: любой разработчик видит, как собирается и куда едет его коммит. Тесты отсекают заведомо битые изменения до сборки образа, а деплой перестаёт быть ручной операцией по SSH и занимает минуты вместо получаса согласований.

КЛЮЧЕВОЙ НЮАНС

Главная экономия времени — не в железе, а в кэше и слим-образах: без кэша каждый шаг заново тянет зависимости, и пайплайн упирается в сеть, а не в CPU.

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- ✕ **Плавающий тег образа в compose**
  latest или drone/drone:2 при первом pull притянут другую сборку. Фиксируем 2.28.2 и раннер 1.8.5, обновляем осознанно и по одному.

- ✕ **RPC-секрет разошёлся**
  Раннер не регистрируется, сборки висят в pending без внятной ошибки. Секрет генерируем раз (openssl rand -hex 16) и берём из .env в оба сервиса.

- ✕ **OAuth callback mismatch**
  Callback URL должен совпадать с DRONE_SERVER_PROTO и DRONE_SERVER_HOST плюс путь /login. Опечатка в протоколе или лишний слэш ломают вход.

- ✕ **Шифрование секретов включили поздно**
  DRONE_DATABASE_SECRET настраивается до первого использования системы. На живой базе включение делает ранее сохранённые секреты нечитаемыми.

- ✕ **sqlite оставили на боевом сервере**
  Встроенная база живёт в томе контейнера и уезжает вместе с ним. Переводим на Postgres 9.6+ через DRONE_DATABASE_DRIVER и включаем в бэкап.

- ✕ **Не убирается /var/lib/docker**
  Слои сборок копятся, диск заканчивается и падает весь Docker, а не только CI. Лечится cron с docker system prune и алертом на свободное место раздела.

- ✕ **Секреты открыты для pull request**
  Включённый Allow Pull Requests позволяет вытащить значение из форк-PR произвольным шагом. Флаг держим выключенным, исключения — по заявке.

- ✕ **Регистрация в Drone открыта всем**
  Любая учётка Git-хостинга логинится и активирует репозитории. Закрываем DRONE_REGISTRATION_CLOSED=true и списком DRONE_USER_FILTER.

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- Сервер и раннер в docker-compose, версии образов зафиксированы тегами
- TLS и домен на reverse proxy, порт панели наружу не проброшен
- Секреты только через from_secret, Allow Pull Requests выключен
- Cron docker system prune и алерт на свободное место раздела Docker

НОРМАЛЬНО

- Postgres 9.6+ вместо sqlite, база в общем графике резервного копирования
- DRONE_DATABASE_SECRET задан до первого запуска системы
- DRONE_REGISTRATION_CLOSED и DRONE_USER_FILTER под список сотрудников
- DRONE_RUNNER_CAPACITY, DRONE_MEMORY_LIMIT и DRONE_CPU_QUOTA под vCPU хоста

ХОРОШО

- Отдельная VM под раннеры, прод-нагрузка не делит хост со сборками
- Второй раннер с DRONE_RUNNER_LABELS для тяжёлых и параллельных пайплайнов
- Общий шаблон .drone.yml и prebuilt-образ с зависимостями на все репозитории
- Кэш через host-volume или meltwater/drone-cache, восстановление CI отработано

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Теги образов drone/drone и drone-runner-docker зафиксированы в compose, а не latest?
- [ ] DRONE_RPC_SECRET одинаков на сервере и на всех раннерах и хранится вне репозитория?
- [ ] Callback URL в OAuth-приложении совпадает с DRONE_SERVER_PROTO, DRONE_SERVER_HOST и путём /login?
- [ ] DRONE_DATABASE_SECRET задан до первого использования, секреты в базе шифруются?
- [ ] База переведена на Postgres и попадает в регулярный бэкап вместе с .env и compose?
- [ ] Регистрация закрыта, доступ ограничен DRONE_USER_FILTER, админ-учётка именная?
- [ ] DRONE_RUNNER_CAPACITY, DRONE_MEMORY_LIMIT и DRONE_CPU_QUOTA согласованы с ресурсами хоста?
- [ ] Настроена регулярная уборка образов и есть мониторинг свободного места под /var/lib/docker?
- [ ] Флаг trusted выдан только тем репозиториям, которым реально нужны host-volume и privileged?
- [ ] Кэш зависимостей включён, и время прохождения пайплайна измеряется, а не оценивается на глаз?

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).

- Разворачиваем Drone под ключ: сервер, раннеры, домен, TLS, OAuth к Gitea, GitHub или GitLab
- Переносим существующий CI на Drone и пишем шаблон .drone.yml под ваш стек и контуры
- Настраиваем секреты, права репозиториев и закрытую регистрацию под список сотрудников
- Ускоряем сборки: кэш зависимостей, prebuilt-образы, параллельные шаги, вторые раннеры
- Берём CI на сопровождение: обновление тегов, мониторинг диска и разбор упавших пайплайнов

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 Server: установка для Gitea и GitHub, OAuth-приложение и RPC-секрет (docs.drone.io — 2.x)

02 Server reference: DRONE_DATABASE_*, DRONE_CLEANUP_*, DRONE_USER_* (docs.drone.io — 2.x)

03 Server storage: database — postgres 9.6+, строка подключения (docs.drone.io — 2.x)

04 Docker Runner: installation и configuration reference (capacity, лимиты) (docs.drone.io — 1.8)

05 Docker Pipelines: steps, workspace /drone/src, volumes, conditions (docs.drone.io — 2.x)

06 Secrets: репозиторные секреты, from_secret и Allow Pull Requests (docs.drone.io — 2.x)

07 Плагин plugins/docker: registry, repo, tags, cache_from, dry_run (plugins.drone.io — 2026)

08 Наш шаблон docker-compose и .drone.yml для Drone (itfresh.ru — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.