Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

Свой GitLab для команды 3–15 разработчиков: CI/CD и бэкапы

GitLab CE 19.3 в docker compose на одном VPS: малый сайзинг, docker-раннер, план восстановления

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

Команде из 3–15 разработчиков нужен свой git-сервер с CI/CD, но бюджета на кластер нет — всё должно жить на одном VPS. Без выверенного сайзинга GitLab на 8 ГБ уходит в своп и отдаёт 502, а без схемы резервирования один сбой диска уничтожает репозитории, задачи и реестр образов. Мы разворачиваем GitLab CE в docker compose с ужатым профилем памяти и проверяемым восстановлением.

Почему это важно бизнесу

- Код и история задач — главный актив компании: потеря единственного сервера без бэкапа означает потерю продукта, а не «неудобство».
- GitLab CE бесплатен: вся плата — один VPS и VM раннера; дешевле облачных подписок уже при 5–7 разработчиках.
- Простой git-сервера останавливает всю разработку сразу: встают ревью, сборки и деплой на прод.
- Отрепетированное восстановление занимает 1–2 часа; без процедуры и секретов — дни, и часть данных не вернуть.

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

19.3

ветка GitLab CE: ставим последний патч 19.3.1, минорные обновления — ежемесячно

GitLab CE 19.3, docs.gitlab.com

4 vCPU/8 ГБ

сайзинг единственного VPS на 3–15 разработчиков (плюс 100 ГБ SSD) — с ужатым профилем памяти

наш стандарт

0

puma['worker_processes'] = 0 — одиночный режим Puma: минус 100–400 МБ RAM, ещё ~300 МБ даёт от…

по докам GitLab (memory-constrained)

10

sidekiq['concurrency'] вместо дефолтных 20 — фоновые очереди не отбирают CPU у web-части

по докам GitLab (memory-constrained)

604800 с

gitlab_rails['backup_keep_time'] — 7 суток локальных архивов, плюс off-site копия

gitlab.rb, наш стандарт

2 vCPU

отдельная VM под gitlab-runner (executor docker, concurrent = 2) — не кладёт сам GitLab

наш стандарт

РЕАЛИЗАЦИЯ 1 ИЗ 3

Ядро: GitLab CE в docker compose на одном VPS

Что настраиваем

Один VPS 4 vCPU / 8 ГБ / 100 ГБ SSD (Debian 12 + Docker): web, git по SSH, Container Registry

Как мы это делаем

1. Образ gitlab/gitlab-ce:19.3.1-ce.0, volumes config/logs/data на хостовые каталоги, shm_size 256m; SSH пробрасываем как 2222:22, чтобы не конфликтовать с sshd хоста
2. Вся конфигурация — через GITLAB_OMNIBUS_CONFIG в compose: external_url, letsencrypt['enable'] = true, SMTP клиента, registry_external_url
3. Профиль памяти по докам: puma['worker_processes'] = 0, sidekiq['concurrency'] = 10, prometheus_monitoring['enable'] = false плюс node/postgres/redis exporters — Git…
4. Первый вход — пароль из /etc/gitlab/initial_root_password (файл автоматически удаляется через 24 часа); сразу включаем 2FA и закрываем открытую регистрацию пользова…
5. Обновление = смена тега образа и docker compose up -d; маршрут между версиями сверяем инструментом Upgrade Path

РЕЗУЛЬТАТ

Команда получает git, merge requests, задачи и реестр образов на одном сервере за один рабочий день. Апгрейды сводятся к смене тега образа, конфигурация воспроизводима — compose-файл и gitlab.rb-фрагмент лежат в том же GitLab.

КЛЮЧЕВОЙ НЮАНС

Дефолтный сайзинг Omnibus малому серверу противопоказан: без одиночного режима Puma и отключения встроенного Prometheus 8 ГБ уходит в своп. Значения берём из раздела memory-constrained официальной документации.

РЕАЛИЗАЦИЯ 2 ИЗ 3

CI/CD: один docker-раннер без Kubernetes

Что настраиваем

Отдельная VM 2 vCPU / 4 ГБ (Debian 12): gitlab-runner той же ветки 19.3, executor docker

Как мы это делаем

1. Регистрация по authentication-токену glrt-… (старые регистрационные токены объявлены устаревшими): создаём раннер в UI, затем gitlab-runner register
2. config.toml: concurrent = 2, executor docker, privileged = false; кэш зависимостей — в docker volume с ключом по ветке
3. Образы собираем rootless-BuildKit (docker buildx) или Buildah вместо docker-in-docker — без privileged-режима на раннере
4. Пайплайн из 4 стадий: lint → test → build → deploy; выкладка на прод — только when: manual и только из protected-ветки main
5. Артефакты сборки уходят во встроенный Container Registry; в cron сервера — еженедельный gitlab-ctl registry-garbage-collect

РЕЗУЛЬТАТ

Полный цикл «коммит → тесты → образ → деплой» на собственных мощностях без Kubernetes: двух параллельных задач хватает команде до 15 человек, а при росте добавляется второй раннер с тегами, а не кластер.

КЛЮЧЕВОЙ НЮАНС

Раннер никогда не живёт на сервере GitLab: тяжёлый пайплайн съедает CPU и память web-части. Версию gitlab-runner держим в той же минорной ветке, что и сервер, — это требование совместимости из документации.

РЕАЛИЗАЦИЯ 3 ИЗ 3

Резервирование и восстановление одного сервера

Что настраиваем

Тот же VPS + off-site хранилище (S3-совместимое или второй сервер); цель — восстановление за 1–2 часа

Как мы это делаем

1. Ночной cron 01:30: gitlab-backup create CRON=1 — архив с репозиториями, БД и задачами; gitlab_rails['backup_keep_time'] = 604800 (7 суток)
2. Отдельным заданием — tar /etc/gitlab: gitlab-secrets.json и gitlab.rb; без секретов архив не расшифровать (CI-переменные, токены, 2FA)
3. Off-site: rclone или backup_upload_connection в S3-совместимое хранилище, шифрование на стороне клиента, retention 30 дней
4. Тестовое восстановление раз в квартал: чистая VM, тот же тег образа, gitlab-backup restore BACKUP=…, проверка клона репозитория и запуска пайплайна
5. Свежесть бэкапа под Zabbix: возраст последнего архива больше 26 часов — алерт в Telegram дежурному

РЕЗУЛЬТАТ

Сбой диска или VPS перестаёт быть катастрофой: сервер поднимается на любой площадке из compose-файла, архива и секретов за 1–2 часа, и эта цифра подтверждена квартальными учениями, а не предположением.

КЛЮЧЕВОЙ НЮАНС

gitlab-backup restore работает только на точно той же версии GitLab, из которой сделан архив, — рядом с бэкапом храним тег образа и никогда не обновляем сервер без свежей копии.

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- ✕ **Раннер на сервере GitLab**
  Тяжёлая сборка съедает CPU/RAM, Puma отдаёт 502. Раннер выносим на отдельную VM 2 vCPU / 4 ГБ и ограничиваем concurrent.
- ✕ **Потерян gitlab-secrets.json**
  CI-переменные, токены и 2FA зашифрованы этим файлом: бэкап без него неполноценен. Копируем /etc/gitlab отдельным заданием off-site.
- ✕ **Restore на другой версии**
  gitlab-backup restore требует точного совпадения версии сервера и архива. Тег образа фиксируем в имени/метаданных бэкапа.
- ✕ **Прыжок через версии при апгрейде**
  Между ветками есть обязательные остановки и background migrations. Маршрут строим инструментом Upgrade Path и ждём завершения миграций.
- ✕ **Registry без garbage-collect**
  Docker-образы копятся бесконечно и за полгода забивают диск. Ставим cleanup policies проектов + gitlab-ctl registry-garbage-collect в cron.
- ✕ **Дефолтный профиль памяти**
  Штатные воркеры Puma, Sidekiq и встроенный Prometheus не влезают в 8 ГБ. Применяем memory-constrained настройки из документации.
- ✕ **Порт 22 в docker-варианте**
  Контейнер конфликтует с sshd хоста. Мапим 2222:22 и прописываем gitlab_rails['gitlab_shell_ssh_port'] = 2222 для корректных ссылок клонирования.
- ✕ **Бэкап на тот же диск**
  Локальный архив гибнет вместе с VPS. Обязателен off-site: S3-совместимое хранилище или второй сервер, плюс контроль свежести копии.

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- GitLab CE 19.3 в docker compose на VPS 4 vCPU / 8 ГБ, память ужата по докам
- Один docker-раннер на отдельной VM 2 vCPU / 4 ГБ, concurrent = 2
- Ночной gitlab-backup + копия gitlab-secrets.json вне сервера

НОРМАЛЬНО

- Off-site бэкапы в S3-совместимое хранилище, retention 30 дней
- Container Registry с cleanup policies и еженедельным garbage-collect
- Zabbix-мониторинг: RAM, диск, возраст бэкапа, статус пайплайнов
- Ежемесячные патчи по Upgrade Path с бэкапом перед каждым

ХОРОШО

- Квартальный тестовый restore на чистой VM с прогоном пайплайна
- Второй раннер с тегами под тяжёлые/приватные сборки
- Protected branches, ручной деплой на прод и 2FA для всех
- Снапшоты гипервизора + документированный DR-план с ролями

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Известна ли точная версия вашего GitLab и когда последний раз ставились патчи безопасности?
- [ ] Переживёт ли компания потерю диска git-сервера — существует ли бэкап вне этого сервера?
- [ ] Сохранён ли gitlab-secrets.json отдельно от самого сервера?
- [ ] Проводилось ли хотя бы одно тестовое восстановление из бэкапа на чистой машине?
- [ ] Раннер CI работает на отдельной VM, а не на сервере GitLab?
- [ ] Ограничен ли рост Container Registry — есть ли garbage-collect по расписанию?
- [ ] Включена ли 2FA и закрыта ли самостоятельная регистрация пользователей?
- [ ] Придёт ли алерт, если ночной бэкап не создался или устарел?
- [ ] Деплой на прод запускается только вручную и только из защищённой ветки?

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 GitLab CE в docker compose под ключ: сайзинг, TLS, SMTP, 2FA — за 1–2 дня
- Настраиваем CI/CD: docker-раннер, реестр образов, пайплайн lint/test/build/deploy под ваш стек
- Строим резервирование: ночные бэкапы, off-site копии, мониторинг свежести, тестовый restore
- Переносим репозитории и задачи с GitHub/Bitbucket, переписываем workflow под GitLab CI
- Сопровождаем: ежемесячные обновления по Upgrade Path и дежурный мониторинг

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 Install GitLab in a Docker container (Docker Compose) (docs.gitlab.com — 19.3)

02 Running GitLab in a memory-constrained environment (docs.gitlab.com — 19.3)

03 Back up and restore GitLab (Rake tasks) (docs.gitlab.com — 19.3)

04 GitLab Runner: Docker executor и config.toml (docs.gitlab.com/runner — 19.3)

05 Container registry administration: garbage collection (docs.gitlab.com — 19.3)

06 Upgrade paths / GitLab Upgrade Path tool (docs.gitlab.com — 2026)

07 Шаблон compose и чек-лист восстановления ITfresh (itfresh.ru — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.