Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

DevOps-культура под ключ: от ручного деплоя к конвейеру CI/CD

Разворачиваем GitLab CE, Terraform, Prometheus/Grafana и метрики DORA в компаниях до 50 PM

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

Релизы у заказчика держатся на одном человеке и 40-шаговой инструкции: деплой раз в месяц, откат — вручную по памяти, об инцидентах узнают от пользователей. Мы решаем это внедрением DevOps-контура: GitLab CI/CD, инфраструктура как код, мониторинг с алертами. Без него каждый релиз — лотерея: простой учётных систем и сайта, потерянные заказы, невозможность быстро выпускать дораб...

Почему это важно бизнесу

- Релиз раз в месяц вместо ежедневного — доработки под клиентов ждут в очереди неделями, бизнес теряет сделки
- Bus-factor = 1: отпуск или уход единственного «релизного» инженера полностью останавливает выпуск обновлений
- Ручной деплой без отката — каждая ошибка превращается в часы простоя учётных систем и веб-сервисов
- Об авариях первыми узнают клиенты, а не мониторинг: репутационные потери и разбор «на глазах» у заказчика

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

19.1

GitLab CE self-hosted: держим текущий стабильный минор, обновление строго по upgrade path

по докам GitLab

1.15.8

Terraform: пиним версию в required_version, state — в S3-совместимом backend с блокировкой

по докам HashiCorp Terraform

3.13 LTS

Prometheus LTS-ветка: scrape_interval 15 с, retention 30 дней — достаточно малому офису

по докам prometheus.io

13.x

Grafana: дашборды и алерты заводим как код через provisioning (YAML в git), не «мышкой»

по докам grafana.com

80%

Порог покрытия тестами в MR-гейте: пайплайн блокирует merge при падении coverage ниже

наш стандарт

30 мин

Максимум на полный пайплайн build→test→deploy; дольше — разработчики копят батчи правок

наш стандарт

РЕАЛИЗАЦИЯ 1 ИЗ 3

Конвейер CI/CD на self-hosted GitLab CE

Что настраиваем

GitLab Omnibus на отдельной VM (4 vCPU/8 ГБ) + gitlab-runner с docker executor на билд-узле; охват — все репозитории клиента

Как мы это делаем

1. Ставим GitLab CE (Omnibus), включаем Container Registry, переносим код из архивов и файловых шар в git; main — protected branch
2. Разворачиваем gitlab-runner (docker executor), пишем .gitlab-ci.yml: stages build→test→deploy, образы тегируем $CI_COMMIT_SHORT_SHA
3. Деплой на staging — автоматом из main; на production — тот же job, но с when: manual и environment: production
4. Секреты — только в masked/protected CI/CD variables, в репозитории ни одного пароля; запуск prod-job — только с protected branches

РЕЗУЛЬТАТ

Релиз перестаёт быть событием: любая правка проходит одинаковый путь от коммита до production за 20–30 минут, откат — повторный деплой предыдущего тега из Registry. «Релизный инженер» больше не единая точка отказа — деплоит любой участник команды кнопкой.

КЛЮЧЕВОЙ НЮАНС

Пилотный репозиторий доводим до образцового, остальные подключаем include-шаблоном пайплайна. При обновлении миноров GitLab сверяем deprecated-ключи YAML по release notes — синтаксис CI меняется между версиями.

РЕАЛИЗАЦИЯ 2 ИЗ 3

Инфраструктура как код: Terraform с plan в merge request

Что настраиваем

Виртуалки, DNS и сетевые правила клиента в облаке; state — в S3-совместимом bucket с нативной блокировкой

Как мы это делаем

1. Пиним required_version = "~> 1.15", backend "s3" с use_lockfile = true; версии провайдеров фиксирует .terraform.lock.hcl в git
2. Существующие «серверы-снежинки» заводим через terraform import по одному в неделю, добиваясь нулевого diff в plan
3. В CI: terraform fmt -check и validate на каждый push, plan -out=tfplan артефактом в MR; apply tfplan — только manual из main
4. Ручные правки в облачной консоли запрещаем; дрейф ловим ночным запуском terraform plan -detailed-exitcode (код 2 = алерт в Telegram)

РЕЗУЛЬТАТ

Вся инфраструктура читается как код и восстанавливается с нуля за часы, а не недели. Каждое изменение видно в MR до применения: ревьюер видит точный diff ресурсов, а случайное удаление VM пайплайн не пропустит без ручного подтверждения.

КЛЮЧЕВОЙ НЮАНС

State хранит секреты открытым текстом — он никогда не попадает в git, только шифрованный bucket с узким доступом. Версию Terraform пиним жёстко: непроверенный апгрейд минора может изменить план.

РЕАЛИЗАЦИЯ 3 ИЗ 3

Мониторинг, алерты и метрики DORA

Что настраиваем

Prometheus + Alertmanager + Grafana на VM мониторинга; экспортеры на всех серверах клиента, метрики деплоев — из GitLab API

Как мы это делаем

1. Prometheus 3.13 LTS: node_exporter и blackbox_exporter на все узлы, scrape_interval 15 с, retention 30 дней
2. Alertmanager: маршрутизация в Telegram-канал дежурного, group_wait 30 с, repeat_interval 4 ч, severity page/warning
3. Grafana 13.x разворачиваем provisioning-ом из git: дашборды сервисов + DORA-борд (частота деплоев и lead time из GitLab API)
4. Каждый инцидент SEV-1/SEV-2 закрываем blameless-постмортемом по нашему шаблону: хронология, корневая причина, action items с владельцами

РЕЗУЛЬТАТ

Об инциденте первым узнаёт дежурный, а не клиент: алерт прилетает раньше жалобы. DORA-борд показывает руководителю прогресс в цифрах — частоту деплоев, lead time и время восстановления, и разговор о пользе DevOps перестаёт быть «на ощущениях».

КЛЮЧЕВОЙ НЮАНС

Алертов должно быть мало и все — действенные: у каждого runbook и владелец, иначе канал замьютят за месяц. Стартовые пороги — error rate 5% и p99 latency, остальное добавляем по итогам постмортемов.

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- ✕ **Big bang на всю компанию**
  Перевод всех команд разом тонет в сопротивлении. Стартуем с одной пилотной командой, показываем результат и тиражируем готовый шаблон.

- ✕ **Деплой умеет один человек**
  Bus-factor 1: отпуск = стоп релизов. Всё, что делается руками по инструкции, переносим в job пайплайна с manual-гейтом — деплоит любой.

- ✕ **Секреты в репозитории**
  Пароли в .gitlab-ci.yml и tfvars утекают с первым клоном. Только masked/protected CI/CD variables и внешнее хранилище секретов.

- ✕ **terraform apply мимо CI**
  Локальный apply создаёт дрейф и конфликты state. Креды облака выдаём только runner-у, инженеры вносят изменения через MR с plan.

- ✕ **Алерты на всё подряд**
  Через месяц канал мониторинга замьючен. Каждому алерту — runbook и владелец; пороги, которые не приводят к действию, убираем.

- ✕ **Покрытие тестами ради цифры**
  80% на геттерах бесполезны. Сначала контрактные и smoke-тесты критичных путей, порог в CI поднимаем постепенно от факта.

- ✕ **Пайплайн на 40+ минут**
  Разработчики перестают ждать и копят батчи правок. Кэш зависимостей, слои Docker и параллельные jobs держат цикл в 30 минут.

- ✕ **Постмортем как поиск виноватого**
  Одна «казнь» — и инциденты начинают скрывать. Фасилитатор пресекает обвинения: разбираем системные причины, не людей.

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- Весь код и конфиги в git (GitLab CE), main — protected branch
- Один пайплайн build→test→deploy хотя бы для главного сервиса
- Базовый мониторинг: node_exporter + алерт о недоступности в Telegram

НОРМАЛЬНО

- CI/CD на все репозитории, деплой в prod — кнопкой с manual-гейтом
- Terraform для новой инфраструктуры, plan в каждом merge request
- Prometheus 3.x LTS + Grafana, у каждого алерта runbook и владелец
- Blameless-постмортем на каждый инцидент SEV-1/SEV-2

ХОРОШО

- 100% инфраструктуры в коде, дрейф-контроль ночным terraform plan
- Контрактные и smoke-тесты как гейт merge, порог покрытия в CI
- DORA-борд: частота деплоев, lead time, CFR, MTTR — автоматически
- Шаблоны пайплайнов через include: новый сервис подключается за час

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Может ли любой инженер команды задеплоить в production кнопкой, без «того самого» человека?
- [ ] Откатите ли вы неудачный релиз за 15 минут повторным деплоем предыдущего тега из Registry?
- [ ] Есть ли хоть один пароль в репозитории или .gitlab-ci.yml? (правильный ответ — ни одного)
- [ ] Проходит ли каждое изменение инфраструктуры через terraform plan в merge request?
- [ ] Хранится ли state Terraform вне git — в бэкенде с блокировкой и ограниченным доступом?
- [ ] Узнаёте ли вы об инциденте от мониторинга раньше, чем от пользователей?
- [ ] У каждого алерта есть runbook и владелец, а несрабатывающие алерты регулярно вычищаются?
- [ ] Заканчивается ли разбор инцидента списком action items с владельцами, а не поиском виноватого?
- [ ] Обновляется ли GitLab по официальному upgrade path, а не «через три мажора разом»?
- [ ] Видит ли руководитель DORA-метрики (частоту деплоев, MTTR) на дашборде, а не в отчётах «на словах»?

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 self-hosted GitLab CE + runner и строим первый пайплайн под ваш стек за 2–3 недели
- Переводим существующую инфраструктуру в Terraform поэтапным import без остановки сервисов
- Ставим Prometheus/Alertmanager/Grafana с алертами в Telegram и DORA-дашбордом для руководителя
- Обучаем команду: воркшопы по CI/CD, shared on-call и фасилитация первых blameless-постмортемов
- Сопровождаем контур по абонентке: обновления GitLab/Terraform, ревизия алертов, постмортемы

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

Техническая база

01 GitLab CI/CD YAML syntax reference (.gitlab-ci.yml) (docs.gitlab.com — 19.1)

02 GitLab Upgrade Path и maintenance policy (docs.gitlab.com — 2026)

03 Terraform: Backend Configuration и State Locking (backend "s3") (developer.hashicorp.com — 1.15)

04 Prometheus: Alerting Rules и конфигурация scrape (prometheus.io — 3.13 LTS)

05 Grafana: Provisioning dashboards and data sources (grafana.com — 13.x)

06 GitLab: DORA metrics / Value Stream Analytics (docs.gitlab.com — 2026)

07 Наш шаблон blameless postmortem и runbook-каталога алертов (itfresh.ru — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.