Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

Docker Compose в продакшне: боевой стандарт compose.yaml

Healthcheck, лимиты, секреты, изоляция сетей, деплой без простоя — как мы это делаем

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

К нам приходят с compose-файлом, написанным для теста: latest-теги, ни одного healthcheck, ни лимитов памяти, логи без ротации, пароли прямо в YAML. Такой стек живёт до первого обновления: база не успевает подняться, приложение уходит в цикл рестартов, логи забивают раздел, заказы перестают приниматься. Мы приводим compose к боевому стандарту эксплуатации.

Почему это важно бизнесу

- Простой интернет-магазина или учётной базы в рабочий день — прямые потери выручки и вал обращений в поддержку.
- Падение из-за забитого логами диска чинится часами: нужен доступ к хосту, а не перезапуск сервиса.
- Пароли в git-репозитории compose утекают вместе с исходниками при любой смене подрядчика.
- Без лимитов один утекающий контейнер кладёт весь хост, включая сервисы соседних подразделений.
- Обновление без проверки готовности превращает релиз в лотерею и заставляет откатываться вручную.

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

2.40.3

Версия Compose CLI, на которой стандартизируем хосты: include, !reset, !override

спецификация Compose

30 с

start_period healthcheck БД в нашем стандарте; в спецификации по умолчанию 0 с

наш стандарт

20m / 5

max-size и max-file драйвера local — дефолт Docker; дублируем в daemon.json

докум. Docker Engine 29

x1.5

Запас deploy.resources.limits.memory к наблюдаемому пику потребления сервиса

наш стандарт

10 с

stop_grace_period по умолчанию; для БД и очередей поднимаем до 60 с

спецификация Compose

/run/secrets

Каталог, куда монтируются секреты: приложение читает *_FILE, а не environment

спецификация Compose

Каркас файлов и фиксация версий

Что настраиваем

Базовый compose.yaml + compose.prod.yaml на одном Linux-хосте заказчика, запуск под non-root пользователем deploy

Как мы это делаем

1. Разносим конфиг: compose.yaml (общее), compose.override.yaml (dev), compose.prod.yaml (лимиты, логи, restart); запуск -f compose.yaml -f compose.prod.yaml
2. Пиним образы тегом и digest: postgres:18.6-alpine@sha256:...; latest под запретом, версия меняется только через ${TAG} в .env
3. Повторяющиеся блоки выносим в YAML-якоря (x-logging, x-healthcheck) и extends, общие куски подключаем через include — не дублируем 200 строк
4. Валидация в CI: docker compose config --quiet на каждый коммит ловит опечатки в ключах и неразрешённые переменные до деплоя

РЕЗУЛЬТАТ

Один источник истины на среду. Прод-запуск воспроизводится с нуля за минуты, а dev-настройки (bind-mounts, порты наружу) физически не попадают на боевой хост — они лежат в override-файле, который в проде не подключается.

КЛЮЧЕВОЙ НЮАНС

compose.override.yaml подхватывается автоматически, но при явном -f игнорируется. Мы всегда указываем полный список файлов в systemd-юните, а не полагаемся на порядок по умолчанию.

РЕАЛИЗАЦИЯ 2 ИЗ 3

Управляемый старт: healthcheck, зависимости и лимиты

Что настраиваем

Слой БД + приложение + reverse proxy: postgres, API, nginx на одном хосте 4 vCPU / 16 ГБ

Как мы это делаем

1. На БД вешаем healthcheck: test CMD-SHELL pg_isready -U app -d app, interval 10s, timeout 5s, retries 5, start_period 30s, start_interval 2s
2. start_interval действует только внутри start_period и требует Docker Engine 25 и новее — на старых хостах ключ молча не даёт эффекта
3. Зависимости: depends_on с condition: service_healthy, миграции — отдельный сервис с condition: service_completed_successfully
4. Ресурсы через deploy.resources: limits.memory 1g, limits.cpus '1.5', limits.pids 200, reservations.memory 512m — 1.5x от наблюдаемого пика
5. restart: unless-stopped на долгоживущих, on-failure:3 на джобах, no на init-контейнерах; stop_grace_period 60s для Postgres и очередей

РЕЗУЛЬТАТ

Стек поднимается детерминированно: приложение не ломится в неинициализированную базу, миграции отрабатывают до старта воркеров. Утечка памяти гасится внутри лимита контейнера и не роняет ни соседей, ни хост целиком.

КЛЮЧЕВОЙ НЮАНС

start_period не безграничен: если сервис стартует дольше, контейнер уйдёт в unhealthy и depends_on заблокирует запуск. Замеряем реальное время старта на холодном томе и берём запас.

РЕАЛИЗАЦИЯ 3 ИЗ 3

Секреты, изоляция сетей и деплой без простоя

Что настраиваем

Периметр стека: сети frontend/backend, секреты БД, обновление образа приложения на боевом хосте

Как мы это делаем

1. Две сети: frontend для nginx, backend с internal: true — БД и API без выхода наружу; ports публикуем только у прокси, где можно — с привязкой к 127.0.0.1
2. Пароли — через top-level secrets с source: file; Compose монтирует их в /run/secrets/<name>, приложение читает POSTGRES_PASSWORD_FILE
3. Для file-секретов Compose игнорирует mode/uid/gid (под капотом bind-mount) — права выставляем на сам файл-источник: владелец deploy, chmod 400
4. .env — только для несекретных параметров, chmod 600 и в .gitignore; сами секреты выкладывает скрипт-обёртка из внешнего хранилища перед стартом
5. Деплой: docker compose pull && docker compose up -d --wait --wait-timeout 180 --remove-orphans — при недостижении healthy код выхода ненулевой

РЕЗУЛЬТАТ

Скомпрометированный веб-контейнер не видит БД напрямую, пароли не светятся в docker inspect и в истории git. Деплой либо доводится до healthy-состояния, либо честно падает в CI — без «зелёного» релиза с мёртвым сервисом.

КЛЮЧЕВОЙ НЮАНС

--wait ждёт состояния running или healthy. У сервиса без healthcheck готовностью считается сам факт запуска процесса, поэтому гарантию деплоя даёт только healthcheck.

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- ✕ **latest вместо фиксированного тега**
  Пересоздание контейнера молча подтягивает новый мажор БД. Пиним тег и digest, версию меняем осознанно через ${TAG} в .env.

- ✕ **restart: always на всём подряд**
  Битый контейнер уходит в бесконечный цикл рестартов и нагружает CPU. Долгоживущим — unless-stopped, джобам — on-failure:3, init — no.

- ✕ **healthcheck проверяет только порт**
  TCP-порт открыт, а база ещё накатывает WAL. Проверяем прикладную готовность: pg_isready, HTTP /healthz с запросом к зависимостям.

- ✕ **depends_on без condition**
  Короткая форма ждёт только запуска процесса, а не готовности. Всегда пишем длинную форму с condition: service_healthy.

- ✕ **Логи без ротации**
  У json-file max-size по умолчанию -1, то есть без предела: за пару недель раздел забит. Ставим local с max-size/max-file в daemon.json и в сервисе.

- ✕ **Секреты в environment**
  Переменные видны в docker inspect, в списке процессов и в дампах краша. Используем secrets и *_FILE-переменные приложений.

- ✕ **Все сервисы в одной сети**
  Любой контейнер достучится до БД по имени. Разносим на frontend и backend с internal: true, наружу торчит только прокси.

- ✕ **Бэкап копированием каталога тома**
  Файлы БД копируются в неконсистентном состоянии и не восстанавливаются. Делаем логический дамп внутри контейнера и проверяем restore.

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- Фиксированные теги образов, restart: unless-stopped на долгоживущих сервисах
- Ротация логов: драйвер local, max-size 10m, max-file 5 в daemon.json
- limits.memory и limits.cpus у каждого сервиса, .env с chmod 600 вне git
- Ежедневный логический дамп БД по cron на отдельный носитель

НОРМАЛЬНО

- healthcheck на каждом сервисе-зависимости + depends_on: service_healthy
- Разделение сетей frontend / backend с internal: true
- Деплой через docker compose up -d --wait с ненулевым кодом при сбое
- Три файла: compose.yaml, compose.override.yaml, compose.prod.yaml

ХОРОШО

- Пин образов по digest и docker compose config --quiet в CI на каждый коммит
- Секреты из внешнего хранилища в /run/secrets, приложение читает *_FILE
- Централизованный сбор логов и метрик, алерты по unhealthy-контейнерам
- Ежеквартальный тест восстановления БД на отдельном стенде

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Все образы зафиксированы тегом (а лучше digest), latest не используется нигде?
- [ ] У каждого сервиса, от которого кто-то зависит, есть healthcheck со start_period?
- [ ] depends_on везде в длинной форме с condition, а не простым списком имён?
- [ ] Для каждого контейнера заданы limits.memory, limits.cpus и limits.pids?
- [ ] Ротация логов включена и в daemon.json, и в compose-файле сервисов?
- [ ] Пароли лежат вне YAML и вне git, а у файла-источника секрета права 400?
- [ ] БД и внутренние API изолированы в сети с internal: true, наружу открыт только прокси?
- [ ] Деплой выполняется с --wait и падает, если сервис не перешёл в healthy?
- [ ] Дампы БД снимаются логически, хранятся вне хоста и проверялись восстановлением за квартал?
- [ ] В своих Dockerfile заданы USER не root и инструкция HEALTHCHECK?

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 текущего compose-файла: теги, healthcheck, лимиты, сети, секреты — с планом правок
- Переписываем стек под прод: базовый файл + prod-оверрайд, systemd-юнит автозапуска
- Настраиваем ротацию и централизованный сбор логов, алерты по unhealthy-контейнерам
- Ставим бэкапы: логические дампы по расписанию, выгрузка вне хоста, тест восстановления
- Собираем деплой-конвейер: pull, валидация config, up --wait, откат на предыдущий тег

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 Compose file reference: services — healthcheck, deploy, restart (docs.docker.com — 2026)

02 Compose file reference: secrets + руководство Use secrets (docs.docker.com — 2026)

03 Compose: Merge, include, fragments, теги !reset и !override (docs.docker.com — 2026)

04 CLI reference: docker compose up (--wait, --wait-timeout, --pull) (docs.docker.com — v2.40)

05 Logging drivers: local и json-file, настройка daemon.json (docs.docker.com — 29.x)

06 Docker Compose release notes, ветка v2.40 (docs.docker.com — 2026)

07 Наш шаблон compose.prod.yaml и деплой-скрипт ITfresh (itfresh.ru — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.