Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

Docker в продакшене: 15 инженерных практик

Как мы приводим контейнерную инфраструктуру к безопасному и предсказуемому виду

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

К нам приходят с типовой картиной: 20+ микросервисов, каждый образ по 1.5-2 ГБ, всё запущено от root, сканирования уязвимостей нет, лимитов CPU и памяти нет, HEALTHCHECK отсутствует, а в прод катится тег :latest. Итог — медленные релизы, утечка памяти кладёт соседние контейнеры, а любая CVE в приложении сразу даёт root внутри контейнера. Мы за полгода приводим такой парк к предсказуемому и безопасному состоянию.

Почему это важно бизнесу

- Контейнер от root плюс непропатченная CVE — это потенциальный побег на хост и компрометация всей ноды, а не одного сервиса.
- Утёкший по памяти сервис без лимитов роняет соседние контейнеры на той же ноде — простой сразу нескольких систем.
- Раздутые образы под 1.8 ГБ и билды по 12-18 минут тормозят каждый релиз и раздувают трафик реестра и диск CI.
- Тег :latest в проде = невоспроизводимый деплой: неясно, что именно работает, и корректный откат невозможен.
- Без HEALTHCHECK оркестратор не видит зависший контейнер и продолжает держать трафик на фактически мёртвом сервисе.

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

29.x

Docker Engine, на который мы стандартизируем прод; tini встроен и включается флагом --init

по докам Docker Engine v29

1.8→0.12 ГБ

типовое сокращение размера образа после перевода на multi-stage и alpine/distroless-базу

наш стандарт

30s/30s/3

дефолт HEALTHCHECK: interval 30s, timeout 30s, 3 повтора; start-period задаём под время старта

по докам Dockerfile HEALTHCHECK

HIGH,CRITICAL

порог падения CI в Trivy: --exit-code 1 --severity HIGH,CRITICAL --ignore-unfixed

по докам Trivy v0.72

порт ≥1024

непривилегированный запуск: USER appuser, слушаем порт выше 1024, наружу пробрасываем -p

наш стандарт

+20-30%

запас лимита памяти над пиком, снятым за неделю через cAdvisor + Prometheus

наш стандарт

РЕАЛИЗАЦИЯ 1 ИЗ 3

Слой образа: multi-stage, кеш и непривилегированный запуск

Что настраиваем

Все образы сервисов: Dockerfile, база alpine 3.24 / distroless, финальный слой

Как мы это делаем

1. Разбиваем Dockerfile на стадии builder→final: компилятор и dev-зависимости остаются в builder, в финал через COPY --from копируем только бинарь или собранный код.
2. Порядок слоёв под кеш: сначала go.mod/go.sum + go mod download (или package*.json + npm ci), затем COPY . . — правка кода не инвалидирует слой зависимостей.
3. Добавляем adduser -S appuser и USER appuser; COPY --chown=appuser переносит артефакт без root; порт слушаем ≥1024.
4. Кладём .dockerignore (.git, node_modules, tests, .env*) — контекст сборки падает с сотен МБ до единиц.
5. Пиним базовый образ по дайджесту (alpine@sha256:...), а не по плавающему тегу.

РЕЗУЛЬТАТ

Образ уменьшается на порядок, билд при правке только кода занимает секунды вместо минут за счёт кеша слоёв, а поверхность атаки сведена к минимальной базе без root.

КЛЮЧЕВОЙ НЮАНС

COPY --from и порядок слоёв — ключ; в доке Dockerfile держим в голове, что каждая RUN/COPY даёт отдельный слой, а секреты в ARG/COPY навсегда оседают в history.

Рантайм: PID 1, healthcheck, лимиты и логи

Что настраиваем

Слой запуска сервиса: ENTRYPOINT, HEALTHCHECK, лимиты ресурсов, драйвер логов

Как мы это делаем

1. Ставим tini как init (ENTRYPOINT ["/sbin/tini","--"]) или запускаем docker run --init — tini reap-ит зомби и форвардит SIGTERM, docker stop не упирается в SIGKILL.
2. В entrypoint-скриптах последней строкой exec /app/server — процесс становится PID 1 и корректно принимает сигналы.
3. HEALTHCHECK --interval=30s --timeout=5s --start-period=10s --retries=3 на общий эндпоинт /health с проверкой БД и Redis.
4. Лимиты памяти/CPU: mem_limit и cpus (Compose v2) или resources.limits; запас 20-30% над пиком по cAdvisor + Prometheus.
5. Логи только в stdout/stderr, драйвер json-file с max-size/max-file или пересылка в Fluentd/Vector на уровне демона.

РЕЗУЛЬТАТ

Оркестратор видит реальное состояние и перезапускает зависшие контейнеры; один сервис не съедает память ноды; логи собираются централизованно без захода внутрь контейнера.

КЛЮЧЕВОЙ НЮАНС

start-period важнее retries: пока сервис поднимается, провалы не считаются; в доке сверяем, что start-interval требует Engine 25.0+, а deploy.resources работает только в Swarm.

РЕАЛИЗАЦИЯ 3 ИЗ 3

Пайплайн: секреты сборки, сканирование, теги и очистка

Что настраиваем

CI/CD и реестр: BuildKit, Trivy-гейт, стратегия тегов, GC реестра

Как мы это делаем

1. Секреты сборки — только через BuildKit: # syntax=docker/dockerfile:1 и RUN --mount=type=secret; docker build --secret не оставляет ключ в слоях и docker history.
2. Trivy в CI: trivy image --exit-code 1 --severity HIGH,CRITICAL --ignore-unfixed; билд падает на уязвимостях, базовые образы пересобираем еженедельно.
3. Trivy пиним по дайджесту aquasec/trivy@sha256:..., а не по плавающему тегу aquasec/trivy:latest — версия сканера в CI фиксирована и воспроизводима.
4. Теги: в прод только неизменяемый SHA или semver; :latest собираем для удобства, но никогда не деплоим.
5. Очистка: ночной docker system prune -af --filter until=168h + registry garbage-collect; lifecycle-политика чистит untagged старше 30 дней.

РЕЗУЛЬТАТ

Ключи не утекают через слои, уязвимые образы не доезжают до прода, каждый деплой воспроизводим и откатываем по конкретному дайджесту, а диски CI и реестра не забиваются мусором.

КЛЮЧЕВОЙ НЮАНС

Неизменяемость — основа: деплоим по дайджесту, а не по тегу; в доке Trivy и BuildKit сверяем, что secret монтируется только на время RUN и не попадает в кеш экспорта.

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- **deploy.resources в обычном docker compose молча игнорируется**
  Секция deploy.resources.limits работает только в Swarm. В legacy docker-compose v1 при up она игнорировалась. Берём Compose v2 (плагин docker compose) либо mem_limit/cpus на уровне сервиса.

- **Приложение остаётся PID 1 и глотает SIGTERM**
  Без tini/--init или exec процесс как PID 1 не получает дефолтных обработчиков сигналов: docker stop ждёт таймаут и добивает SIGKILL. Ставим tini или exec последней строкой entrypoint.

- **Секрет прокинут через ARG/COPY и лежит в history**
  Значения ARG и файлы, добавленные COPY, навсегда остаются в слоях и достаются через docker history и докачку слоёв. Только RUN --mount=type=secret через BuildKit — секрет живёт лишь во время RUN.

- **HEALTHCHECK без start-period убивает медленный старт**
  При start-period=0 проверки идут сразу; тяжёлый старт (миграции, прогрев кеша) даёт 3 провала и рестарт-петлю. Задаём --start-period под реальное время выхода сервиса на /health.

- **Пин образа по тегу вместо дайджеста**
  Тег :latest и даже semver перезаписываемы: под тем же именем в реестр может лечь другой дайджест, деплой становится невоспроизводимым. Образы и инструменты пиним по image@sha256, прод — по неизменяемому SHA-тегу.

- **USER задан, но процесс всё равно от root**
  Инструкция USER без COPY --chown оставляет артефакты во владении root, а порт <1024 требует привилегий. Проверяем chown файлов, слушаем порт ≥1024, наружу пробрасываем -p 80:8080.

- **Логи пишутся в файл внутри контейнера**
  docker logs показывает пустоту, лог не ротируется и раздувает слой или том. Все сервисы переводим на stdout/stderr, а ротацию и доставку настраиваем драйвером демона (json-file, Fluentd/Vector).

- **json-file без max-size забивает диск ноды**
  Дефолтный драйвер json-file не ротирует логи — за недели он съедает диск хоста. В daemon.json или на сервисе задаём log-opts max-size=10m и max-file=3.

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- Убрать root: adduser -S + USER appuser, COPY --chown, слушать порт ≥1024.
- HEALTHCHECK на /health для каждого сервиса: interval 30s, timeout 5s, retries 3.
- Trivy-гейт в CI на HIGH/CRITICAL и tini/--init как init-процесс контейнера.

НОРМАЛЬНО

- Multi-stage + .dockerignore + кеш слоёв (зависимости отдельным слоем).
- Лимиты mem_limit/cpus с запасом 20-30% над пиком по cAdvisor + Prometheus.
- Секреты сборки через BuildKit RUN --mount=type=secret, деплой по SHA-тегу.
- Логи в stdout/stderr плюс ротация json-file max-size/max-file.

ХОРОШО

- Managed Kubernetes (например Yandex, 1.34) с rolling updates, автоскейлингом и probes.
- Пин всех образов и Trivy по дайджесту image@sha256, еженедельная пересборка баз.
- Автоочистка: prune until=168h + registry GC + lifecycle untagged>30д, semver 90д.
- Отладка отдельным netshoot-контейнером и kubectl debug, без инструментов в проде.

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Все контейнеры запущены от непривилегированного пользователя (USER, не root)?
- [ ] У каждого сервиса есть HEALTHCHECK с корректным start-period под время старта?
- [ ] PID 1 — это tini/--init или exec, и docker stop не упирается в SIGKILL?
- [ ] Заданы лимиты памяти и CPU, и они реально работают (Compose v2 / resources), а не игнорируются?
- [ ] Trivy в CI роняет билд на HIGH/CRITICAL, базовые образы пересобираются регулярно?
- [ ] Секреты сборки идут через BuildKit --mount=type=secret, а не через ARG/COPY?
- [ ] В прод деплоится неизменяемый SHA или дайджест, а не тег :latest?
- [ ] Логи уходят в stdout/stderr с ротацией, а не в файл внутри контейнера?
- [ ] Есть .dockerignore и multi-stage, размер образа и контекста сборки минимальны?
- [ ] Настроена автоочистка образов/слоёв и garbage collection реестра?

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 Dockerfile и compose-файлов: root, размеры образов, отсутствие лимитов и healthcheck — с планом исправления.
- Внедряем multi-stage, непривилегированный запуск, HEALTHCHECK и лимиты ресурсов по всему парку сервисов.
- Встраиваем Trivy-гейт и BuildKit-секреты в CI/CD, наводим стратегию тегов по SHA/semver.
- Миграция с Docker Compose на managed Kubernetes (Yandex Cloud) с rolling updates и автоскейлингом.
- Настраиваем централизованные логи (Fluentd/Vector), метрики cAdvisor + Prometheus и автоочистку реестра.

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 Dockerfile reference: HEALTHCHECK, USER, ENTRYPOINT, COPY --chown (docs.docker.com — Engine 29)

02 Build secrets: RUN --mount=type=secret (BuildKit frontend) (docs.docker.com — dockerfile:1)

03 Compose file reference: resources, deploy, logging, mem_limit (docs.docker.com — Compose v2)

04 Runtime options: memory/CPU limits, флаг --init (docs.docker.com — Engine 29)

05 Trivy: scanning, --severity, --exit-code, --ignore-unfixed (trivy.dev — v0.72)

06 tini — init для контейнеров: reaping зомби и форвардинг сигналов (github.com/krallin/tini — master)

07 Yandex Managed Service for Kubernetes: каналы обновлений (yandex.cloud — 1.34)

08 Наш стандарт Dockerfile и CI для контейнеризации ITfresh (itfresh.ru — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.