Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

Self-hosted runners GitHub Actions: CI/CD на своём железе

Как мы строим парк раннеров: агенты под systemd, метки, ARC-автоскейлинг и изоляция контура

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

Команда разработки упирается в лимиты и очереди облачных раннеров GitHub Actions: минуты тарифа сгорают, сборки ждут агентов десятками минут, а CI не достаёт до внутренних ресурсов — registry, тестовых БД, стендов за VPN. Мы переносим сборки на собственные раннеры: убираем плату за минуты, даём CI доступ во внутренний контур и контролируем скорость железом.

Почему это важно бизнесу

- Плата за минуты растёт с командой: Linux $0.008/мин, Windows x2, macOS x10 — при активном CI это сотни долларов в месяц
- Очередь на раннеры — это простой разработчиков: PR, собирающийся 20 минут вместо 5, тормозит каждый релиз
- Облачный раннер не попадает во внутренний контур: закрытый registry, тестовые базы и стенды за VPN ему недоступны
- Небрежно подключённый self-hosted раннер — исполнение чужого кода внутри сети компании
- Свой раннер снимает лимит длительности job — долгие интеграционные прогоны перестают падать по таймауту

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

2.329.0+

минимальная версия агента для регистрации; дальше — обновление в 30 дней с релиза, держим в мо…

по докам actions/runner

0.14.0

ARC: чарт gha-runner-scale-set — автоскейлинг эфемерных подов-раннеров в Kubernetes, с 0.14 —…

по докам ARC

443 исх.

только исходящий HTTPS к github.com, api.github.com и *.actions.githubusercontent.com, входящи…

по докам GitHub Actions

1 job

один агент = одна job; параллелизм — N экземпляров, 250–500 МБ RAM на каждый

наш стандарт

60 мин

срок жизни registration-токена — регистрацию автоматизируем через REST API и JIT-конфиг

по докам GitHub Actions

2/20

minRunners/maxRunners — стартовый профиль: два тёплых раннера против холодного старта

наш стандарт

РЕАЛИЗАЦИЯ 1 ИЗ 3

Базовый парк: агенты под systemd на выделенном хосте

Что настраиваем

Linux-хост (Ubuntu 24.04 LTS) в серверной клиента; раннеры уровня организации, маршрутизация job метками

Как мы это делаем

1. Отдельный пользователь без sudo: useradd github-runner; агент из tarball actions/runner, регистрация ./config.sh --unattended на URL организации
2. Метки под роли хостов: self-hosted,linux,x64,docker,high-cpu — runs-on в workflow сам выбирает раннер, lint уводим на слабый хост меткой lightweight
3. Служба через svc.sh install: юнит actions.runner.<org>.<имя>, автозапуск после ребута, логи в _diag/, статус юнита заводим в Zabbix
4. На фаерволе — только исходящий 443 к github.com, api.github.com и *.actions.githubusercontent.com; входящих правил и белого IP не требуется
5. Очистку _work/ и docker system prune --filter until=72h вешаем в cron; порог свободного места 15% — триггер в мониторинге

РЕЗУЛЬТАТ

Плата за минуты уходит, сборки получают локальный кэш зависимостей и внутренние registry напрямую; отказ раннера виден в мониторинге раньше, чем разработчики упрутся в очередь.

КЛЮЧЕВОЙ НЮАНС

Registration-токен живёт 60 минут — в Ansible-роль закладываем получение токена через REST API непосредственно перед config.sh, а не хранение его в переменных.

РЕАЛИЗАЦИЯ 2 ИЗ 3

Автоскейлинг: ARC и runner scale sets в Kubernetes

Что настраиваем

Kubernetes-кластер клиента (k3s/managed); эфемерные поды-раннеры на пиковые 10–20 параллельных job

Как мы это делаем

1. Ставим helm-чарт gha-runner-scale-set-controller в namespace arc-systems, затем gha-runner-scale-set на каждый профиль раннеров
2. Аутентификация через GitHub App (app ID + private key в secret), а не PAT — токен переживает смену сотрудников и ротацию
3. githubConfigUrl на организацию, minRunners=2 / maxRunners=20; имя scale set становится меткой для runs-on
4. Сборка образов — containerMode: kubernetes или dind в values; docker.sock хоста в поды не пробрасываем
5. Каждый job получает свежий эфемерный под, после завершения под удаляется — состояние между сборками не протекает

РЕЗУЛЬТАТ

Пиковые 15 параллельных PR разъезжаются по подам за секунды, ночью кластер держит только два тёплых раннера; узлы не заняты простаивающими агентами, стоимость масштабируется вместе с нагрузкой.

КЛЮЧЕВОЙ НЮАНС

Multilabel для scale sets появился только в ARC 0.14 — на старых версиях runs-on принимает лишь имя scale set, и привычные наборы меток ломаются при миграции со статичных раннеров.

РЕАЛИЗАЦИЯ 3 ИЗ 3

Контур безопасности раннеров

Что настраиваем

Все self-hosted раннеры клиента: сеть, права токенов, политика репозиториев

Как мы это делаем

1. Self-hosted никогда не подключаем к публичным репозиториям; для приватных включаем Require approval for first-time contributors
2. В workflow задаём permissions: contents: read по умолчанию; write-права GITHUB_TOKEN выдаём точечно на уровне job
3. Раннеры выносим в отдельный VLAN: доступ только к нужным registry и стендам, продовый сегмент закрыт ACL
4. Разовые и чувствительные задачи — эфемерные раннеры: ./config.sh --ephemeral, агент снимается с учёта после одной job
5. Runner groups (план Team/Enterprise) ограничивают, какие репозитории видят какие группы раннеров

РЕЗУЛЬТАТ

Компрометация workflow не даёт закрепиться: эфемерный раннер умирает вместе с job, сеть не пускает дальше своего сегмента, а токен без write не может подменить код или релизы.

КЛЮЧЕВОЙ НЮАНС

Раннер исполняет код любого, у кого есть write в репозиторий, — считаем его недоверенным узлом сети и проектируем сегментацию до подключения, а не после первого инцидента.

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- ✕ **docker.sock внутри контейнера раннера**
  Проброс /var/run/docker.sock равен root на хосте — вместо него DinD с отдельным daemon, rootless Docker или containerMode: kubernetes в ARC

- ✕ **Раннер на публичном репозитории**
  PR из форка исполнит произвольный код на вашем железе; self-hosted оставляем только приватным репо плюс approval для новых контрибьюторов

- ✕ **Протухший registration-токен**
  Токен живёт 60 минут — скрипты деплоя падают; берём его REST-запросом registration-token прямо перед config.sh или используем JIT-конфиг

- ✕ **Очередь при живом раннере**
  Один агент обслуживает одну job; для параллелизма ставим N экземпляров в отдельных каталогах с уникальными именами

- ✕ **Автообновление за прокси**
  Агент старее минимума 2.329.0 не зарегистрируется, а за прокси автообновление молча ломается — ставим --disableupdate и обновляем сами не реже чем ра…

- ✕ **PAT в values Helm-чарта**
  Личный токен умирает вместе с сотрудником и светится в values; для ARC регистрируем GitHub App с правами на self-hosted runners организации

- ✕ **Диск, съеденный _work и образами**
  Каждый checkout и docker build копят гигабайты; cron-очистка _work/, docker system prune по расписанию и алерт на 15% свободного места

- ✕ **Секреты в окружении хоста**
  Переменные хоста читает любой job; секреты держим только в Settings → Secrets и передаём через контекст secrets, а не через профиль пользователя

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- Раннер от отдельного пользователя без sudo, служба через svc.sh, логи в _diag/
- Только приватные репозитории + approval для первых контрибьюторов
- Исходящий 443 к github.com, api.github.com и *.actions.githubusercontent.com, входящ…

НОРМАЛЬНО

- Раннеры уровня организации, маршрутизация метками по ролям хостов
- Мониторинг: юнит службы, диск, версия агента не ниже минимальной
- Cron-очистка _work/ и docker prune, порог диска 15% в алертинге
- Секреты только в GitHub Secrets, permissions: contents: read по умолчанию

ХОРОШО

- ARC в Kubernetes: эфемерные поды, minRunners/maxRunners под профиль нагрузки
- Аутентификация через GitHub App вместо PAT, ротация ключей
- Отдельный VLAN для раннеров, runner groups по чувствительности репо
- Эфемерные раннеры для всего: одна job — один чистый агент

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Подключены ли self-hosted раннеры только к приватным репозиториям?
- [ ] Работает ли агент от отдельного пользователя без sudo и без docker.sock хоста?
- [ ] Версия агента не ниже минимальной (2.329.0) и обновляется не реже чем раз в 30 дней?
- [ ] Есть ли алерт на упавшую службу раннера и на заполнение диска рабочего каталога?
- [ ] Ограничены ли права GITHUB_TOKEN до contents: read там, где запись не нужна?
- [ ] Вынесены ли раннеры в отдельный сетевой сегмент без доступа к продовому контуру?
- [ ] Автоматизирована ли регистрация раннеров (REST API/JIT) вместо ручной вставки токена?
- [ ] Хватает ли параллелизма: число агентов соответствует пиковым одновременным job?
- [ ] Настроен ли автоскейлинг (ARC) для пиков вместо закупки простаивающего железа?

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 парк раннеров под ключ: физика или Kubernetes, метки, runner groups, мониторинг
- Мигрируем workflow с облачных минут на self-hosted без остановки разработки
- Ставим ARC с автоскейлингом и GitHub App-аутентификацией в ваш кластер
- Строим контур безопасности: сегментация сети, эфемерные раннеры, аудит прав токенов
- Подключаем раннеры к внутренним registry (Harbor/GitLab) и кэшу зависимостей

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 Hosting your own runners — Self-hosted runners (docs.github.com — 2026)

02 Deploying runner scale sets with Actions Runner Controller (ARC 0.14.0) (docs.github.com — 2026)

03 Security hardening for GitHub Actions (docs.github.com — 2026)

04 actions/runner — Releases, минимальная версия агента 2.329.0 (github.com — 2026)

05 REST API: Self-hosted runners (registration-token, JIT) (docs.github.com — 2026)

06 Шаблон ITfresh: Ansible-роль gha-runner (systemd) (itfresh.ru — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.