Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

OpenFaaS и Knative: serverless на своих серверах

Разворачиваем FaaS на своём Kubernetes: OpenFaaS (чарт 15.x) и Knative 1.22 со scale to zero

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

Заказчику нужна событийная обработка — вебхуки CRM, конвертация документов, выгрузки 1С, интеграции, — но облачный FaaS недоступен или дорог, а самописные cron-скрипты и вечно крутящиеся pods жгут ресурсы и падают без ретраев. Мы разворачиваем serverless на собственном Kubernetes: OpenFaaS или Knative со scale to zero, очередями и канареечными выкатами.

Почему это важно бизнесу

- Железо не жгётся впустую: фоновые функции масштабируются в 0 подов вне нагрузки — на том же кластере живёт больше сервисов
- Нет vendor lock-in и валютных счетов за облачный FaaS: код и данные остаются в контуре компании, включая персональные данные
- Интеграции перестают теряться: очередь с ретраями и dead-letter вместо cron-скрипта, который молча упал в три ночи
- Выкат фич без простоя: канареечные 10% трафика и откат правкой процентов — релиз не роняет продажи
- Один стандарт деплоя для всех задач автоматизации: меньше зоопарка скриптов — дешевле поддержка и передача дел

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

1.22

Knative Serving + Eventing: релиз, который мы ставим; апгрейды — по квартальному релиз-циклу

релиз-цикл knative.dev

15.x

Helm-чарт openfaas (faas-netes): gateway, оператор, queue-worker и Prometheus в ns openfaas

чарт openfaas, faas-netes

60 с

stable-window автоскейлера KPA — окно усреднения нагрузки; короче — реплики дёргаются

config-autoscaler, Knative

30 с

scale-to-zero-grace-period: сколько держится маршрут через activator до остановки пода

config-autoscaler, Knative

100

container-concurrency-target-default: целевая конкурентность на под, база расчёта KPA

config-autoscaler, Knative

5 RPS

порог алерта APIHighInvocationRate на функцию в OpenFaaS CE — автоскейлер добавляет по 20% от…

docs.openfaas.com, autoscaling

РЕАЛИЗАЦИЯ 1 ИЗ 3

OpenFaaS на компактном кластере K3s

Что настраиваем

3-нодовый K3s заказчика; namespace openfaas (ядро платформы) + openfaas-fn (функции)

Как мы это делаем

1. helm install openfaas/openfaas (чарт 15.x): gateway, faas-netes, queue-worker, Prometheus + Alertmanager; generateBasicAuth=true, пароль — в secret basic-auth
2. Функции собираем из официальных шаблонов python3-http / node20 на of-watchdog; образы — в приватный Harbor, деплой из stack.yml командами faas-cli build/push/deploy
3. Автоскейл: labels com.openfaas.scale.min=1 и com.openfaas.scale.max=20; алерт APIHighInvocationRate (>5 RPS на функцию) — добавляется по 20% от max реплик (scale.fa...
4. Тяжёлые задачи — через /async-function/*: в CE очередь NATS с max_inflight=1, параллелизм — репликами queue-worker; результат — колбэком по заголовку X-Callback-Url

РЕЗУЛЬТАТ

Пакетная обработка (конвертация PDF, выгрузки 1С, вебхуки CRM) уезжает из cron-скриптов в функции: деплой одной командой, метрики per-function в Prometheus, разбор очередью. Ядро OpenFaaS занимает доли vCPU и сотни МБ RAM — живёт даже на трёх нодах по 8 ГБ.

КЛЮЧЕВОЙ НЮАНС

Scale-to-zero (label com.openfaas.scale.zero=true), ретраи очереди и max_inflight до 50 на JetStream queue-worker — функции коммерческих редакций; в CE минимальная реплика крутится всегда, а очередь без ретрае...

РЕАЛИЗАЦИЯ 2 ИЗ 3

Knative Serving: автоскейл и канареечные релизы

Что настраиваем

Prod-кластер Kubernetes актуальной minor-версии; Knative Serving 1.22 + Kourier как networking-слой

Как мы это делаем

1. kubectl apply serving-crds.yaml, затем serving-core.yaml (v1.22), поверх — net-kourier; в configmap config-network: ingress-class=kourier.ingress.networking.knative…
2. В config-autoscaler фиксируем базовую линию: stable-window 60s, scale-to-zero-grace-period 30s, container-concurrency-target-default 100
3. Каждый сервис — Knative Service с аннотациями autoscaling.knative.dev/min-scale=0 и max-scale=50; латентно-критичным ставим min-scale=1 против cold start
4. Канарейка через ревизии: в spec.traffic делим 90/10 между revision'ами, откат — правкой процентов без пересборки и редеплоя

РЕЗУЛЬТАТ

Сервисы автоматически живут от 0 до 50 подов по фактической конкурентности, ночью кластер свободен под бэкапы и отчёты. Каждый выкат — новая ревизия с историей: канареечные 10% ловят регресс до того, как его увидят все пользователи.

КЛЮЧЕВОЙ НЮАНС

При малом трафике запросы идут через activator — это плюс к латентности. Смотрим в доке параметр target-burst-capacity и настраиваем его под профиль нагрузки, прежде чем жаловаться на «медленный Knative».

РЕАЛИЗАЦИЯ 3 ИЗ 3

Knative Eventing: события из Kafka в функции

Что настраиваем

Событийная шина заказчика на Kafka 3.x; Knative Eventing 1.22 + Kafka Broker как data plane

Как мы это делаем

1. kubectl apply eventing-crds.yaml → eventing-core.yaml (v1.22); в прод ставим Kafka Broker — события хранятся в Kafka, а не в InMemoryChannel
2. KafkaSource: consumerGroup, bootstrapServers, список topics → sink на Knative Service; параллелизм обработки задаём числом партиций топика
3. На Trigger настраиваем delivery: retry=5, backoffPolicy=exponential, deadLetterSink — отдельная функция-карантин с алертом в Telegram
4. Мониторим лаг consumer group и p95 cold start первой обработки; если SLA на реакцию <2 с — держим у обработчика min-scale=1

РЕЗУЛЬТАТ

Каждое сообщение в топике гарантированно доходит до обработчика: сбойные события не теряются, а оседают в dead-letter с алертом. Пиковые всплески (закрытие месяца, массовая выгрузка) разбираются автоскейлом без ручного вмешательства дежурного.

КЛЮЧЕВОЙ НЮАНС

Доставка у брокера — at-least-once: обработчики пишем идемпотентными, дедупликация по id события. Это требование архитектуры функций, и закладывать его надо на этапе кода, а не после первого двойного списания.

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- ✕ **Cold start на латентных сервисах**
  Scale to zero даёт 1–5 с на первый ответ; API с жёстким SLA держим на min-scale=1, в ноль уводим только фоновые и событийные обработчики.
- ✕ **OpenFaaS CE вместо нужной редакции**
  Scale-to-zero, ретраи очереди, max_inflight>1 и IAM — в коммерческих редакциях; сверяем матрицу фич по докам до старта, чтобы не переезжать посреди п…
- ✕ **InMemoryChannel в проде**
  Дефолтный in-memory канал Eventing теряет события при рестарте пода; в прод ставим только Kafka Broker/Channel с персистентным хранением.
- ✕ **Функции без requests/limits**
  KPA скейлит по конкурентности, не по ресурсам: без requests планировщик переподпишет ноды. Ставим лимиты и ResourceQuota на openfaas-fn.
- ✕ **Апгрейд Knative через версию**
  Обновляем строго по одной minor: сначала *-crds.yaml, затем core. Прыжок через версии ломает conversion-webhook'и и вешает кластер.
- ✕ **Activator как скрытое звено**
  На малом трафике activator проксирует каждый запрос и добавляет миллисекунды; target-burst-capacity настраиваем под профиль нагрузки.
- ✕ **Ретраи без идемпотентности**
  NATS JetStream и Kafka доставляют at-least-once: повтор не должен создать второй заказ. Дедупликация по id события в начале handler'а.
- ✕ **Gateway наружу без защиты**
  Gateway OpenFaaS не публикуем в интернет напрямую: basic-auth обязателен, доступ — через ingress с TLS и ограничением по IP.

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- OpenFaaS по Helm-чарту 15.x на K3s; basic-auth, namespaces openfaas/openfaas-fn
- Функции из официальных шаблонов of-watchdog; образы — в приватном registry
- requests/limits на каждую функцию, Prometheus-алерт на 5xx у gateway

НОРМАЛЬНО

- Knative Serving 1.22 + Kourier; min/max-scale аннотациями на каждый сервис
- Async-вызовы через queue-worker; ретраи и max_inflight до 50 — JetStream queue-worke…
- Канареечный выкат traffic-split по ревизиям; CI-деплой faas-cli/kn из git
- Тюнинг config-autoscaler: stable-window, grace-period, целевая конкурентность

ХОРОШО

- Knative Eventing + Kafka Broker: Trigger с retry и deadLetterSink-карантином
- Дашборд Grafana: RPS, конкурентность, p95 cold start, лаг consumer group
- Регламент квартальных апгрейдов Knative и чарта OpenFaaS по релиз-циклу
- Идемпотентность обработчиков и дедупликация событий как стандарт кода

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Разделены ли нагрузки на событийные (масштабируем в ноль) и латентно-критичные (min-scale=1)?
- [ ] Выбрана ли редакция OpenFaaS с учётом матрицы фич — scale-to-zero и ретраи очереди в CE отсутствуют?
- [ ] Стоят ли requests/limits у каждой функции и ResourceQuota на namespace функций?
- [ ] Закрыт ли gateway аутентификацией и TLS, нет ли функций, торчащих в интернет без auth?
- [ ] Идемпотентны ли обработчики очередей и настроен ли deadLetterSink с алертом?
- [ ] Замерен ли p95 cold start и укладывается ли он в SLA ответа для каждого сервиса?
- [ ] Есть ли канареечный сценарий выката и откат правкой traffic-процентов за минуты?
- [ ] Хранится ли data plane событий в Kafka, а не в InMemoryChannel?
- [ ] Расписан ли график апгрейдов: Knative — квартальные релизы, шаг строго в одну minor?
- [ ] Собираются ли per-function метрики в Prometheus и видны ли они дежурному в Grafana?

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 нагрузок и выбор платформы: OpenFaaS или Knative под ваш кластер и команду
- Разворачиваем ядро под ключ: Helm/CRD, приватный registry, CI-пайплайн faas-cli или kn
- Переносим cron-скрипты, вебхуки и выгрузки 1С в функции с очередями и ретраями
- Настраиваем автоскейл и мониторинг: config-autoscaler, Grafana, алерты в Telegram
- Сопровождаем: квартальные апгрейды, ревизия лимитов, контроль cold start и dead-letter

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 Knative Serving — Autoscaling (KPA, config-autoscaler) (knative.dev — v1.22)

02 Knative — Install Serving with YAML + net-kourier (knative.dev — v1.22)

03 Knative Eventing — Kafka Broker, Sources, Delivery (knative.dev — v1.22)

04 OpenFaaS Docs — Architecture: Gateway, Autoscaling (docs.openfaas.com — 2026)

05 OpenFaaS Docs — Async invocations, JetStream queue-worker (docs.openfaas.com — 2026)

06 Helm-чарт openfaas (faas-netes) (github.com/openfaas — 15.x)

07 Наш шаблон CI-деплоя функций (stack.yml + GitLab CI) (itfresh.ru — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.