Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

API Gateway с нуля: проектирование и реализация

Как мы проектируем шлюз на Go: маршрутизация, JWT, rate limiting, circuit breaker, деплой

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

У заказчика интеграции с партнёрскими API размазаны по сервисам: авторизация, лимиты и логирование в каждом реализованы по-своему, а сбой одного внешнего API валит всю цепочку обмена. Готовые шлюзы не всегда закрывают маршрутизацию по телу запроса и лимиты по бизнес-ключам. Мы проектируем и внедряем собственный API Gateway на Go — единую управляемую точку входа с бюджетом задержки p99 ≤ 5 мс.

Почему это важно бизнесу

- Единая точка входа даёт контроль доступа и аудит всех API-вызовов — вместо зоопарка самописных проверок в каждом сервисе
- Rate limiting по бизнес-ключу гасит всплески и защищает от каскадного отказа и лишних расходов на инфраструктуру
- Circuit breaker изолирует сбой одного партнёра — обмен заказами и платежами по остальным направлениям продолжается
- Hot-reload конфигурации и rolling update: новые маршруты и лимиты — без остановки продаж и окон обслуживания
- Шлюз с p99 ≤ 5 мс не съедает общий бюджет латентности в 100 мс, который держат клиентские приложения

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

Go 1.26

версия компилятора шлюза: текущий стабильный релиз (1.26.4 с security-фиксами crypto/x509)

go.dev, релиз 06.2026

jwt v5.3.1

golang-jwt: WithValidMethods([RS256, ES256]) + WithLeeway(5 с) против подмены алгоритма

pkg.go.dev/golang-jwt/jwt/v5

Redis 8.x

sliding window и кэш ответов; клиент go-redis v9, окно лимита — одним Lua-скриптом

redis.io, go-redis v9

p99 ≤ 5 мс

бюджет задержки самого шлюза — закладываем при проектировании и держим нагрузочными тестами

наш стандарт

45 с

terminationGracePeriodSeconds > таймаута srv.Shutdown (30 с): под дорабатывает соединения

Kubernetes 1.36

5 с

интервал health check upstream и periodSeconds readinessProbe: мёртвая нода уходит из ротации

наш стандарт

РЕАЛИЗАЦИЯ 1 ИЗ 3

Маршрутизация и hot-reload конфигурации

Что настраиваем

Пакет router на Go 1.26: radix tree + atomic.Pointer, weighted round-robin с health check

Как мы это делаем

1. Описываем маршруты в routes.yaml: path, methods, upstream с весами (weight 70/30), цепочка middleware задаётся на каждый маршрут отдельно
2. Строим radix tree по маршрутам и публикуем через atomic.Pointer — ServeHTTP читает дерево без блокировок, поиск за O(k) от длины пути
3. SIGHUP → полная валидация нового YAML → атомарная подмена дерева r.tree.Store(); при ошибке конфига остаёмся на рабочей версии
4. Балансировка smooth weighted round-robin; каждые 5 с опрашиваем /healthz каждого upstream и снимаем флаг alive у не ответивших
5. Маршрутизацию по телу (route_by_body по полю event_type) включаем только для webhook-путей — там формат события определяет получателя

РЕЗУЛЬТАТ

Изменение маршрутов, весов и лимитов вкатывается за секунды без рестарта процесса и обрыва соединений; сбойная нода upstream выпадает из ротации за 5–10 с без участия дежурного

КЛЮЧЕВОЙ НЮАНС

Сначала полностью валидируем новый конфиг, потом атомарно подменяем; в hot path никаких mutex — только atomic.Pointer, иначе под нагрузкой ловим contention на каждом запросе

РЕАЛИЗАЦИЯ 2 ИЗ 3

JWT + RBAC и rate limiting на Redis

Что настраиваем

Пакеты auth и limiter: golang-jwt v5.3.1, ключи по JWKS, sliding window на Redis 8.x

Как мы это делаем

1. ParseWithClaims строго с jwt.WithValidMethods([]string{"RS256","ES256"}) и jwt.WithLeeway(5*time.Second) — alg:none и HS256-подмена отсекаются на входе
2. Публичные ключи берём по kid из JWKS-эндпоинта auth-сервиса, кэшируем и обновляем в фоне; RBAC — сверка claim roles со списком ролей маршрута
3. Sliding window log в ZSET: ZRemRangeByScore → ZAdd(score=UnixMicro) → ZCard → Expire(window+1s); все четыре команды — одним Lua-скриптом через EVALSHA
4. Ключ лимита формируем из конфига маршрута: seller:{seller_id}:marketplace:{marketplace_id}, типовой порог 100 запросов / 60 с на пару

РЕЗУЛЬТАТ

Авторизация и лимиты закрываются на шлюзе без похода в auth-сервис на каждый запрос; лимиты честные на границах окон (нет всплеска x2, как у fixed window) и считаются по бизнес-ключу, а не по IP

КЛЮЧЕВОЙ НЮАНС

Pipeline в Redis не атомарен — параллельные запросы гонятся между ZAdd и ZCard; переносим последовательность в Lua-скрипт, тогда счётчик окна консистентен при любой конкуренции

РЕАЛИЗАЦИЯ 3 ИЗ 3

Circuit breaker и zero-downtime деплой в Kubernetes

Что настраиваем

Пакет circuit + Deployment в Kubernetes 1.36: graceful shutdown, RollingUpdate без потерь

Как мы это делаем

1. Breaker на каждый upstream: Closed→Open по порогу 5 ошибок подряд, таймаут Open 30 с, Half-Open пропускает 3 пробных запроса перед возвратом в Closed
2. Graceful shutdown: по SIGTERM перестаём принимать новые соединения и дорабатываем открытые через srv.Shutdown(ctx) с таймаутом 30 с
3. strategy RollingUpdate: maxUnavailable:0, maxSurge:1; terminationGracePeriodSeconds:45 — заведомо больше таймаута Shutdown
4. readinessProbe /readyz (periodSeconds:5) отдельно от livenessProbe /healthz (periodSeconds:10); resources requests 500m/128Mi, limits 2000m/512Mi
5. lifecycle.preStop sleep 5 с — endpoints и kube-proxy успевают убрать под из балансировки до начала остановки

РЕЗУЛЬТАТ

Релизы проходят без единого 502: старый под дорабатывает соединения, новый принимает трафик после готовности; сбой партнёрского API изолируется брейкером и не кладёт остальной обмен

КЛЮЧЕВОЙ НЮАНС

readiness ≠ liveness: /readyz зависит от загруженного конфига и доступности Redis, /healthz — только от живости процесса; завяжешь liveness на Redis — получишь каскадную рестарт-петлю всех подов

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- ✕ **Fixed window вместо sliding**
  На границе окон проходит до x2 лимита. Держим sliding window log на ZSET Redis: ZRemRangeByScore + ZAdd + ZCard в одном Lua-скрипте

- ✕ **Подмена алгоритма JWT**
  Без белого списка токен с alg:none или HS256 на публичном ключе проходит. Всегда jwt.WithValidMethods([RS256, ES256]), ключ строго по kid из JWKS

- ✕ **Rate limit без атомарности**
  ZAdd и ZCard в pipeline гонятся между параллельными запросами — счётчик врёт. Переносим всю последовательность в Lua через EVALSHA

- ✕ **SIGTERM рвёт соединения**
  Без srv.Shutdown(ctx) и preStop-паузы под умирает с открытыми соединениями — клиенты ловят 502. Grace-период 45 с строго больше таймаута Shutdown 30 с

- ✕ **Liveness завязана на Redis**
  Деградация Redis перезапускает все поды каскадом. В /healthz — только живость процесса; зависимость от Redis и конфига — только в /readyz

- ✕ **Кэш без учёта Vary**
  Один ключ на путь — и клиент получает ответ на чужом языке или чужие данные. Ключ кэша = хеш(путь + query + значения Vary-заголовков)

- ✕ **Mutex в hot path роутера**
  Блокировка на каждый запрос — contention под нагрузкой. Дерево маршрутов публикуем через atomic.Pointer, читаем без блокировок

- ✕ **Нет таймаутов http.Server**
  Дефолты Go — без таймаутов: slowloris держит соединения бесконечно. Задаём ReadHeaderTimeout, ReadTimeout, WriteTimeout, IdleTimeout явно

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- Готовый шлюз (Kong/Envoy/nginx), если он закрывает задачи — своё не пишем
- Явные таймауты http.Server и лимит размера тела запроса на входе
- Структурные логи каждого запроса: trace_id, маршрут, статус, длительность

НОРМАЛЬНО

- JWT RS256/ES256 по JWKS + RBAC-роли на уровне маршрута
- Sliding window rate limit в Redis по бизнес-ключу, а не по IP
- Health check upstream каждые 5 с + weighted round-robin по весам
- RollingUpdate maxUnavailable:0 + graceful shutdown по SIGTERM

ХОРОШО

- Circuit breaker на каждый upstream + кэш GET-ответов с учётом Vary
- Hot-reload конфигурации маршрутов по SIGHUP без рестарта процесса
- Сквозной трейсинг W3C traceparent (OpenTelemetry) во все upstream-вызовы
- Нагрузочный прогон p99/p99.9 и профиль RAM/CPU перед каждым релизом

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Зафиксирован ли бюджет задержки самого шлюза (у нас p99 ≤ 5 мс) и проверяется ли он нагрузочным тестом?
- [ ] Проверяются ли JWT строго по белому списку алгоритмов (WithValidMethods), а ключи — по kid из JWKS?
- [ ] Ограничен ли rate limit по бизнес-ключу (клиент/партнёр), а не только по IP?
- [ ] Переживает ли шлюз деплой без единого 502 (maxUnavailable:0 + graceful shutdown по SIGTERM)?
- [ ] Отключается ли сбойный upstream автоматически: health check каждые 5 с + circuit breaker?
- [ ] Обновляется ли конфигурация маршрутов без рестарта процесса (hot-reload по SIGHUP)?
- [ ] Разделены ли /healthz и /readyz и не завязана ли liveness-проба на внешние зависимости?
- [ ] Есть ли trace_id (W3C traceparent) в каждом лог-событии и прокидывается ли он в upstream?
- [ ] Заданы ли ReadHeaderTimeout/ReadTimeout/WriteTimeout/IdleTimeout у http.Server и лимит размера тела?
- [ ] Учитывает ли ключ кэша GET-ответов Vary-заголовки (язык, арендатор), чтобы не отдать чужой ответ?

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 текущих интеграций и честный выбор: готовый шлюз (Kong/Envoy/nginx) или кастомный на Go — с расчётом стоимости владения
- Проектируем и пишем шлюз под ключ: маршрутизация, JWT/RBAC, rate limiting, circuit breaker, кэширование
- Разворачиваем в Kubernetes с zero-downtime деплоем, пробами и мониторингом p99-латентности
- Нагрузочное тестирование до боевого трафика: профили RPS, p99/p99.9, потребление CPU/RAM на ноду
- Сопровождаем после запуска: дежурный мониторинг, доработка маршрутов и лимитов под новых партнёров

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 net/http: Server (таймауты), Shutdown — graceful остановка (pkg.go.dev — Go 1.26)

02 golang-jwt/jwt v5: ParseWithClaims, WithValidMethods, WithLeeway (pkg.go.dev — v5.3.1)

03 Redis: Sorted Set (ZADD/ZREMRANGEBYSCORE/ZCARD), Pipelining, Lua scripting (redis.io — 8.x)

04 go-redis v9: Pipeline и выполнение скриптов (redis.io — v9)

05 Kubernetes: Deployments (RollingUpdate), Pod Lifecycle, Liveness/Readiness Probes (kubernetes.io — v1.36)

06 W3C Trace Context: заголовок traceparent (w3.org — 2021)

07 Наш шаблон routes.yaml и чек-лист приёмки шлюза (itfresh.ru — 2026)

Основано на официальной документации продуктов и нашей практике внедрения.