
ozontech

Лекция 14. Микросервисы в Ozon

Юсипов Гаяз

Разработчик информационных систем в команде пользовательского контента

OZON

Москва, 2021

ozon tech

Содержание

- общие принципы написания сервисов
- именование, документация и версионирование сервисов
- интерфейсы «REST» и «gRPC»
- обязательные методы
- пробы в Kubernetes
- логирование, трейсинг и мониторинг
- конфигурирование
- Canary и blue-green деплой
- контейнеризация, сборка и деплой
- Postgres, Kafka и Redis
- стандарты Go

ozon tech

Общие принципы

- сервисы реализуют функционал, соответствующий некой бизнес-потребности
- не следует делать сервисы слишком большими и универсальными
- Мандат Безоса (в вольной форме):
  - сервисы предоставляют доступ к данным и функциональности только через интерфейсы
  - сервисы взаимодействуют только через интерфейсы
  - интерфейсы нужно грамотно проектировать
- Graceful degradation
- Service statelessness principle
- экземпляры сервиса должны быть независимыми друг от друга
- экземпляры сервиса независимы от того, на каком физическом сервисе запущены
- быстрые, небольшие тесты, независящие от окружения
- для версионирования SemVer

ozon tech

Именование сервисов

- имя сервиса должно отражать его назначение. Например, image-storage-api — сервис, предоставляющий функции для хранения и обработки изображений
- в имени сервиса используются только маленькие латинские буквы, цифры и знак «-» (hyphen-minus, unicode U+002D)
- имя сервиса уникальное и каноническое

ozon tech

Интерфейсы «REST» и «gRPC»

- с внешним интернетом сервисы используют HTTP (без gRPC)
- между собой сервисы взаимодействуют по gRPC
- для даты и времени в ответах и запросах используется RFC 3339
- дата и время хранится и передается в UTC
- для передачи данных (не бинарных) по HTTP указывается заголовок `Content-Type: application/json`
- Proto файлы должны содержать:
  - директиву package
  - опции go_package, csharp_namespace

ozon tech

Безопасность

- во внутренней сети сервисы не требуют аутентификацию или авторизацию
- сервисы должны поддерживать протокол `HTTPS` помимо `HTTP`

ozon tech

Уровень критичности

У каждого сервиса есть уровень критичности (severity level):

- P0 — `HIGH`
- P1 — `MEDIUM`
- P2 и ниже — `LOW`

Если любое нарушение работоспособности сервиса (включая, но не ограничиваясь недоступностью сервиса или любых его частей, ухудшением времён ответа сервиса, высоким процентом ошибок в ответах сервиса) приводит к инциденту уровня P0 в продакшене — уровень критичности сервиса определяется как HIGH

ozon tech

Обязательные методы

В сервисах есть ряд HTTP методов которые должны быть доступны по debug порту:

- /live, /ready — liveness/readiness probe в k8s
- /version — общая информация
- /swagger.json — документация в формате Swagger JSON (если есть HTTP ручки)
- /docs — документация по методам сервиса для пользователя в виде Swagger UI (если есть HTTP ручки)
- /metrics — Prometheus метрики
- /channelz — информация о внутреннем состоянии gRPC-соединений
- /debug/pprof — pprof (для сервисов на Go)

ozon tech

Liveness probe

- возвращает ошибку только в том случае, если наступила проблема, которая препятствует обработке запросов и может быть решена только рестартом сервиса
- liveness probe должна быть всегда включена

Readiness probe

- отвечает за то, готов ли сервис обрабатывать запросы или нет
- частое применение - прогрев кеша или загрузка необходимых данных для работы при старте приложения
- на время загрузки (до запуска gRPC-сервера) возвращает ошибку
- нельзя добавлять внешние зависимости, т.к. это может привести к каскадному падению всех систем
- должна быть корректная обработка недоступности зависимостей (например, с помощью circuit-breaker'a)
- можно выключать руками в приложении для проведения диагностики (во время инцидентов)
- после получения SIGTERM'a сервисом, readiness probe возвращает ошибку

ozon tech

Конфигурирование

- для конфигурирования сервисов под разные окружения используются `env` переменные
- для динамического (real-time) конфига используется `etcd`
- секреты хранятся в Vault

Логирование и трейсинг

- сервисы должны либо писать логи, либо трейсы, но рекомендует и то и другое
- логи пишутся в `STDOUT` в формате структурированного JSON
- по умолчанию используется уровень `WARNING`

Пример лога:

```
{
  "ts": "2018-03T13:24:59.321",
  "message": "Your message",
  "level": "INFO",
  "trace_id":"1a93a60e17efa0bd" ,
  "span_id":"7cbc711c34a1f44d"
}
```

- трассировка на уровне запросов как минимум (span при получении запроса и его завершение после записи ответа)
- сервисы пробрасывают tracing-заголовки
- для выбора стратегии семплирования используется remote-стратегия

Метрики и мониторинг

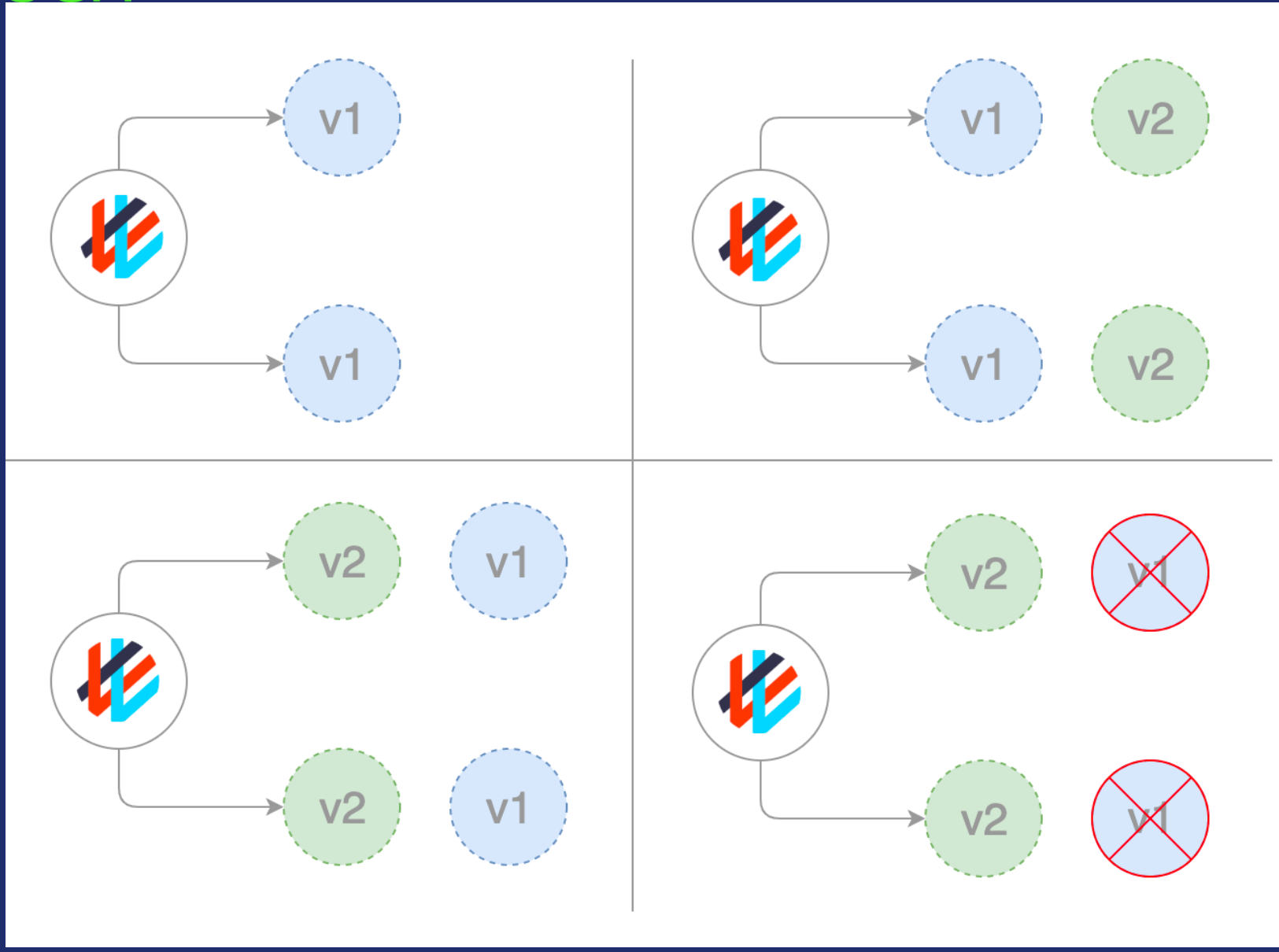
- сервисы предоставляют метрики `ozon_app_build_info` и `ozon_app_metrics_standard`
- у сервисов, которые обрабатывают входящие запросы должны быть метрики: RPS, Response Time, Error Rate
- названия метрик, лейблы и бакеты закреплены во внутреннем стандарте метрик
- названия метрик соответствуют формату <где>_<что>_<единицы_измерения>
- у гистограмм количество бакетов максимум 12
- есть запросы в другие сервисы? БД? кэш? — нужны метрики по каждому пункту
- для дискавери сервиса Kubernetes, нужен лейбл monitoring_scope: `goapi` или `dotnetapi`

Рекомендуемые наборы бакетов

```
var (
    TimeBucketsFast = []float64{0.001, 0.003, 0.007, 0.015, 0.05, 0.1, 0.2, 0.5,
    TimeBucketsMedium = []float64{0.001, 0.005, 0.015, 0.05, 0.1, 0.25, 0.5, 0.75
    TimeBucketsSlow = []float64{0.05, 0.1, 0.2, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2,
)
```

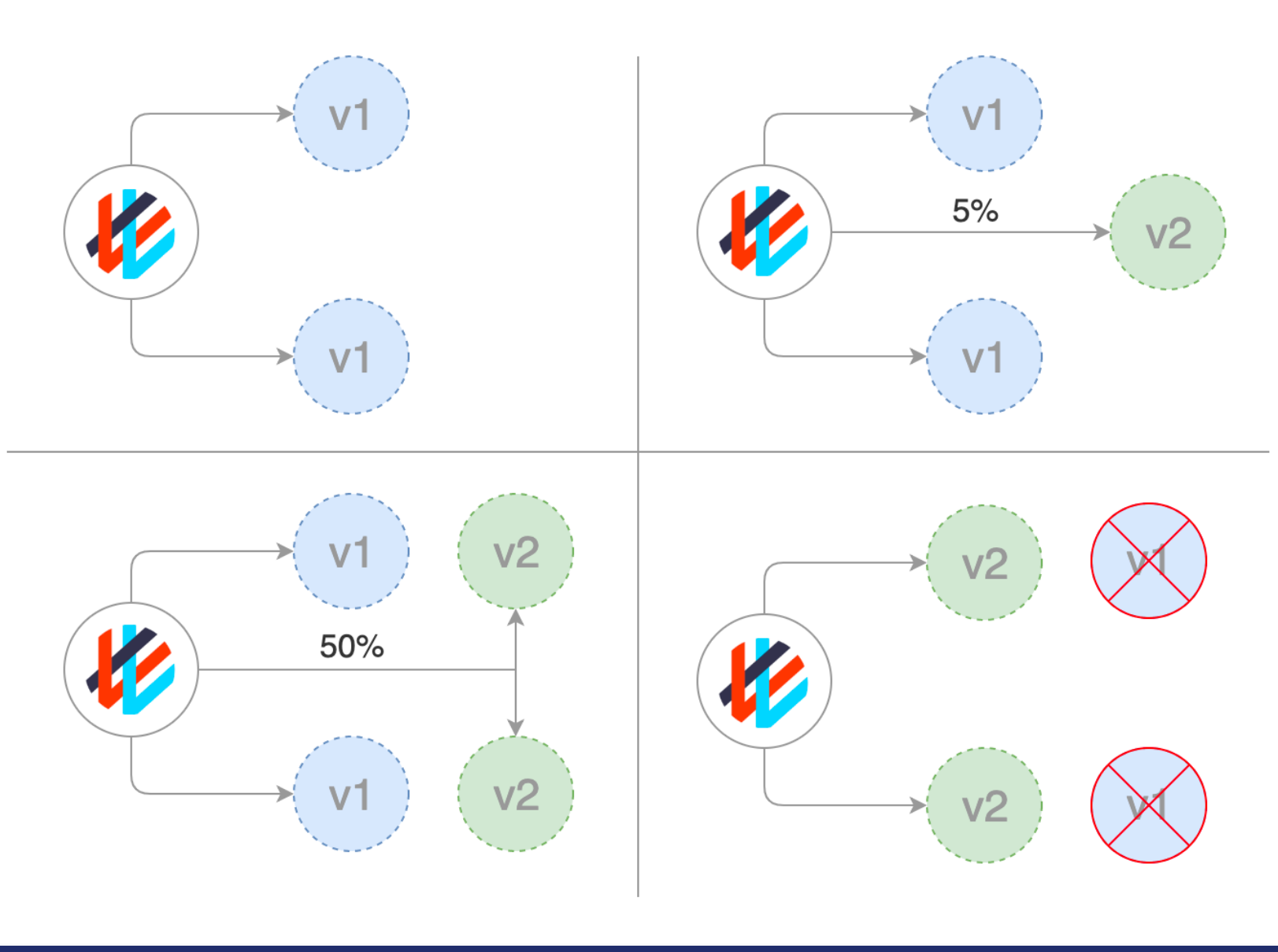
Blue-green деплой (или red-black)

ozon tech



Canary деплой (канарейка)

ozon tech



ozon tech

Сборка

Для сборки используется Gitlab-CI и общий Common pipeline

- ветки создаются от `master`
- для релизов создаются ветки `release/<name>` и в нее сливаются ветки разработки
- после сборки и теста релизной ветки прожимается `ready to prod` который проставляет тег
- после успешного деплоя в пайплайне тега релиз закрывается кнопкой `close release`
- закрывать релиз нужно убедившись что все в порядке
- после закрытия релиза, ррелизная ветка сливается в `master`
- в пайплайне для `master` ветки происходит деплой мастера на девелопмент и `latest` на стейджинг, а также переключение роута по умолчанию на `master`

ozon tech

Деплой

- релизы рекомендовано катить с 11:00 до 19:00
- хотфиксы можно катить в любое время возможны при наличии инцидента
- в хотфиксах правки только относящиеся к проблеме
- помимо хотфикса возможно откатить приложение на предыдущую версию

ozon tech

Контейнеризация

- в Docker контейнерах в качестве `entrypoint` или `cmd` только само приложение
- используем Kubernetes CronJob, никаких cron-скриптов в контейнерах
- в контейнерах нельзя запускать посторонние процессы

Postgres, Kafka и Redis

Postgres

- если Go сервис работает с базой данных, то должен использоваться Go entrypoint (пакет с клиентской балансировкой)

Kafka

- то сервис должен шардировать данные по нескольким инстансам:
  - количество шардов для каждого сервиса не меньше 2
  - рекомендуется использовать consistent hashing алгоритм (миниминизирует количество ключей, которые переедут на другой шард)
- нужно учитывать, что в любой момент времени любой (а в некоторых случаях и все) из инстансов memcached могут быть перезагружены

Redis

- можно использовать как кэш или как персистентное хранилище
- то сервис должен шардировать данные по нескольким инстансам:
  - количество шардов для каждого сервиса не меньше 2
- нужно учитывать, что в любой момент времени любой (а в некоторых случаях и все) из инстансов redis могут быть перезагружены

Большее количество шардов позволяет снизить объем потерянных данных при рестарте, это не так сильно увеличит нагрузку на основное хранилище.

ozon tech

Стандарты Go

- поддерживаются 2 последние версии языка
- обязательны модули (`go.mod`)
- есть линтер на определенные модули или определенные версии определенных модулей
- используются Athens в качестве локального прокси и кеша модуле
- для форматирования импортов используется `goimports`
- для ассертов — github.com/stretchr/testify
- для моков — github.com/gojuno/minimock
- сгенерированный код должен быть закоммичен в репозиторий
- используем внутренний framework

ozon{tech

Спасибо за внимание

- Jeff Bezos' Mandate: Amazon and Web Services
- Semantic Versioning
- A short introduction to Channelz
- etcd
- Hashicorp Vault
- Athens

Gaiaz Iusipov g.iusipov@gmail.com 2021