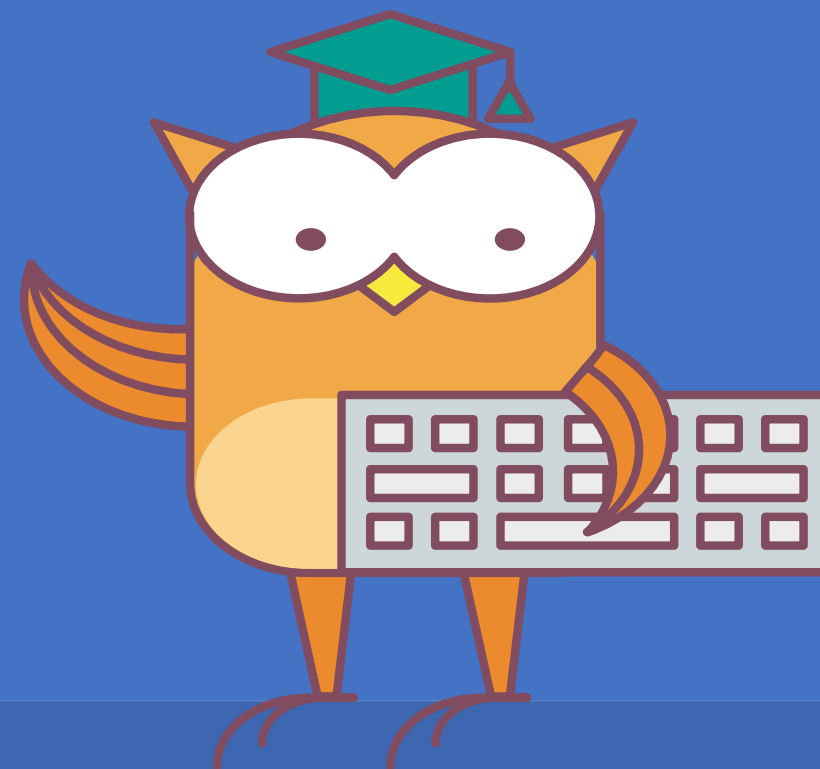
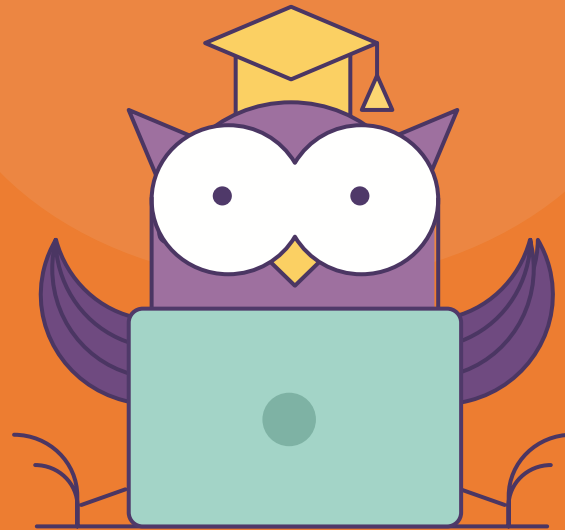


Инфраструктурные паттерны микросервисной архитектуры

Архитектор ПО



Меня хорошо
слышно && видно?



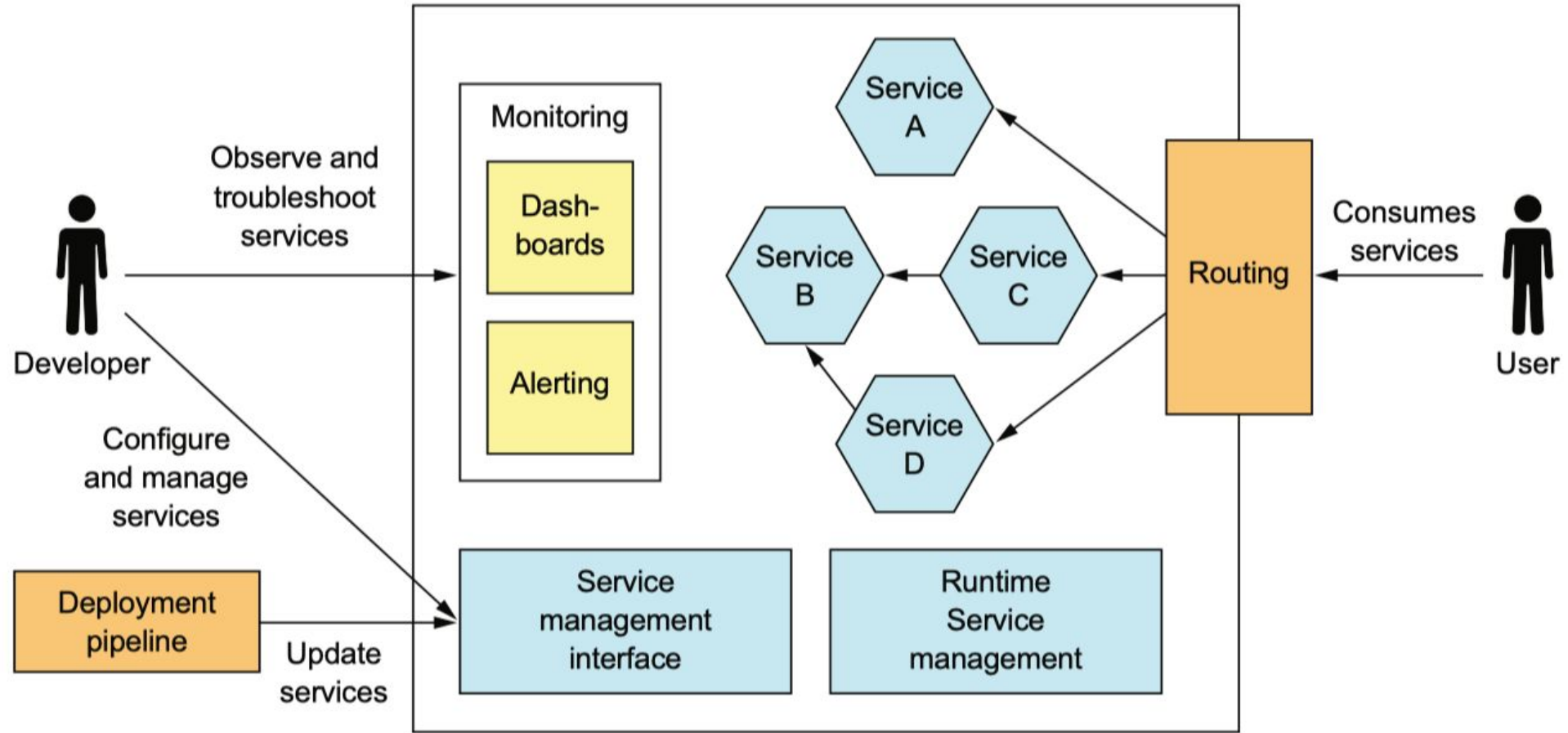
Напишите в чат, если есть проблемы!

Ставьте если все хорошо

Карта вебинара

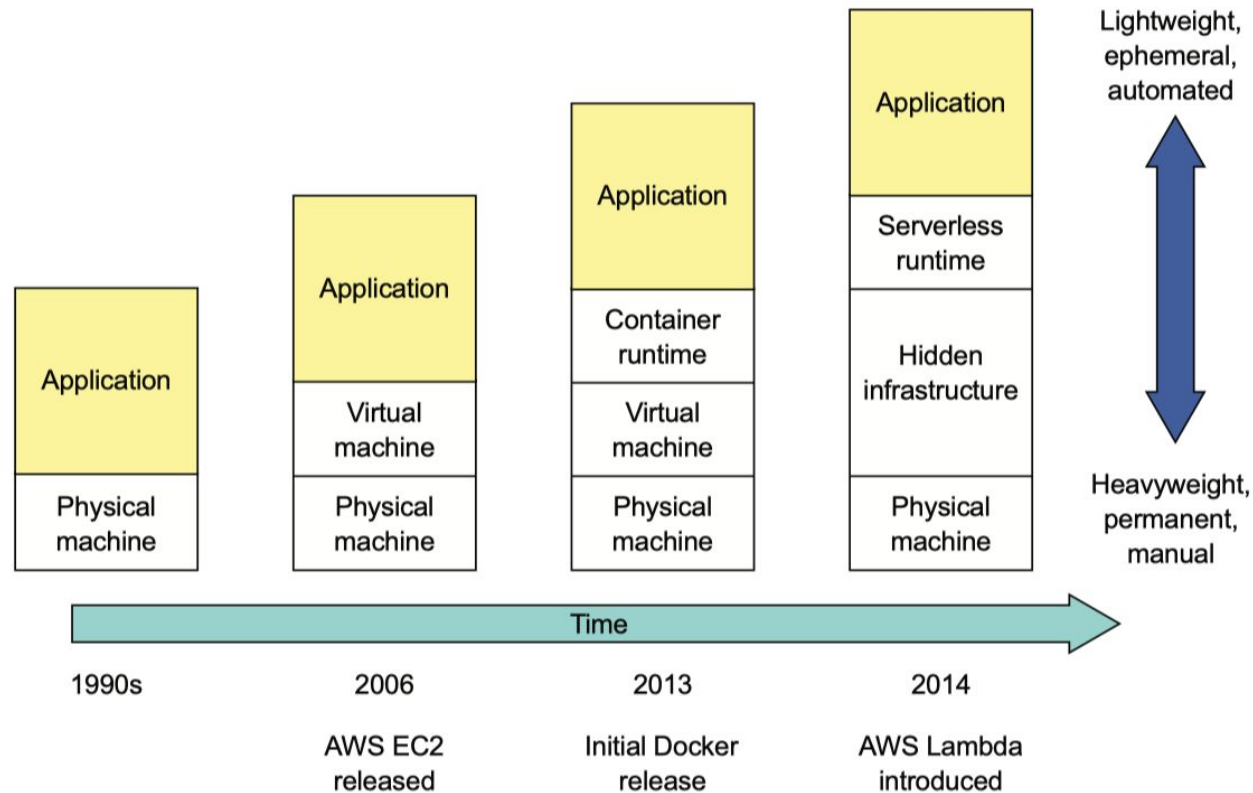
- Системы оркестрации
- App server vs virtual machine vs container
- Service discovery
- Стратегии деплоя
- Конфигурирование приложений
- Логирование
- Мониторинг и алертинга на примере прометеуса
- Распределенные транзакции
- Service mesh

Инфраструктурные паттерны микросервисной архитектуры



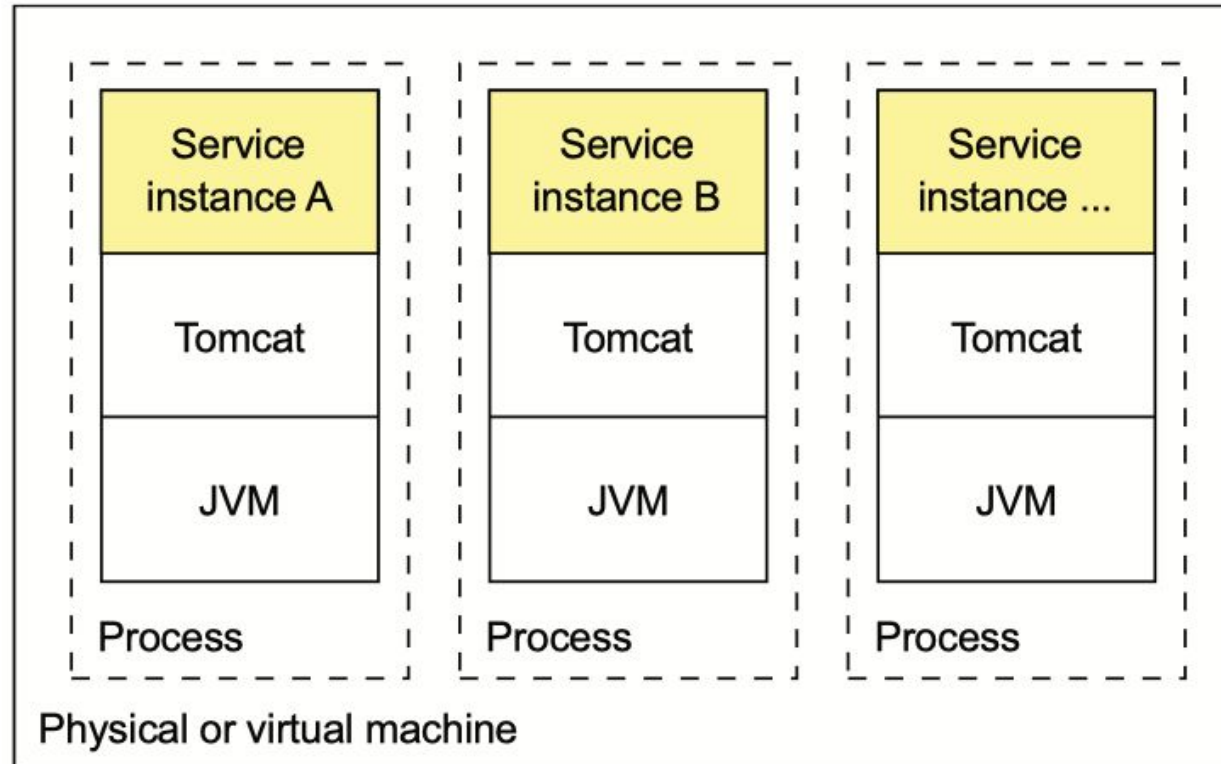
Как разместить несколько сервисов на одной машине?

- Сервер приложений jvm tomcat/jboss, python uwsgi
- Виртуальные машины Vagrant, vmware
- Контейнеры Docker, rkt



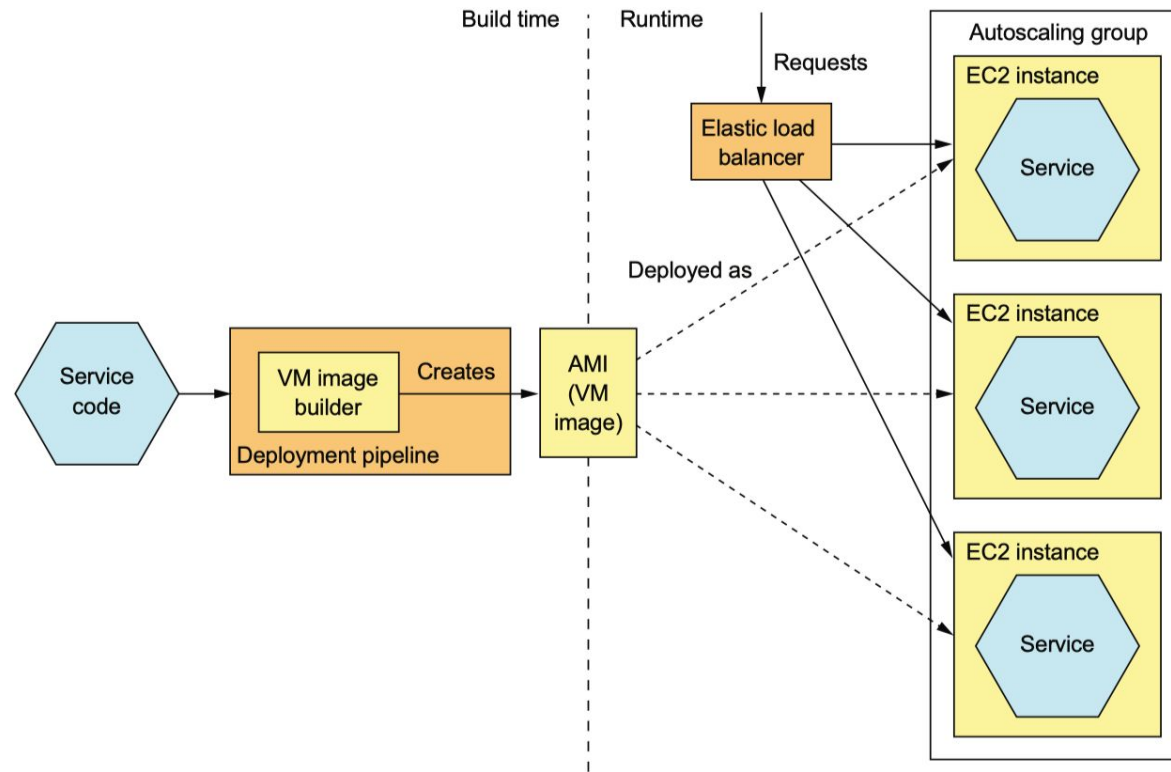
Сервер приложений

- Быстрый деплой
- Хорошая утилизация ресурсов
- Отсутствие изоляции по ресурсам между разными сервисами – CPU, memory
- Фиксированный язык программирования или фреймворк



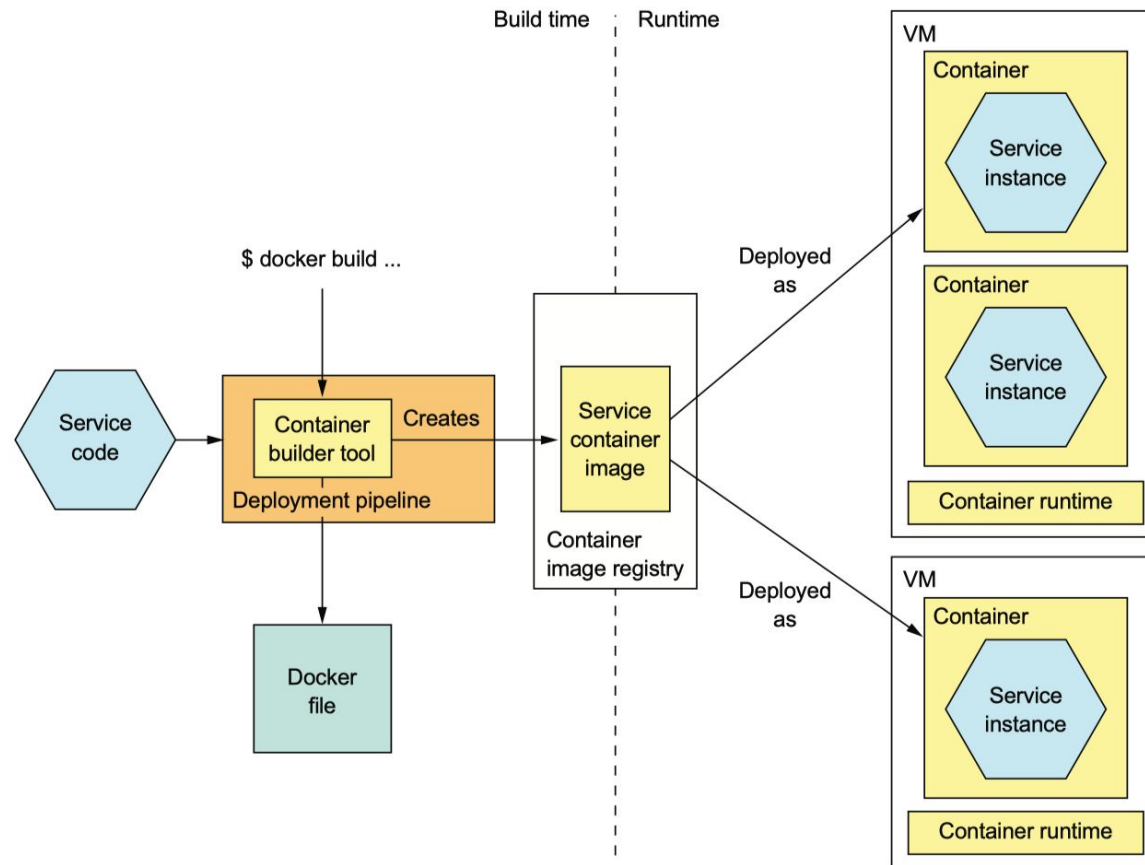
Виртуальная машина

- Technology agnostic
- Изоляция ресурсов между сервисами
- Большая утилизация ресурсов
- Долгий деплой



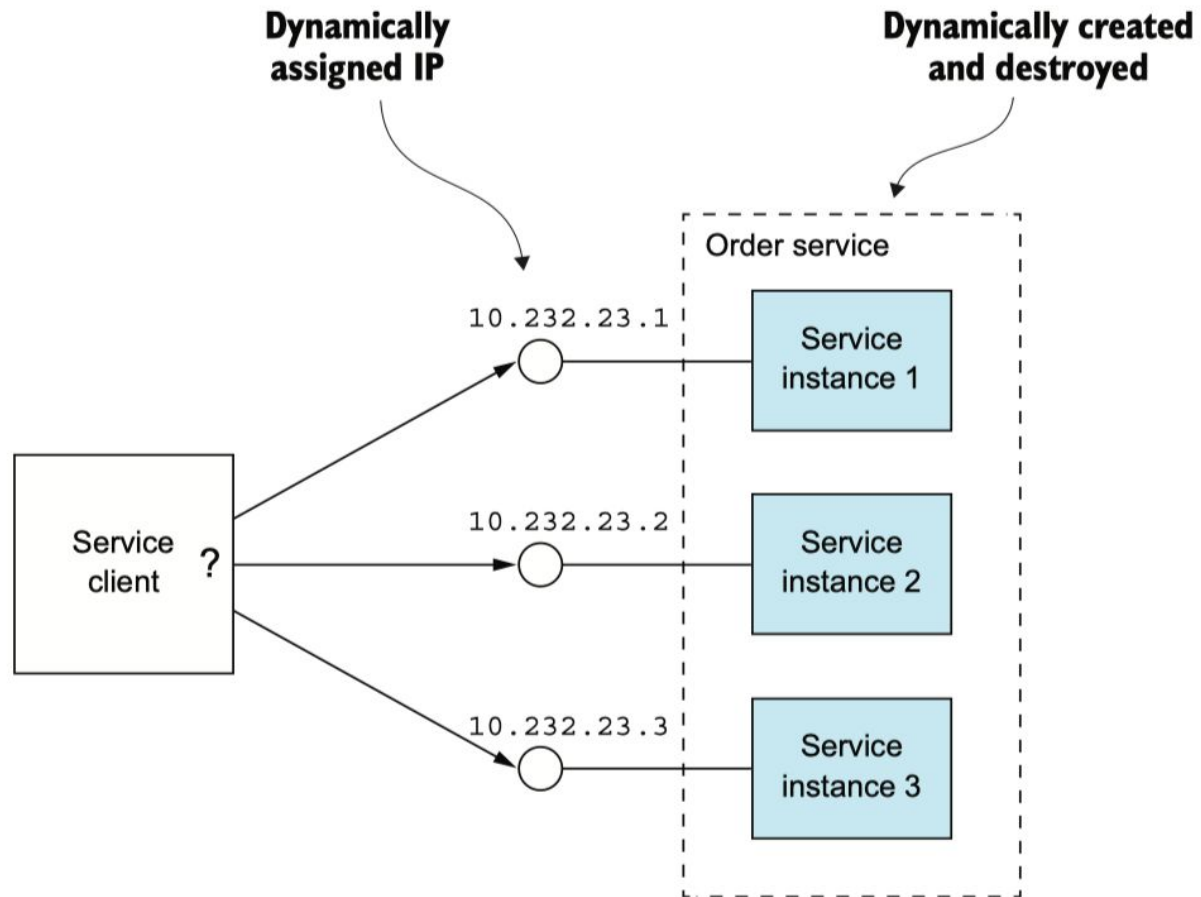
Контейнеры

- Technology agnostic
- Изоляция и ограничение сервисов друг от друга
- Эффективная утилизация ресурсов



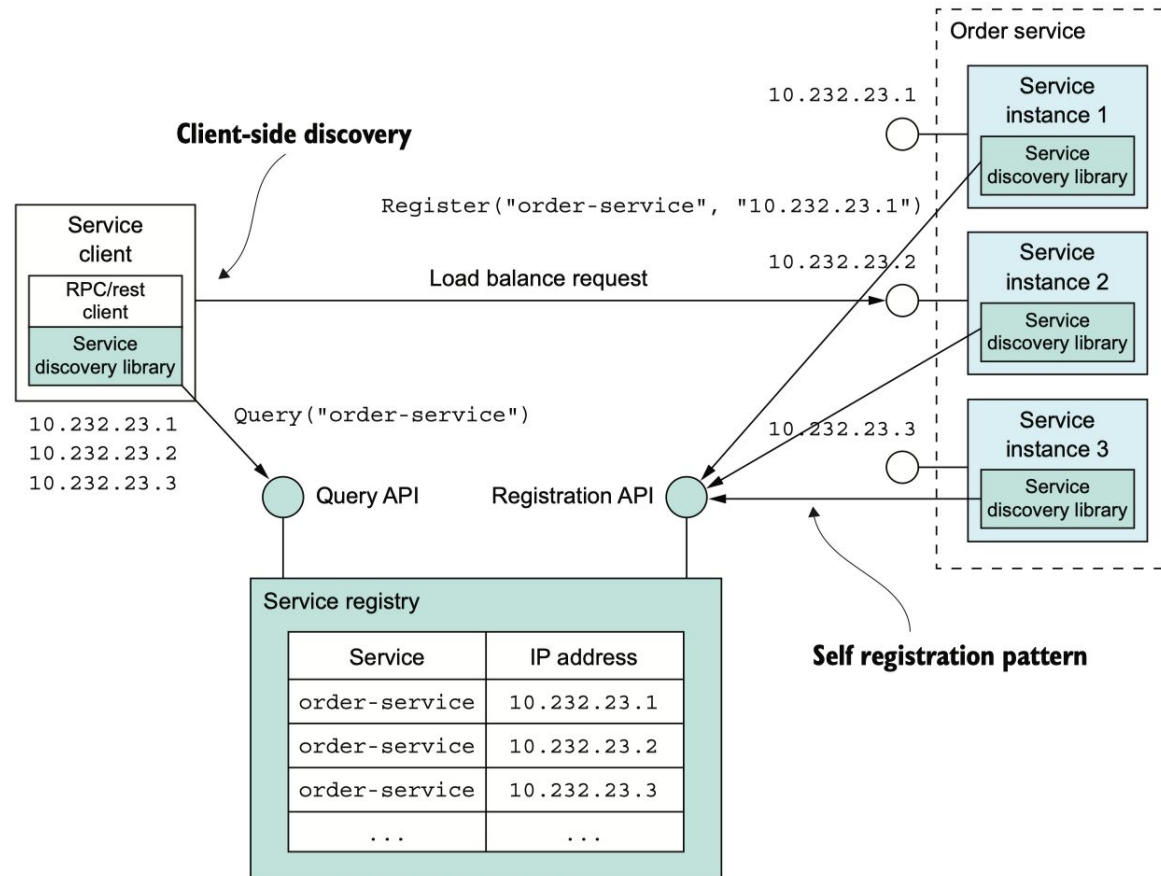
Service discovery

Как клиенту понять, где находится инстанс сервиса?



Client-side discovery

- Клиент сам ходит в реестр сервисов, получает оттуда данные
- Сервисы сами себя регистрируют в этом реестре



Eureka

- <https://github.com/Netflix/eureka>

```
@Autowired
private EurekaClient eurekaClient;

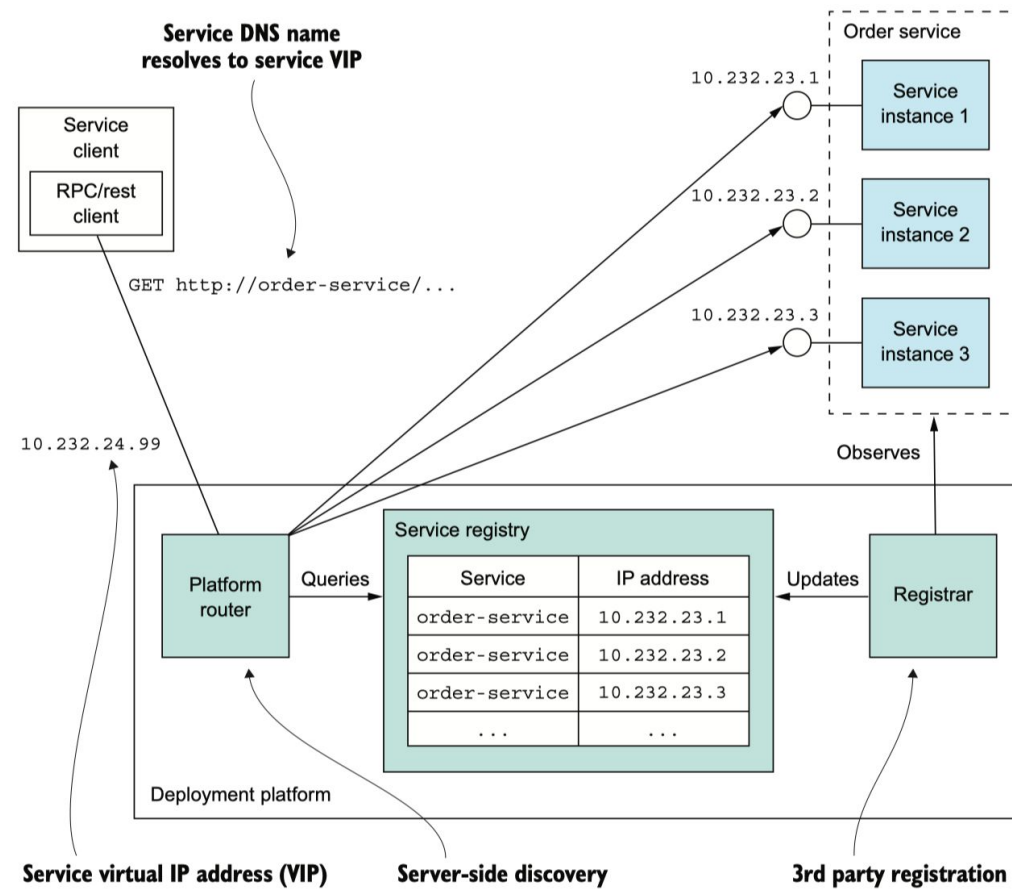
public void doRequest() {
    Application application
        = eurekaClient.getApplication("spring-cloud-eureka-client");
    InstanceInfo instanceInfo = application.getInstances().get(0);
    String hostname = instanceInfo.getHostName();
    int port = instanceInfo.getPort();
    //...
}
```

Client-side discovery

- Работает с несколькими системами оркестрации одновременно: k8s, standalone-сервисы, nomad и т.д.
- Зависит от поддержки языка программирования и фреймворка сервисов

Server-side discovery

- Оркестратор регистрирует сервис в реестре
- При обращении клиент использует service name или ip
- При обращении на этот ip роутер заглядывает в реест и перенаправляет запрос (осуществляет LoadBalancing)



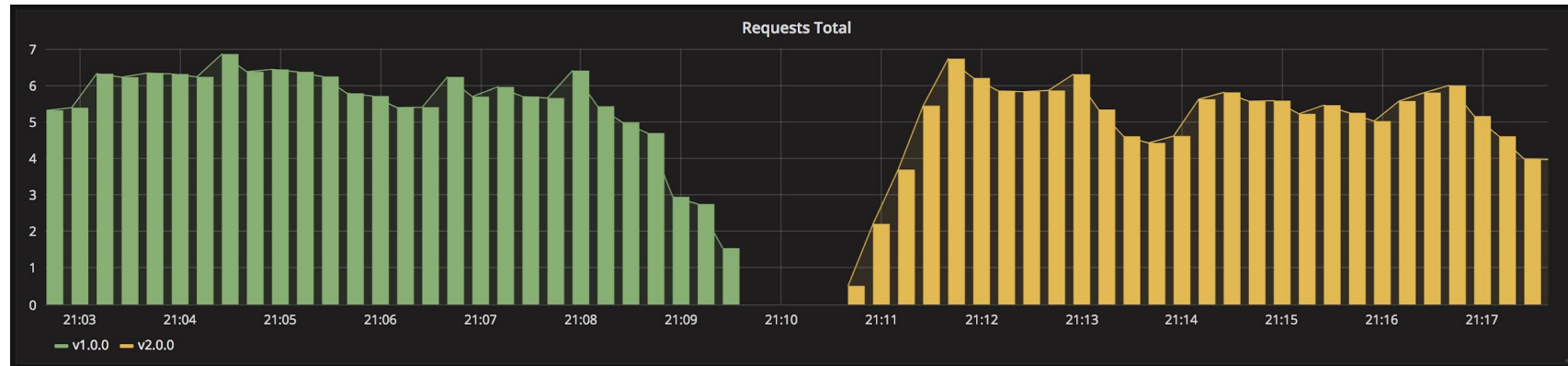
Стратегии деплоя

- Recreate
- Rolling update
- Blue/green
- Canary

Recreate

Убить существующий деплой
Поднять новый

- Даунтайм

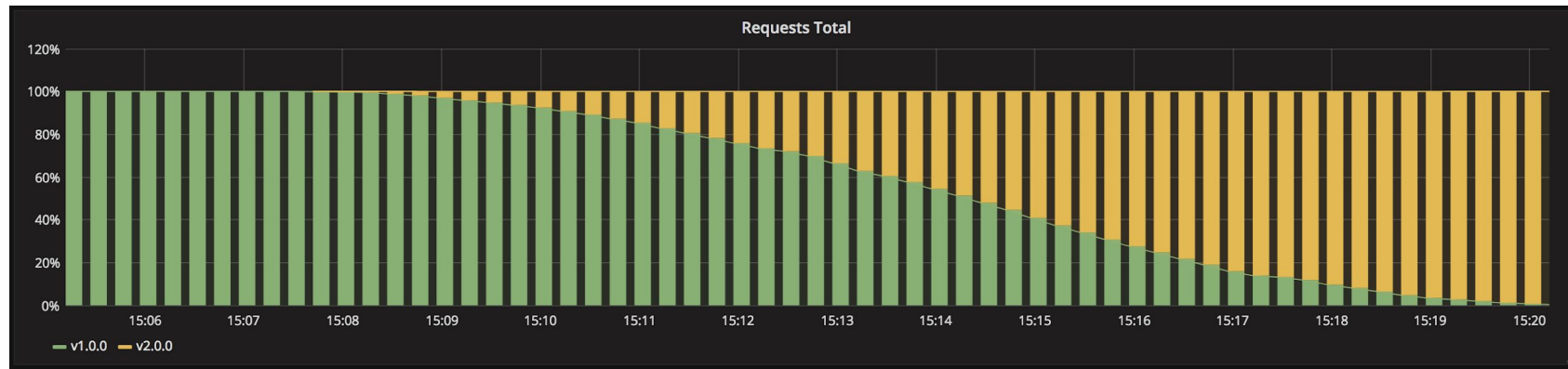
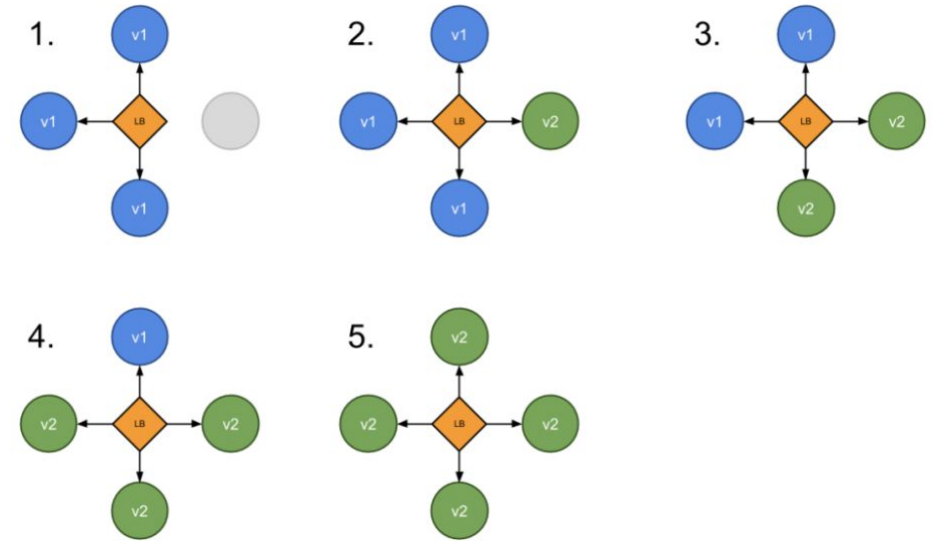


Rolling update

- Снимаем трафик с пода
- Обновляем версию
- Переключаем трафик в поду

Плюсы и минусы:

- + нет даунтайма
- - долгое время раскатки
- - API без обратной совместимости не раскатать

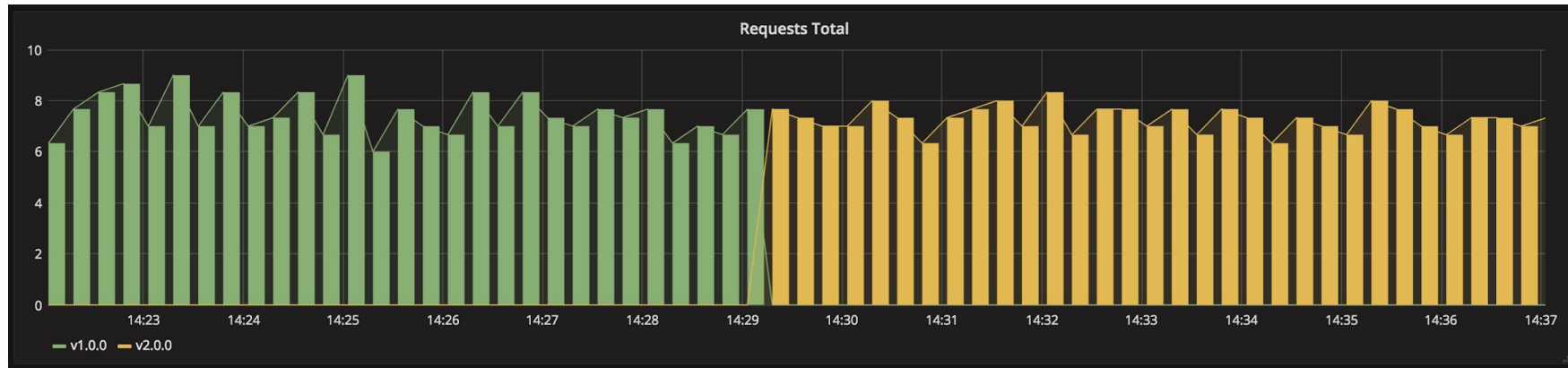
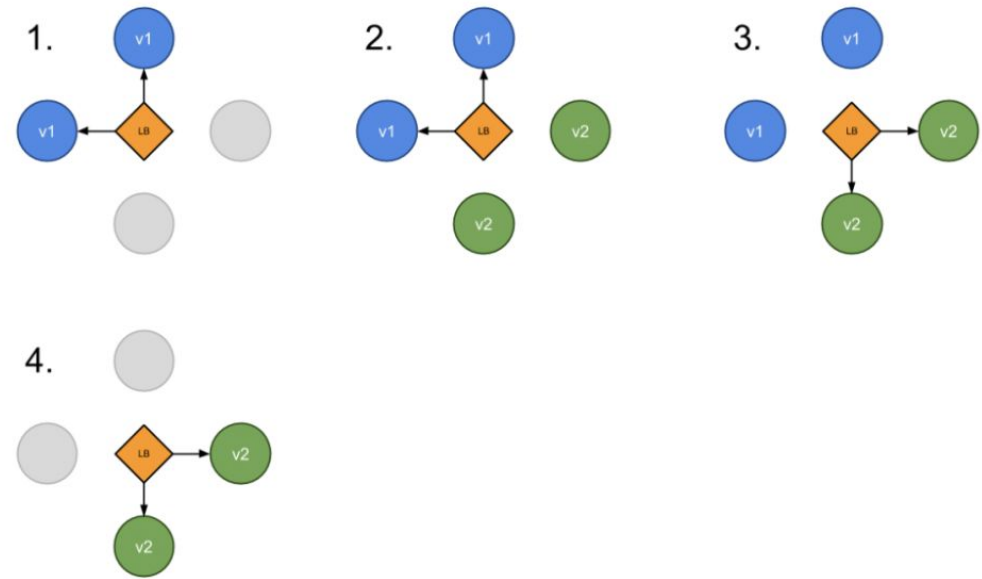


Blue/green deployment

- Поднимаем новую версию
- Проверяем ее
- Переключаем трафик в новую версию

Плюсы и минусы:

- + нет даунтайма
- + API без обратной совместимости можно раскатить
- - требуется в 2 раза больше ресурсов

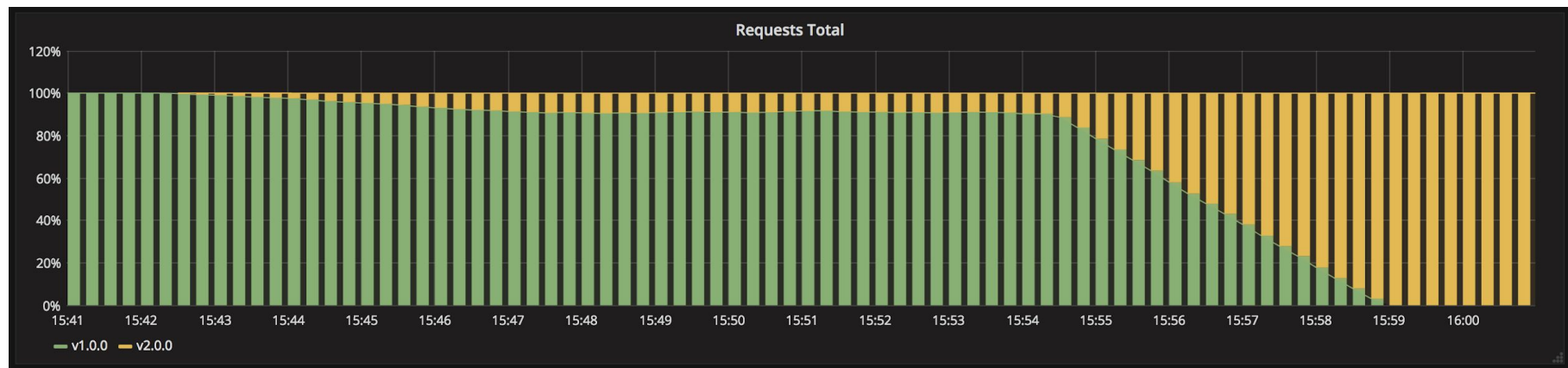
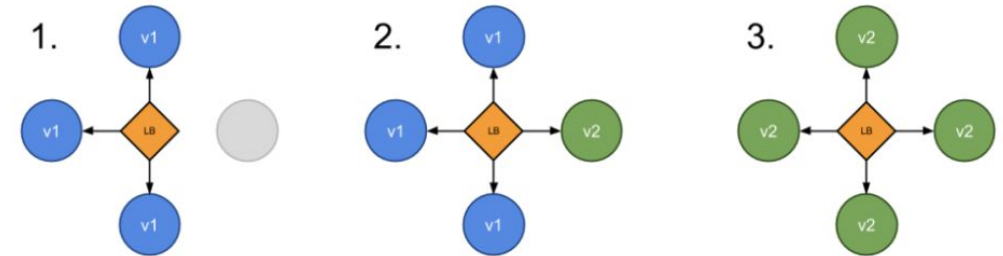


Канареечные деплои

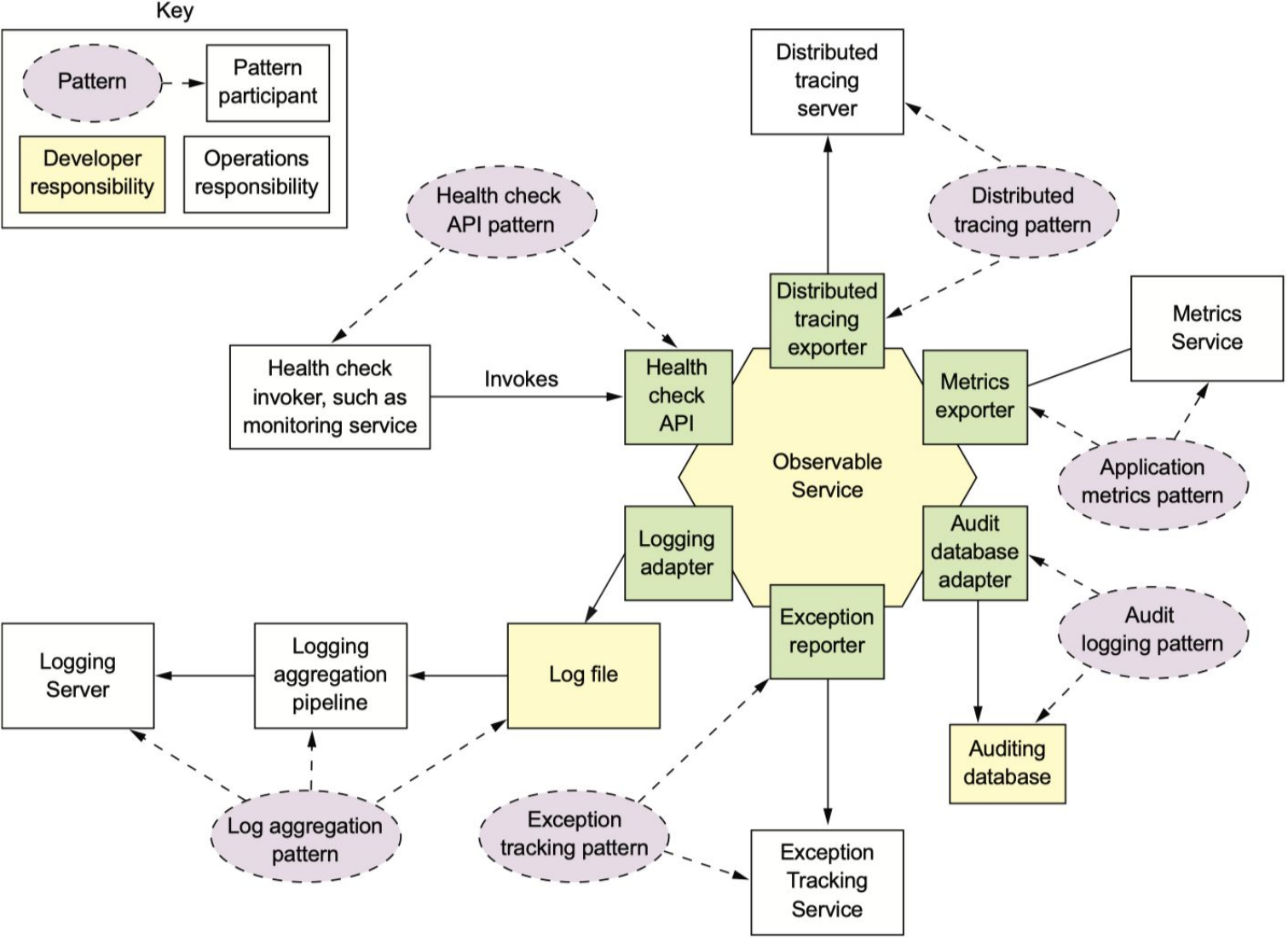
- Поднимаем новую версию одновременно со старой
- Переключаем на нее часть трафика
- Если все хорошо, переключаем остальной трафик

Плюсы и минусы:

- - API без обратной совместимости нельзя раскатить
- + нет даунтайма
- + быстро откатить
- + уменьшаем риски плохого релиза



Observability

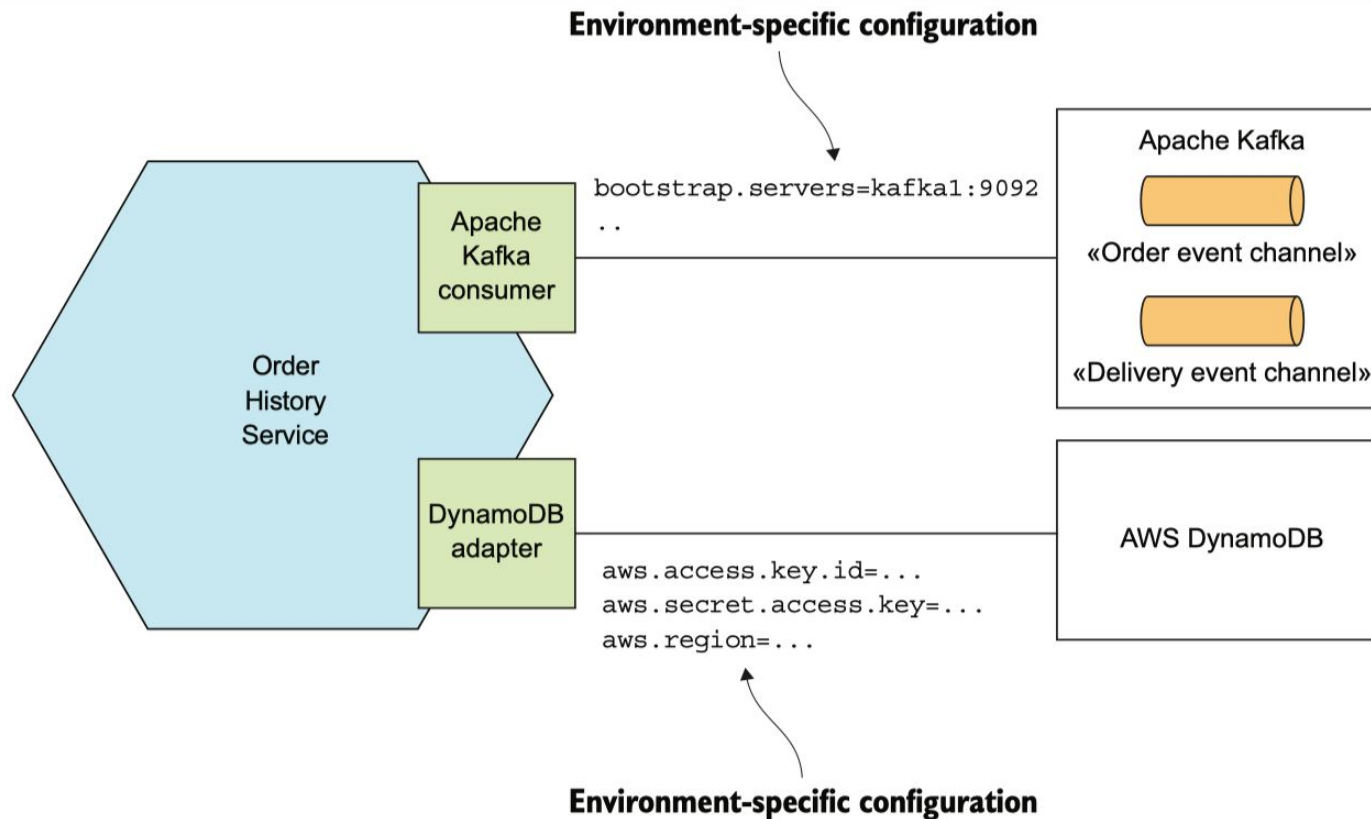


Конфигурирование приложений

Конфигурация приложения – это все, что может меняться, между развертываниями приложений.

Например,

- Идентификаторы подключения к ресурсам типа базы данных, кэш-памяти и другим сторонним службам

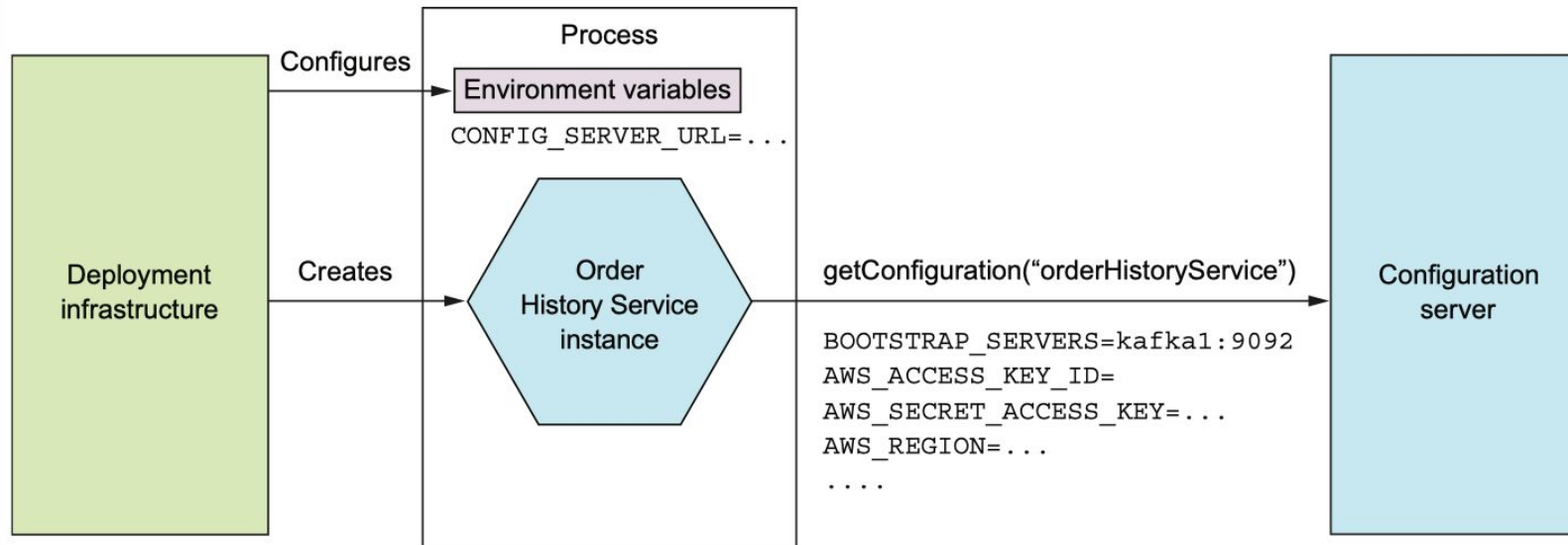


Pull модель конфигурирования

В момент старта приложение читает свой конфиг из внешнего сервиса.

Конфиг может храниться в:

- SQL
- NoSQL
- Git
- Vault
- И т.д



Конфигурирование приложений

Конфигурация должна быть отделена от кода

Кодовая база приложения может быть в любой момент открыта в свободный доступ без компрометации каких-либо частных данных

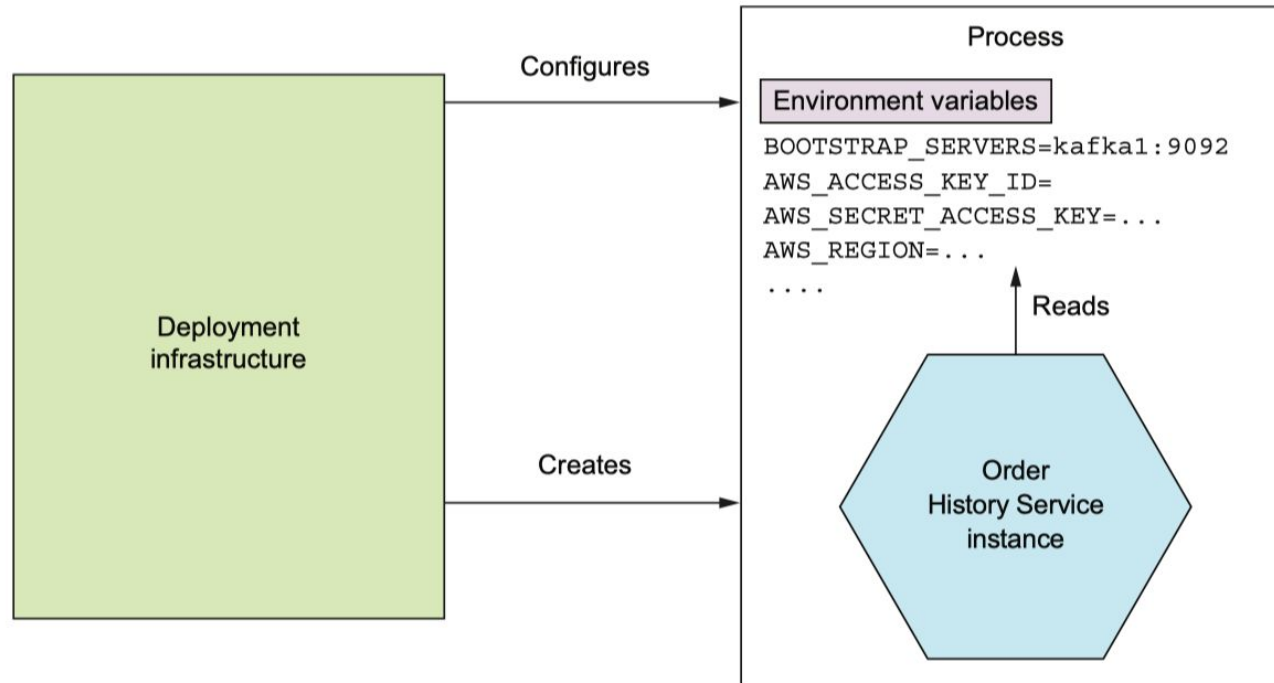
<https://12factor.net/ru/config>

Push модель конфигурирования

После деплоя оркестратор передает приложению конфиг

Конфиг может передаваться через

- Переменные окружения (ENV)
- Конфигурационный файл
- Параметры командной строки

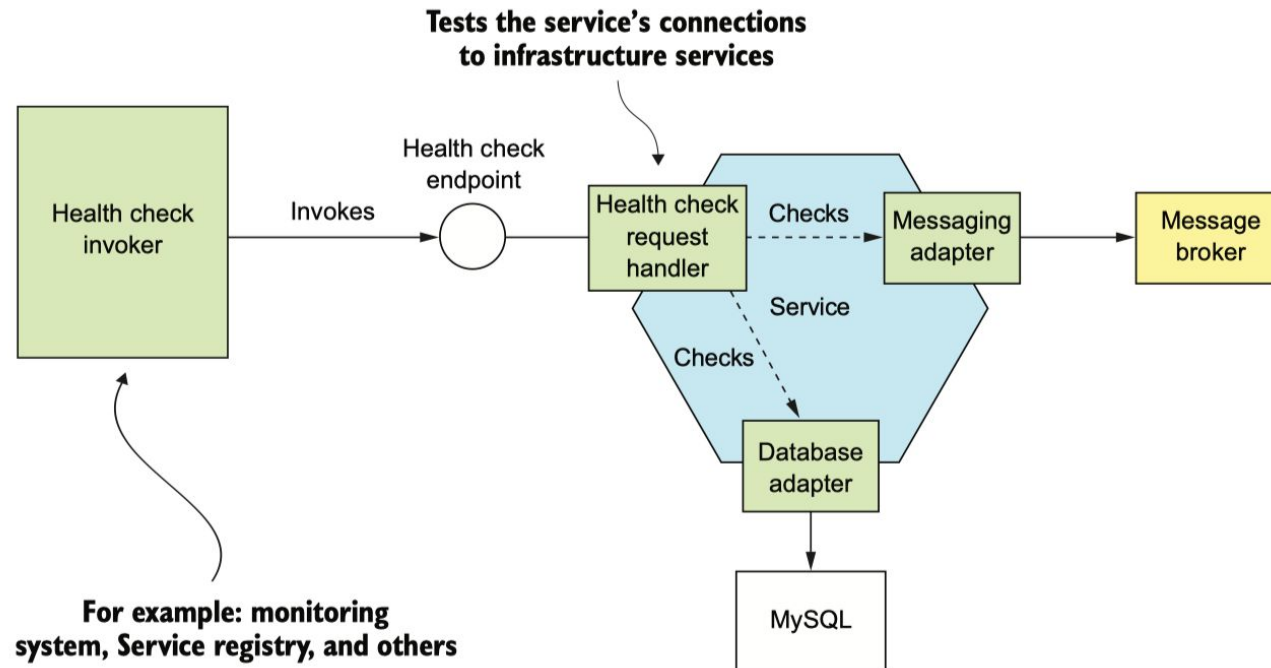


Health check

Чтобы проверять, умер под или нет заводится специальный метод (endpoint), по которому проверяется общая живость приложения.

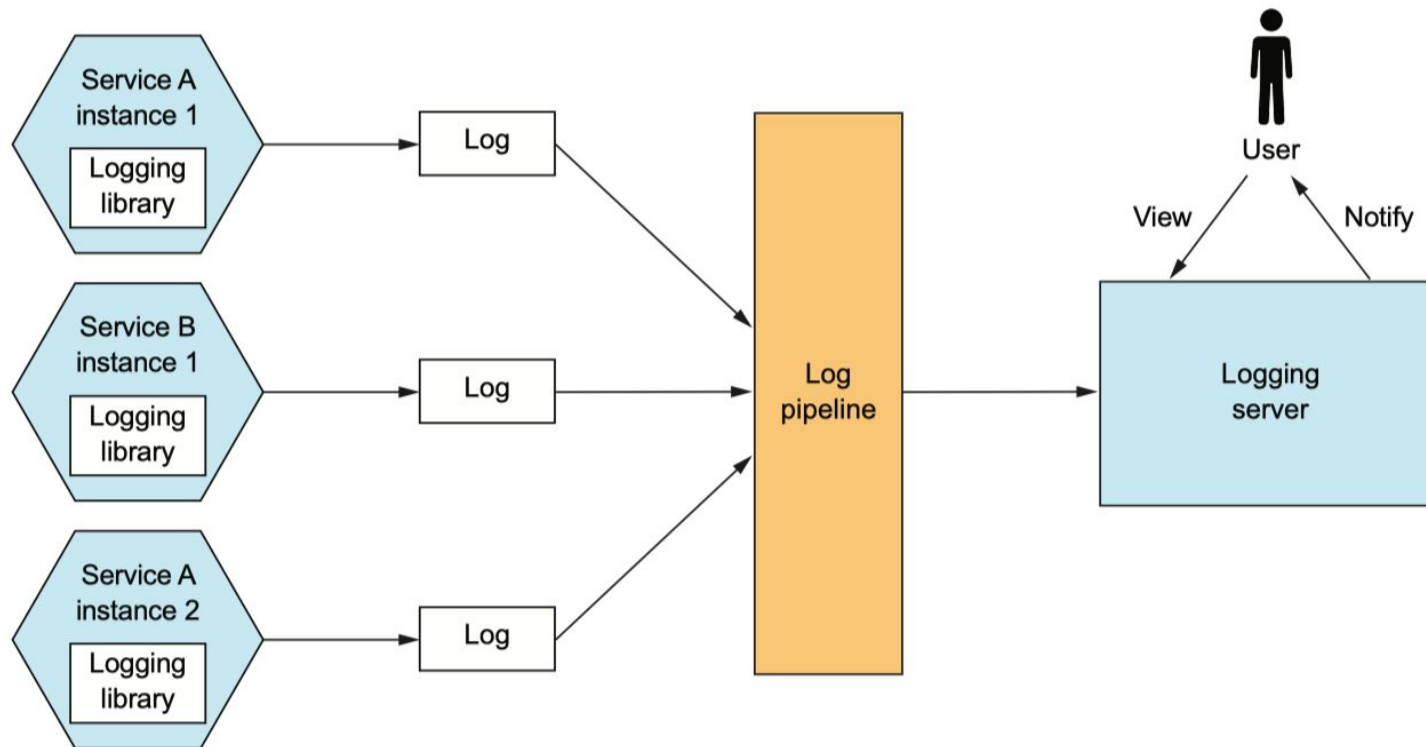
- Health probe – приложение живо
- Readiness probe – приложение готов принимать трафик

За жизнью под смотрит система мониторинга и алертинга, service registry, оркестратор, чтобы снимать трафик с больных под



Логирование

- Отправить логи из приложения
- Принять для доставки
- Доставить для анализа и хранения
- Проанализировать
- Хранить



ELK

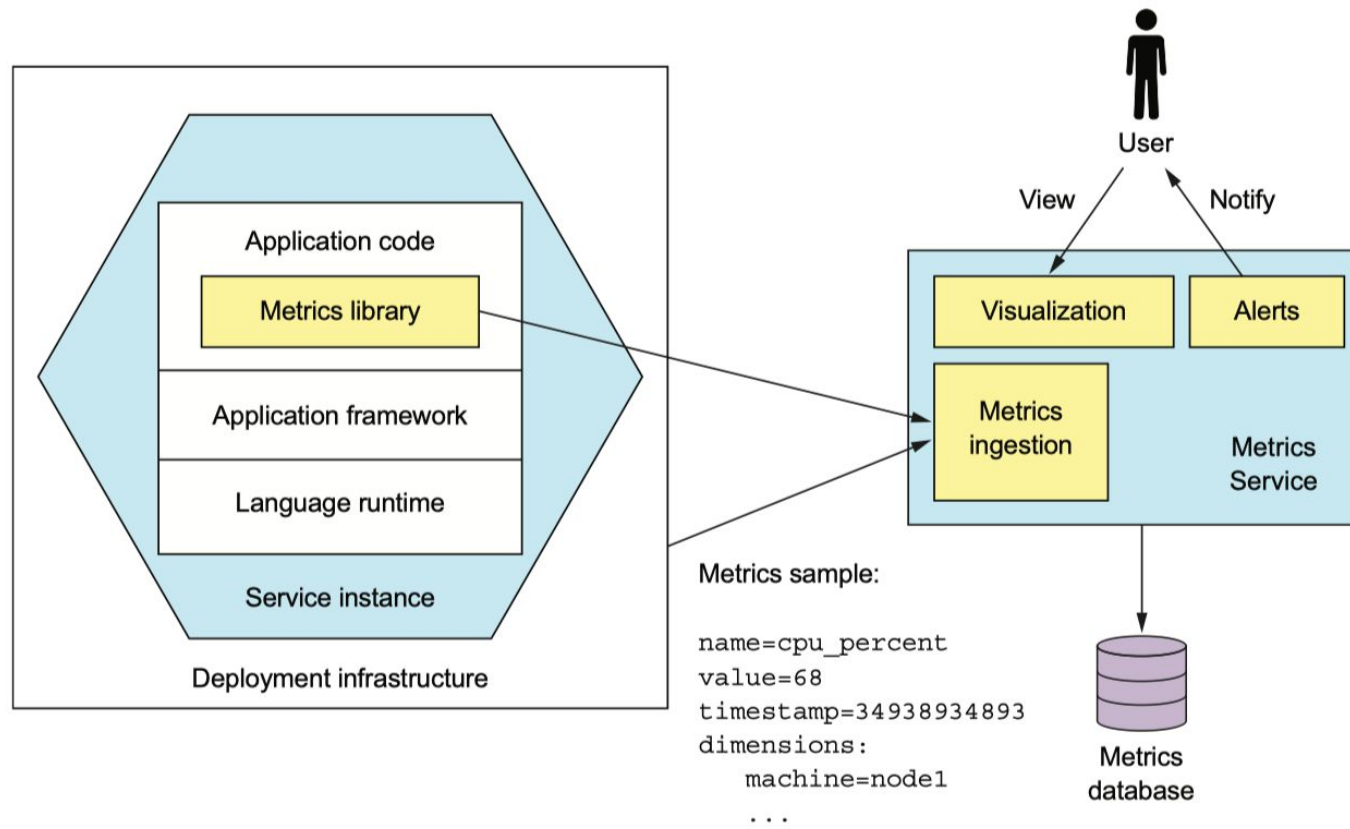
ELK

- Elastic Search
- Logstash
- Kibana

Мониторинг и алертинг

Как собирать метрики

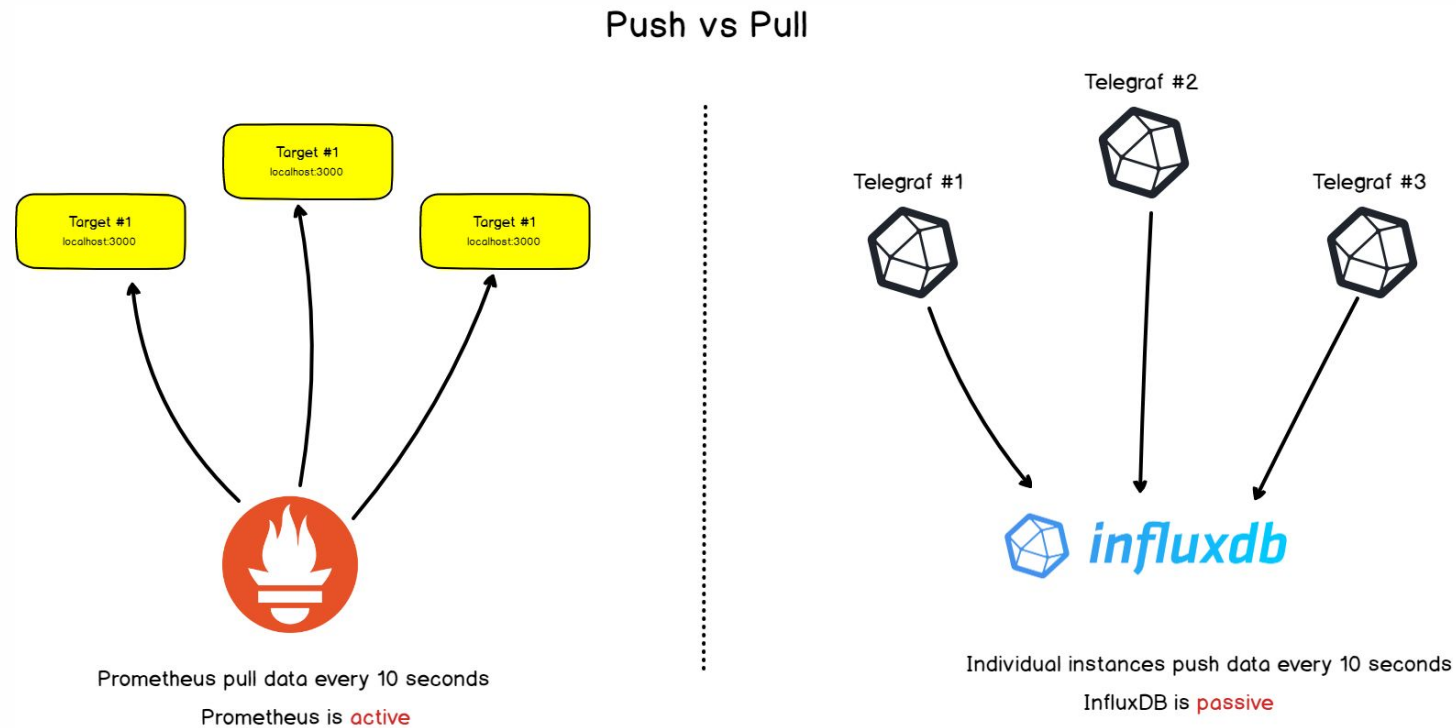
- спу/темогу
- Продуктовые метрики
- Технические



Pull vs Push модель сбора метрик

Push модель: приложение само ходит в сервис метрик и пушит туда все метрики

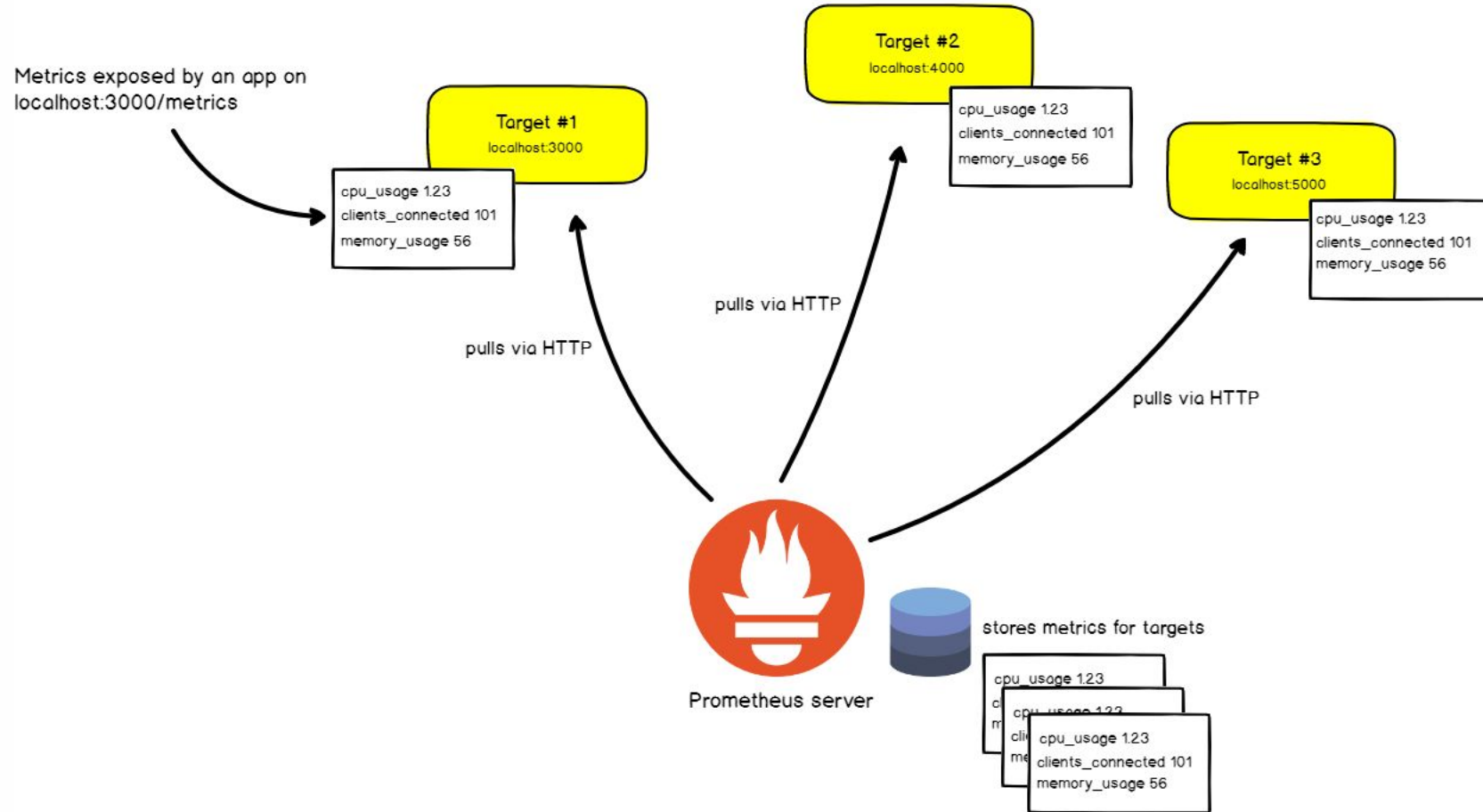
Pull модель: приложение выставляет урл (обычно /metrics), в котором все метрики, а сервис метрик забирает, и потом отображает.



Prometheus

Прометеус ходит по сервисам, забирает агрегированную статистику и складывает в базу.

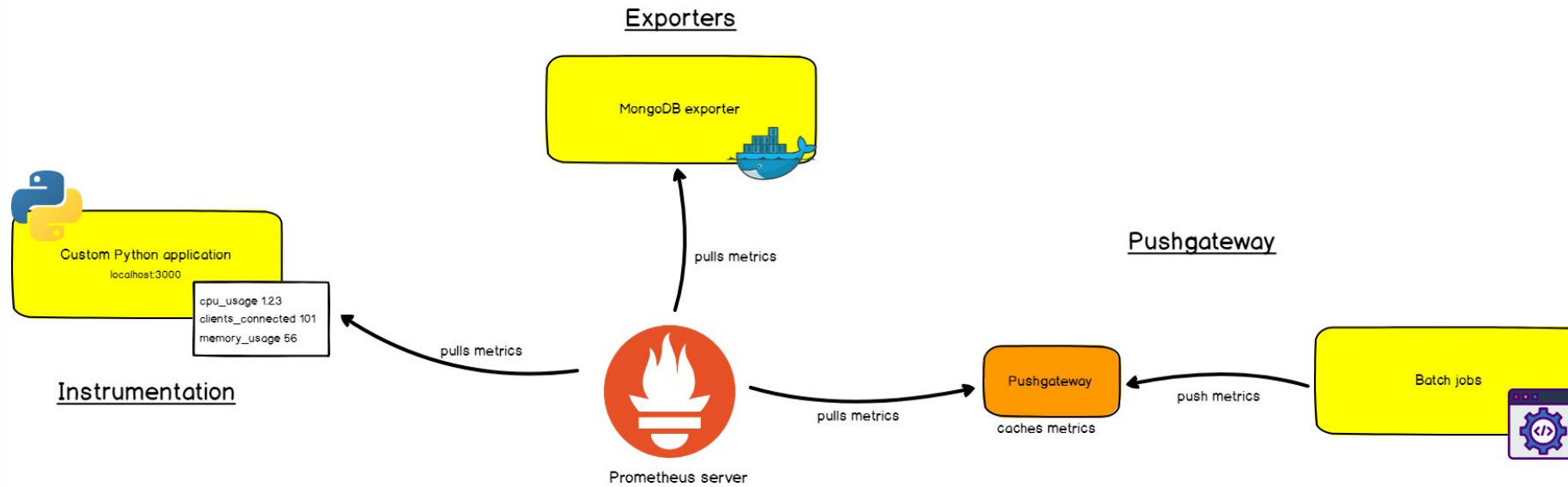
What does Prometheus do?



Prometheus

Прометеус может мониторить инфраструктуру с помощью экспортеров

Ways to gather metrics in Prometheus



Prometheus + grafana

Grafana – это интерфейс для визуализации графиков, метриков, в целом инструмент построения дашбордов

Distrubuted tracing

Распределенная транзакция – это путь прохождения запроса по разным сервисам.

- При трассировке, к каждому запросу добавляются метаданные о контексте этого запроса и эти метаданные сохраняются и передаются между компонентами, участвующими в обработке запроса
- В различных точках трассировки происходит сбор и запись событий вместе с дополнительной информацией (URL-запроса, идентификатор клиента, код запроса к БД)
- Информация о событиях сохраняется со всеми метаданными и контекстом и явным указанием причинно-следственных связей между событиями

Для чего используется tracing?

- Упрощенное взаимодействие между командами - при регрессах можно скинуть TracelD, связать систему трэкинга ошибок с трейсами
- Оценка критического пути выполнения запроса и влияния разных факторов на время выполнения (сетевые проблемы, медленные запросы к БД)
- Графы зависимостей - с кем взаимодействует мой сервис, кого затронут изменения в нем?

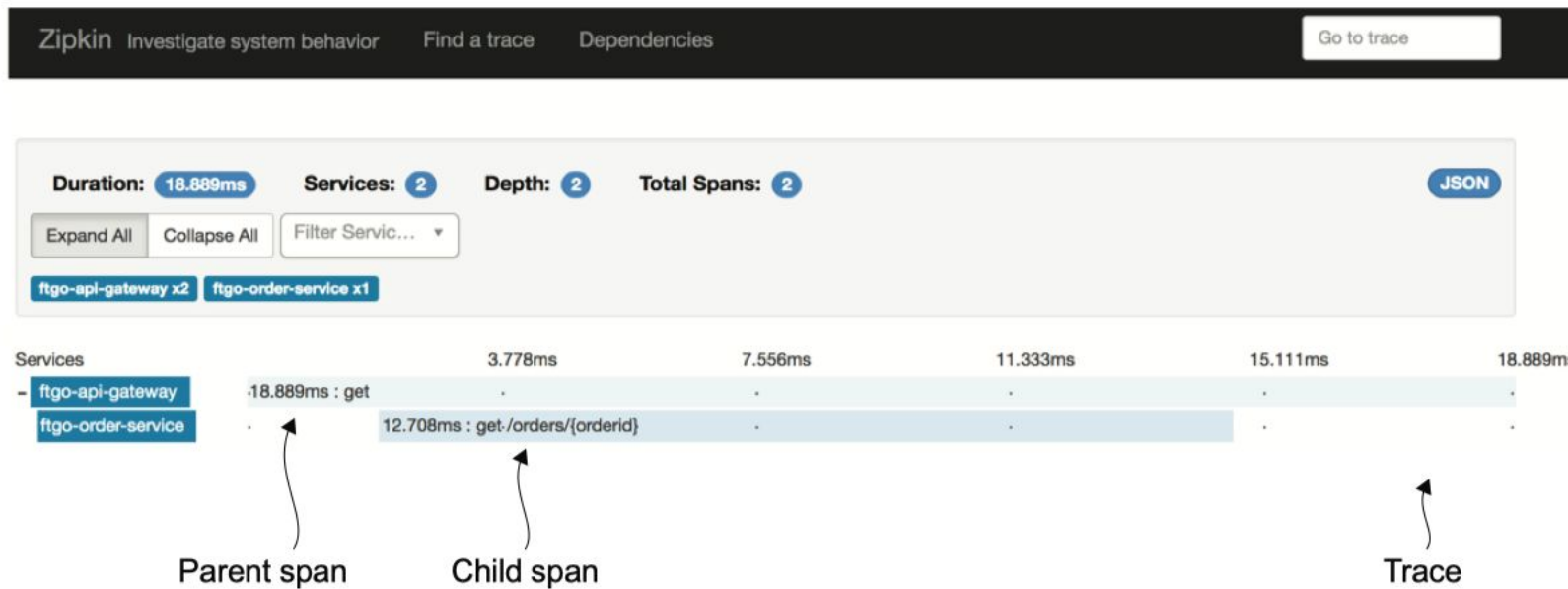
Основная терминология

Span - запись об одной логической операции по обработке запроса (тайминги и метаданные).

- Каждый спан обязательно содержит ссылку на Trace-ID
- Каждый спан содержит свой уникальный идентификатор Span-ID

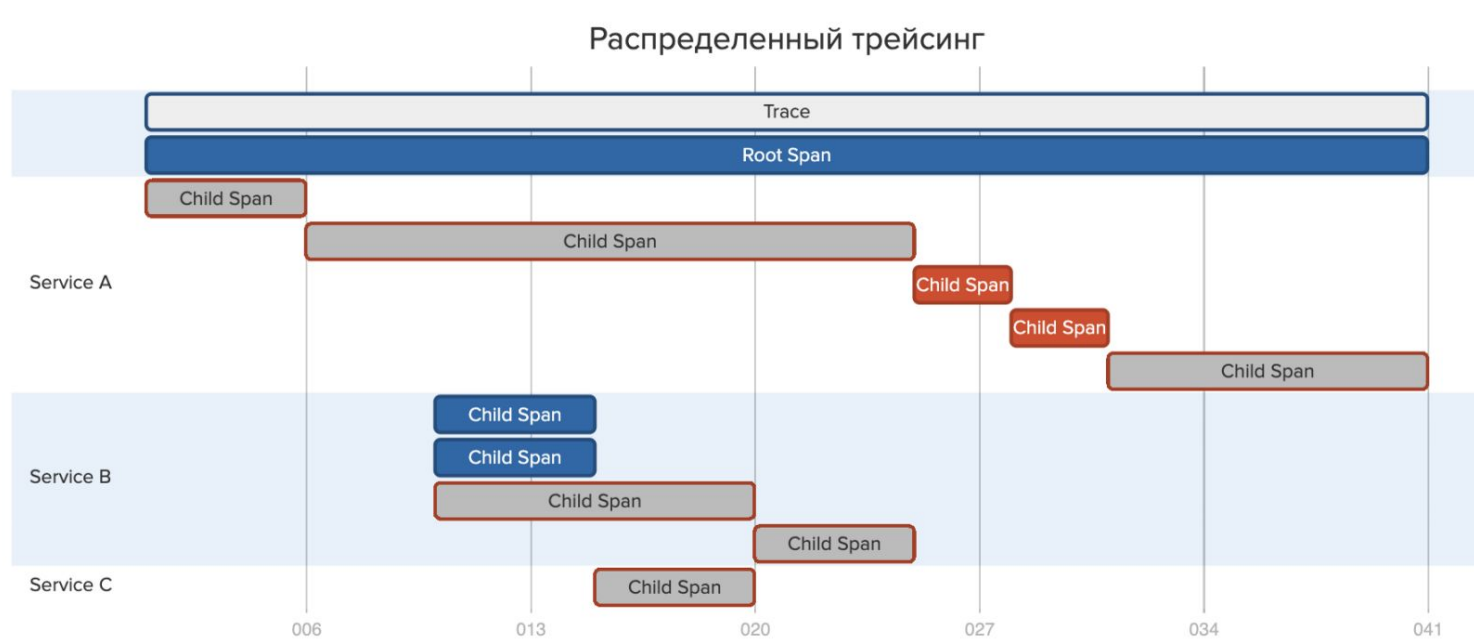
Trace - коллекция связанных записей (Spans), описывающая обработку одного запроса (end-to-end)

- каждый трейс имеет свой уникальный идентификатор - Trace ID



Основная терминология

- **Root Span** - это спан, у которого нет ссылки на родительский спан (только Trace ID), он показывает общую длительность выполнения запроса



Какие есть проблемы

- Не видны проблемы общей инфраструктуры (состояние очередей, IOPS и т. п.), "серые ошибки" в облаках
- В трассировках нет "низкоуровневых" данных - состояние ОС, ядра и т.п., то что добывается strace, ss и прочим
- Для протоколов, где нет метаданных (Kafka), надо писать свои обвязки и прокидывать
- Надо выбирать нагрузку и частоту сэмплирования

Инструментарий distributed-tracing

2 основных протокола:

- OpenTracing (X-OT-* заголовки)
- B3 (Zipkin) (X-B3-* заголовки)

Клиентские библиотеки и сервера для хранения и визуализации трассировок

- OpenTelemetry
- Jaeger, OpenZipkin, LightStep

APM

- Elastic APM

Сами трейсы и индексы хранятся обычно в Elasticsearch

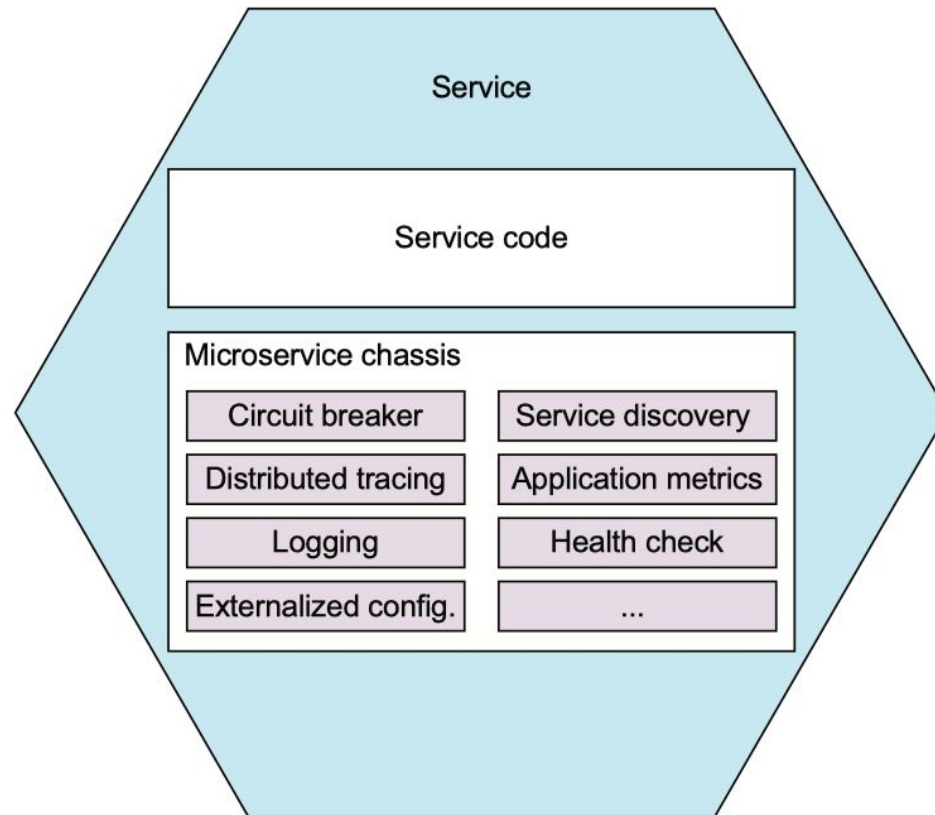
Elastic APM

Elastic APM – средства для анализа производительности приложений с tracing-ом и метриками

Microservice chassis

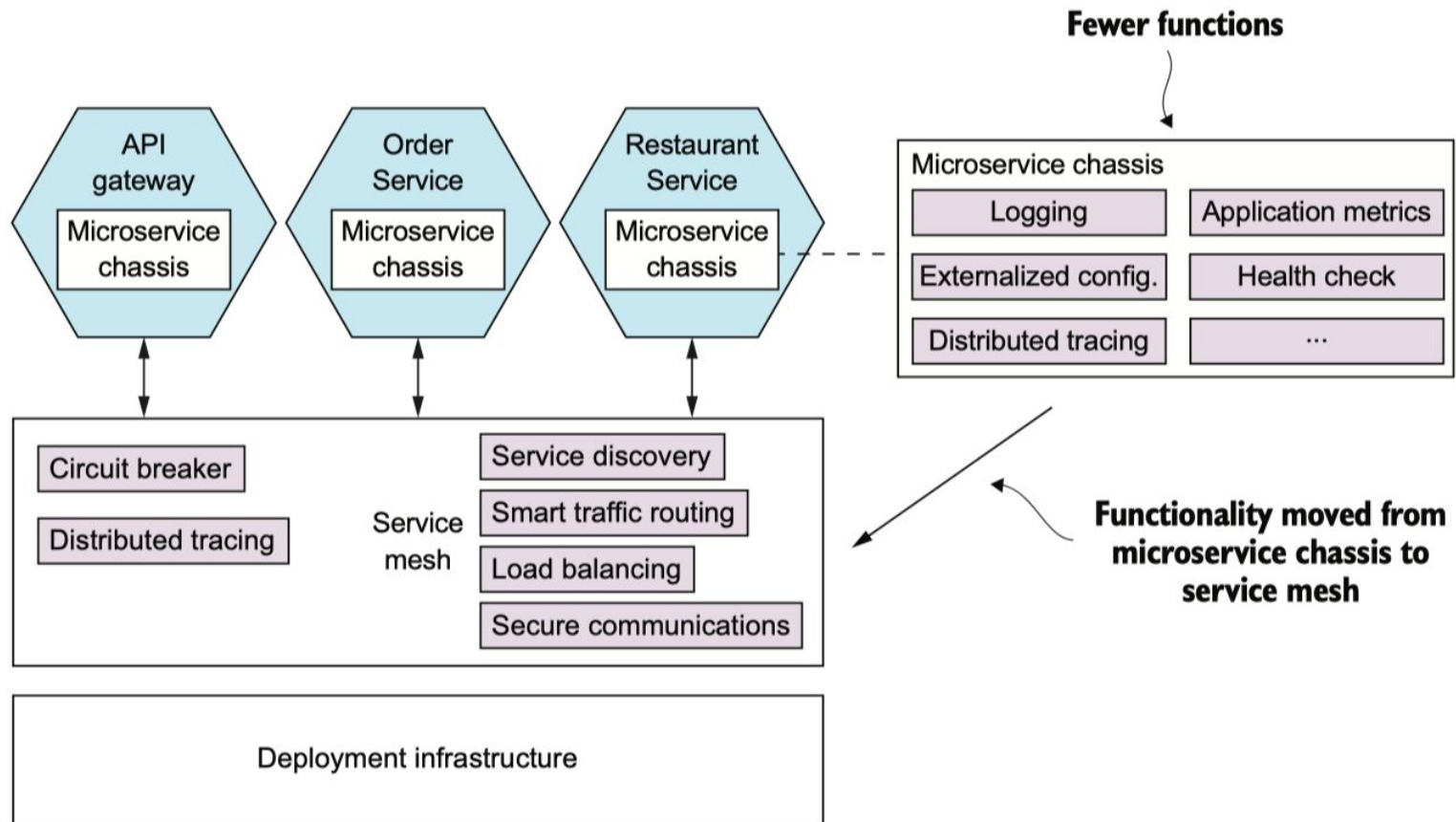
Microservice chassis – это паттерн, при котором есть фреймворк, помогающий встраивать сервисы в микросервисную архитектуру

Примеры: SpringBoot/SpringCloud, Go-kit



Service mesh

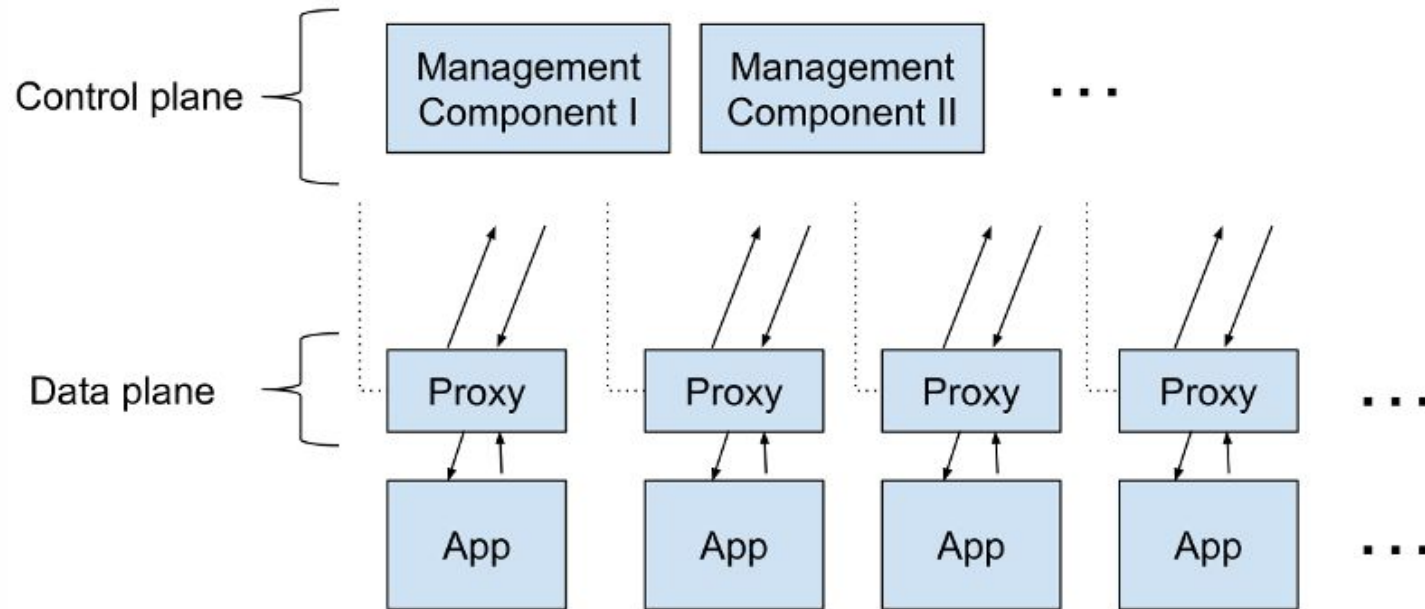
Почему бы не вынести часть логики из кода приложения в отдельный «слой» взаимодействия сервисов?



Service mesh

Давайте с каждым подом поставим side-прокси сервис, в котором будет вся логика по работе с другими сервисами:

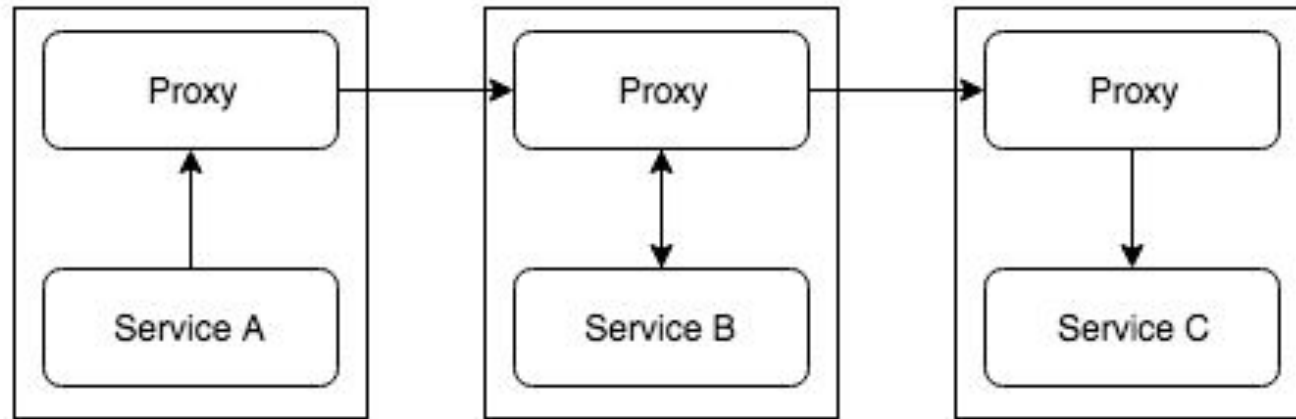
- Проверка прав доступа
- Отправка метрик и телеметрии
- Distributed tracing
- Circuit breaker
- И т.д.



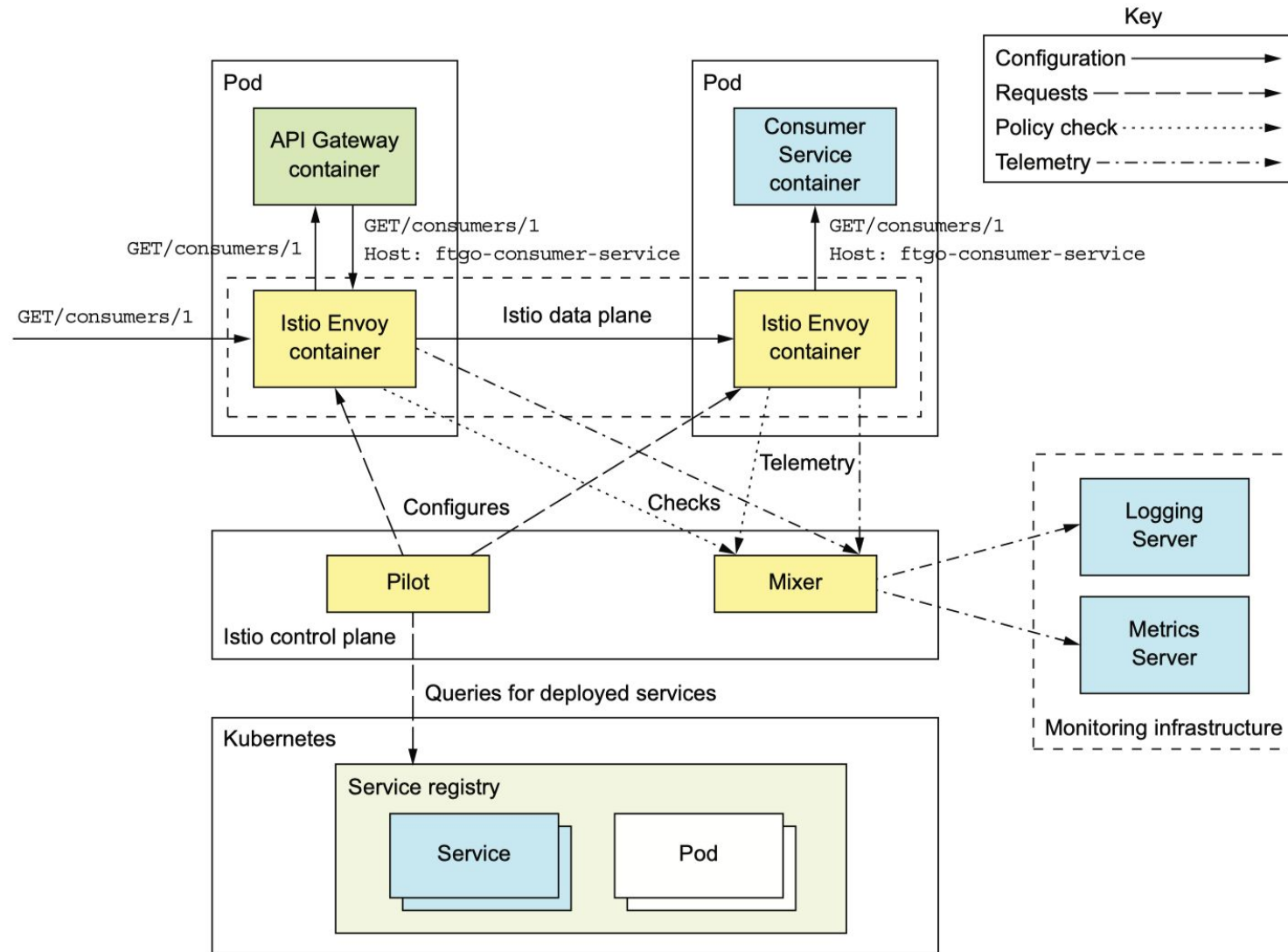
Service mesh

Неважно, что делают сервисы, но трафик между ними является идеальной точкой для добавления новой функциональности.

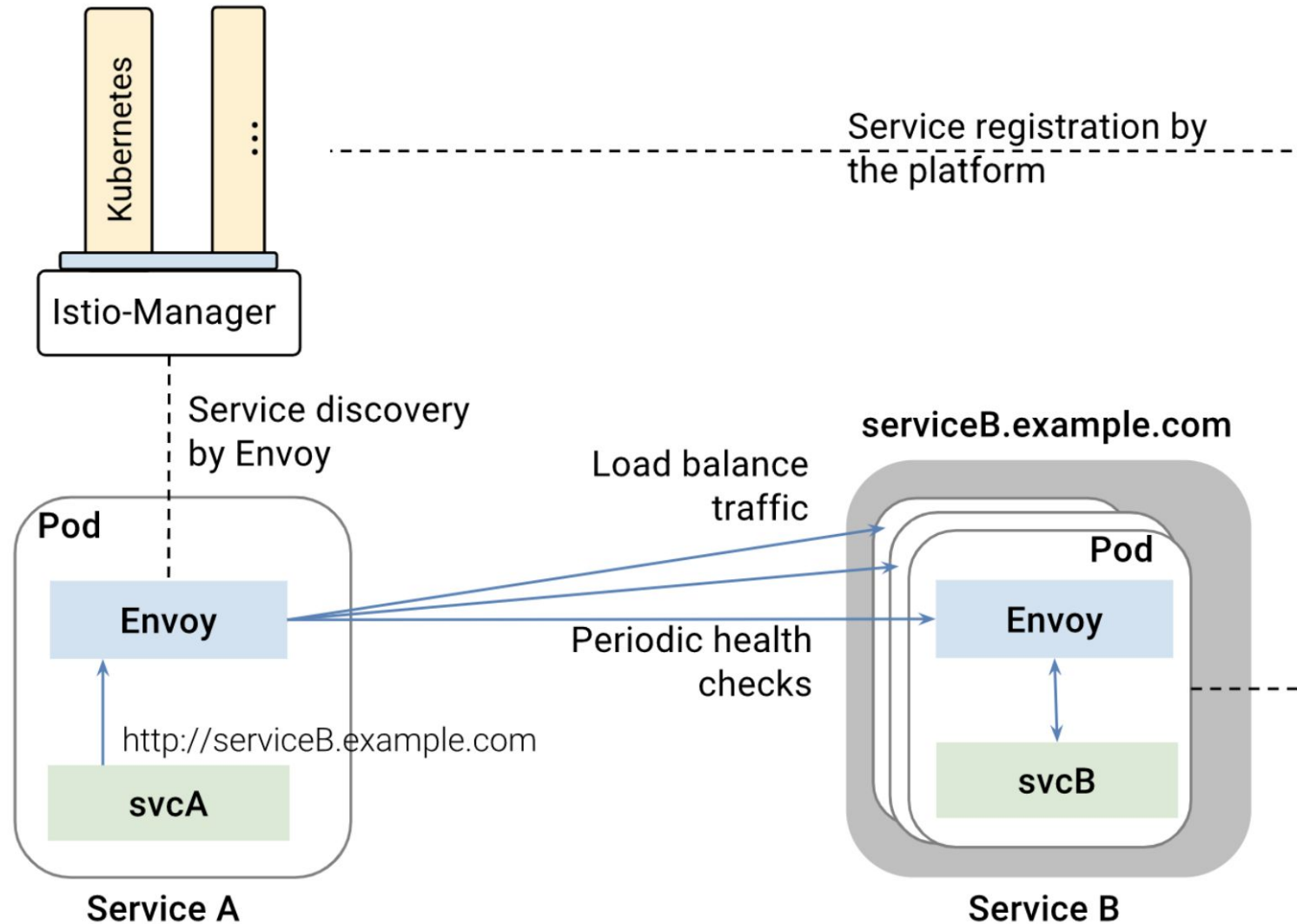
Service mesh выглядит так: вы разворачиваете кучу прокси, которые «что-то делают» с внутренним, межсервисным трафиком, и используете control plane для мониторинга и управления ими.



Service mesh на примере istio



Балансировка запросов



Граф сервисов

Namespace: tutorial

Display

- Node Names
- Service Nodes
- Traffic Animation
- Unused Nodes

Badges:

- Circuit Breakers
- Virtual Services
- Missing Sidecars
- Security

Graph Type: Versioned app

Fetching: Last min, Every 15 sec

Namespace: tutorial

- 5 apps
- 6 links

HTTP Traffic (requests per second):


Total	%Success	%Error
1.98	100.00	0.00

HTTP - Total Request Traffic min / max:
RPS: 0.00 / 2.20, %Error 0.00 / 0.00

TCP - Total Traffic - min / max:
ⓘ Not enough traffic to generate chart.

Waiting for kiali-istio-system.192.168.99.103.nip.io...

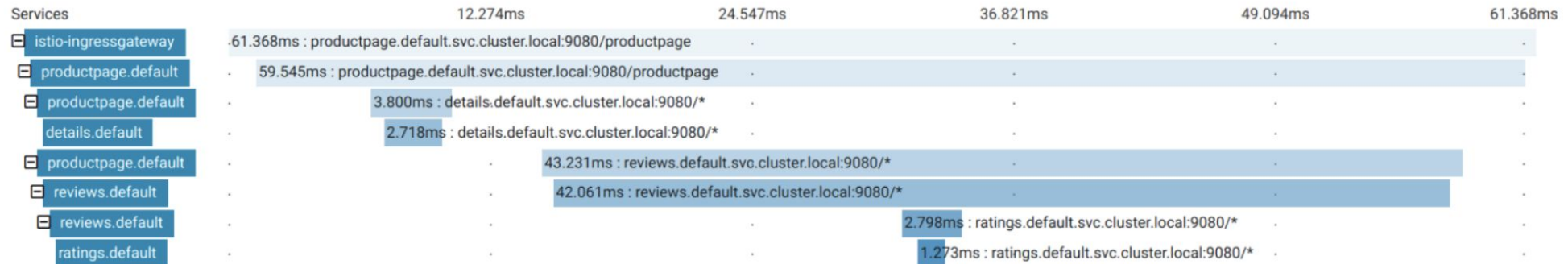
Распределенная трассировка

 Investigate system behavior Find a trace View Saved Trace Dependencies [Try Lens UI](#)

Duration: **61.368ms** Services: **5** Depth: **6** Total Spans: **8** [JSON](#) 

[Expand All](#) [Collapse All](#)

[details.default x1](#) [istio-ingressgateway x1](#) [productpage.default x3](#) [ratings.default x1](#) [reviews.default x2](#)



Distributed Trace for a single request

Service mesh на примере istio



Dave Strebel

@dave_strebel

Читать

“Finally got Istio into production”



17:21 - 17 сент. 2019 г. Apple Valley, MN



Adam Phillabaum

@adamb0mb

Читать

Tech tip for the day: Don't use #istio in production. It sounds awesome, but moving away from it has been one of the biggest stress-relievers I've seen in a long time.



13:23 - 26 сент. 2019 г.

Service mesh плюсы и минусы

Минусы:

- Увеличение latency
- Дополнительные ресурсы (mem, cpu) на работу прокси
- Сложность поддержки

Плюсы:

- Возможность централизованно и без внесения правок в код приложений и независимо от их стека добавлять функциональность.

Опрос

<https://otus.ru/polls/6408/>

**Спасибо
за внимание!**

