PaaS в Avito

Лукьянченко Александр
Team Lead, Architecture

Москва, 2020

Avito

План

План

1. Платформа – что и зачем?

План

1. Платформа – что и зачем?
2. Концепция PaaS

План

1. Платформа – что и зачем?
2. Концепция PaaS
3. Проблемы и технические решения

Зачем платформа?

Зачем платформа?

1. Снижение оверхэда на интеграцию с инфраструктурой

Зачем платформа?

1. Снижение оверхэда на интеграцию с инфраструктурой
2. Переиспользование лучших практик

Зачем платформа?

1. Снижение оверхэда на интеграцию с инфраструктурой
2. Переиспользование лучших практик
3. Возможность контроля и централизованного внедрения технологий и подходов

Особенности

Особенности

1. Своё железо

Особенности

1. Своё железо
2. Management и все решения on-premise

Особенности

1. Своё железо
2. Management и все решения on-premise
3. Своё облако

Тогда и сейчас

Dev instruments

- libraries
- minikube
- dotenv
- reloader
- brief

Runtime instruments

- elastic
- prometheus
- Jaeger
- sentry
- harbor

Cloud, Provisioning, Runtime

- jibe
- helm
- redis operator
- data-bus
- kafka
- navigator
- ingress controller
- netramesh
- vault
- soc
- fluentd
- kubernetes
- calico
- docker
- debian
- puppet

Инфраструктура

Инфраструктура

- Kubernetes as is

Инфраструктура

- Kubernetes as is
- RBAC и выдача прав в продуктовые команды

Инфраструктура

- Kubernetes as is
- RBAC и выдача прав в продуктовые команды
- Шаблонные pipeline'ы для deploy в CI

Инфраструктура

- Kubernetes as is
- RBAC и выдача прав в продуктовые команды
- Шаблонные pipeline'ы для deploy в CI
- Низкоуровневые интерфейсы: helm, kubectl

PaaS

PaaS CLI

PaaS Dashboard

Dev instruments

- libraries
- minikube
- dotenv
- reloader
- brief

Runtime instruments

- elastic
- prometheus
- Jaeger
- sentry
- harbor

Cloud, Provisioning, Runtime

- jibe
- helm
- redis operator
- data-bus
- kafka
- navigator
- ingress controller
- netramesh
- vault
- soc
- fluentd
- kubernetes
- calico
- docker
- debian
- puppet

Концепция PaaS

Концепция PaaS

- Максимальная автоматизация

Концепция PaaS

- Максимальная автоматизация
- Построение продукта, решающего реальные потребности разработчиков

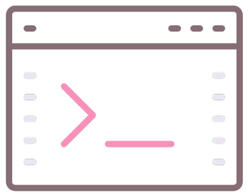
Концепция PaaS

- Максимальная автоматизация
- Построение продукта, решающего реальные потребности разработчиков
- Низкий порог входа, концентрация на скорости доставки фичей

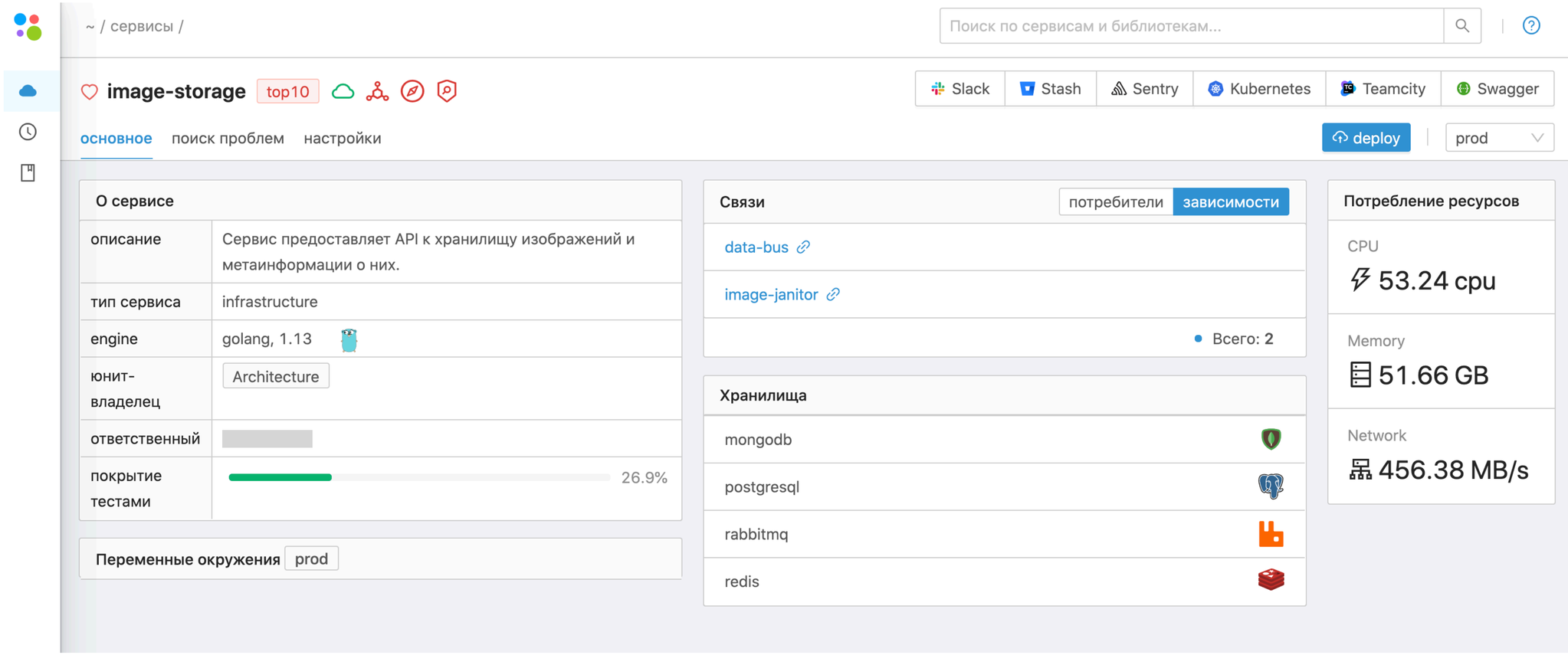
Концепция PaaS

- Максимальная автоматизация
- Построение продукта, решающего реальные потребности разработчиков
- Низкий порог входа, концентрация на скорости доставки фичей
- Нулевой оверхэд интеграции с инфраструктурой

Со стороны пользователя



avito CLI



PaaS Dashboard

Что вообще может болеть?

Avito

Как положить секрет в vault?

Как положить секрет в vault?

Почему у меня upgrade failed waiting for a condition?

Потратил неделю на настройку связки сервис, pgbouncer, vault

Как положить секрет в vault?

Почему у меня upgrade failed waiting for a condition?

Потратил неделю на настройку связки сервис, pgbouncer, vault

Как положить секрет в vault?

Почему у меня upgrade failed waiting for a condition?

Как прокинуть хост наружу?

Потратил неделю на настройку связки сервис, pgbouncer, vault

Как положить секрет в vault?

Какие ресурсы мне поставить в deployment?

Почему у меня upgrade failed waiting for a condition?

Как прокинуть хост наружу?

Потратил неделю на настройку связки сервис, pgbouncer, vault

Как положить секрет в vault?

Какие ресурсы мне поставить в deployment?

Почему у меня upgrade failed waiting for a condition?

Как прокинуть хост наружу?

Что, опять переезжаем в новый кластер?

Почему у меня нет метрик в staging?

Потратил неделю на настройку связки сервис, pgbouncer, vault

Как положить секрет в vault?

Какие ресурсы мне поставить в deployment?

Почему у меня upgrade failed waiting for a condition?

Как прокинуть хост наружу?

Что, опять переезжаем в новый кластер?

Почему у меня нет метрик в staging?

Потратил неделю на настройку связки сервис, pgbouncer, vault

Как положить секрет в vault?

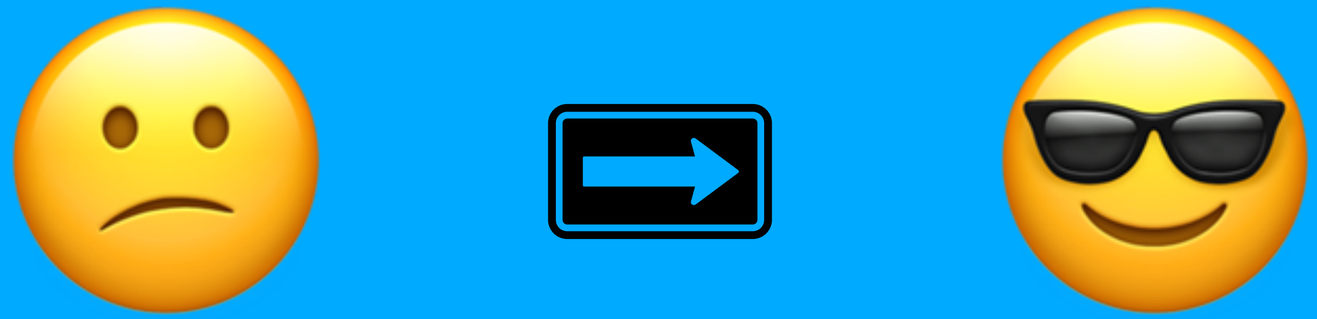
Какие ресурсы мне поставить в deployment?

Почему у меня upgrade failed waiting for a condition?

Сервис упал, куда смотреть?

Как прокинуть хост наружу?

Что, опять переезжаем в новый кластер?



Москва, 2020

Avito

Жизненный цикл

Жизненный цикл

1. Создание сервисов

Жизненный цикл

1. Создание сервисов
2. Разработка сервисов

Жизненный цикл

1. Создание сервисов
2. Разработка сервисов
3. Тестирование сервисов

Жизненный цикл

1. Создание сервисов
2. Разработка сервисов
3. Тестирование сервисов
4. Доставка сервисов

Жизненный цикл

1. Создание сервисов
2. Разработка сервисов
3. Тестирование сервисов
4. Доставка сервисов
5. Эксплуатация сервисов

1. Создание сервисов
2. Разработка сервисов
3. Тестирование сервисов
4. Доставка сервисов
5. Эксплуатация сервисов

Avito

Ручное создание 😕

Ручное создание 😕

1. Создание репозитория

Ручное создание 😕

1. Создание репозитория
2. Регистрация в системе учета

Ручное создание 😕

1. Создание репозитория
2. Регистрация в системе учета
3. Создание основных ресурсов: grafana dashboard, CI pipelines, sentry проект, пространства для секретов, etc

Ручное создание 😕

Ручное создание 😕

1. ~ 20 минут каждый раз

Ручное создание 😕

1. ~ 20 минут каждый раз
2. Возможность забыть создать какой-то ресурс или сделать это неверно

Создание сервиса 😎

```
$ avito service create
```


Создание сервиса 😎

```
$ avito service create
Выберите шаблон:
(1) Go
(2) PHP
(3) Python (Flask)
(4) Static (Frontend)
…
Choice: 1
```

Создание сервиса 😎

```
$ avito service create
…
Введите имя сервиса: my-awesome
Введите краткое описание сервиса: My Awesome service for quickguide
```


Создание сервиса 😎

```
$ avito service create
…
Создаю сервис my-awesome из шаблона go_service для пользователя awesome-user…
```


Создание сервиса 😎

```
$ avito service create
...

Сервис успешно создан!

Сервис склонирован в директорию /Users/myusername/projects/go/src/
go.avito.ru/av/service-my-awesome

Репозиторий в Stash: https://stash/projects/service-my-awesome
Проект в TeamCity: https://teamcity/project.html?projectId=awesome-
service
Проект в Sentry: https://sentry/avito/service-my-awesome/
Дашборд в Grafana: https://monitoring/grafana/d/1HiBumrik/service-my-
awesome

Перед началом разработки прочитайте README.md!
```

Создание сервиса 😎

Создание сервиса 😎

1. Репозиторий с уже работающим сервисом

Создание сервиса 😎

1. Репозиторий с уже работающим сервисом
2. Созданные и готовые к работе pipeline'ы выкатки

Создание сервиса 😎

1. Репозиторий с уже работающим сервисом
2. Созданные и готовые к работе pipeline'ы выкатки
3. Observability ресурсы: grafana dashboard, kibana, sentry

Создание сервиса 😎

1. Репозиторий с уже работающим сервисом
2. Созданные и готовые к работе pipeline'ы выкатки
3. Observability ресурсы: grafana dashboard, kibana, sentry
4. Политики и настройки доступа к сервису извне

1. Создание сервисов
2. Разработка сервисов
3. Тестирование сервисов
4. Доставка сервисов
5. Эксплуатация сервисов

Avito

Структура сервисов 😕

Структура сервисов 😕

1. Без унификации все сервисы разные

Структура сервисов 😕

1. Без унификации все сервисы разные
2. Метрики, логи – всё пишется по разному

Структура сервисов 😕

1. Без унификации все сервисы разные
2. Метрики, логи – всё пишется по разному
3. Интерфейсы не похожи

Структура сервисов 😕

1. Без унификации все сервисы разные
2. Метрики, логи – всё пишется по разному
3. Интерфейсы не похожи
4. Долгий вход

Унификация структуры сервисов 😎

Унификация структуры сервисов 😎

1. Единая структура для каждой технологии

Унификация структуры сервисов 😎

1. Единая структура для каждой технологии
2. Автоматическая отправка метрик и логов в одном формате

Унификация структуры сервисов 😎

1. Единая структура для каждой технологии
2. Автоматическая отправка метрик и логов в одном формате
3. Health check endpoint

Унификация структуры сервисов 😎

1. Единая структура для каждой технологии
2. Автоматическая отправка метрик и логов в одном формате
3. Health check endpoint
4. Поддержка RPC протокола взаимодействия между сервисами из коробки

Конфигурация 😕

Конфигурация 😕

1. plain kubernetes манифесты

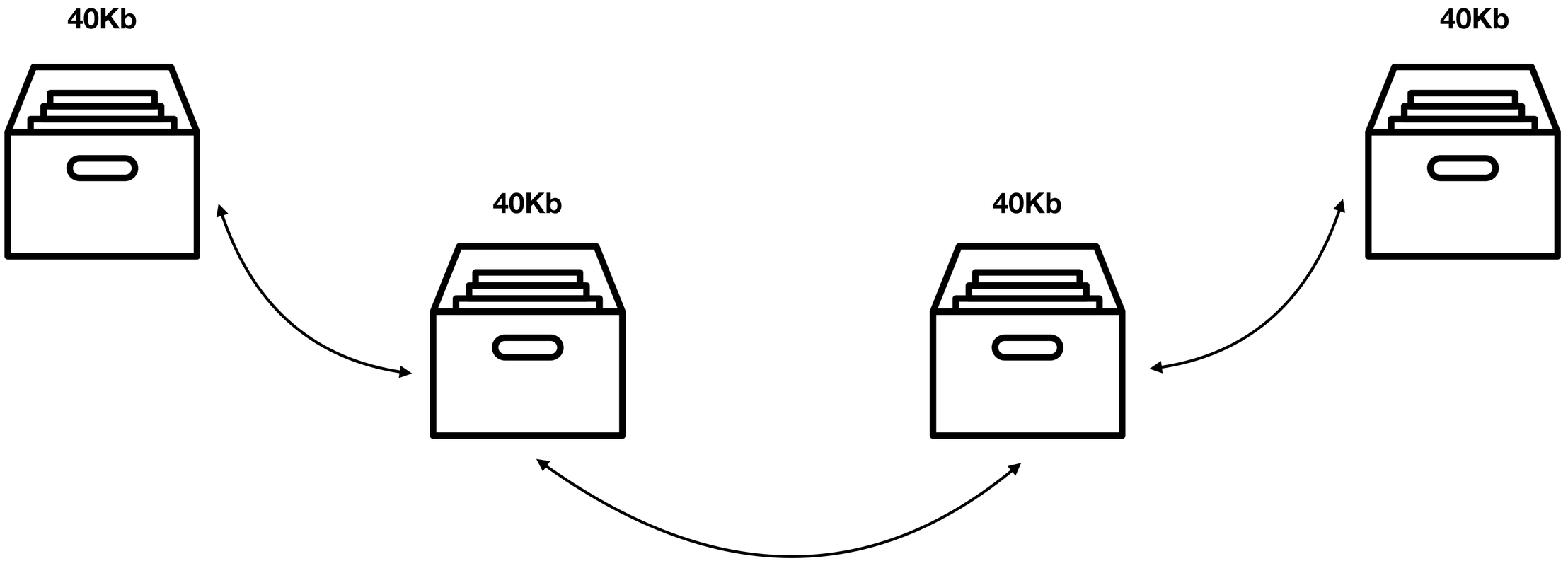
Конфигурация 😕

1. plain kubernetes манифесты
2. helm чарты

Конфигурация 😕

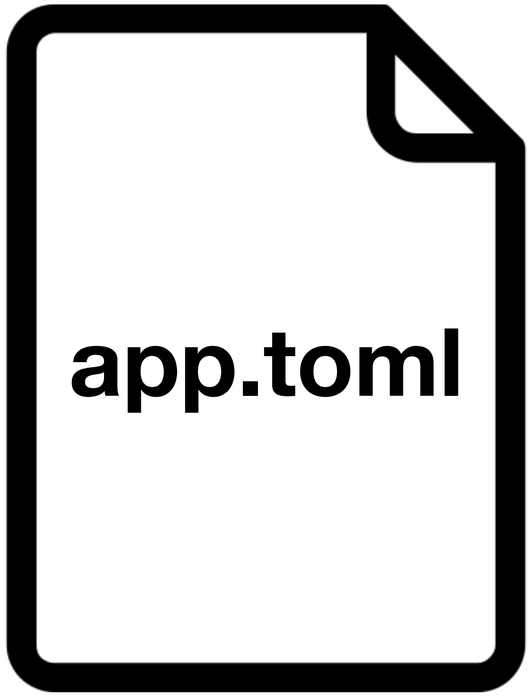
1. plain kubernetes манифесты
2. helm чарты
3. ~ 40 Kb манифестов на сервис

Конфигурация 😕



diff: несколько строк

Конфигурация 😎



app.toml 😎

```
name = "user"
description = "process user info"
kind = "business"
replicas = 1

[engine]
name = "golang"
version = "1.14"
size = "small"

[envs.prod]
replicas = 70
```

app.toml

app.toml

1. Выносим только "движущиеся" части

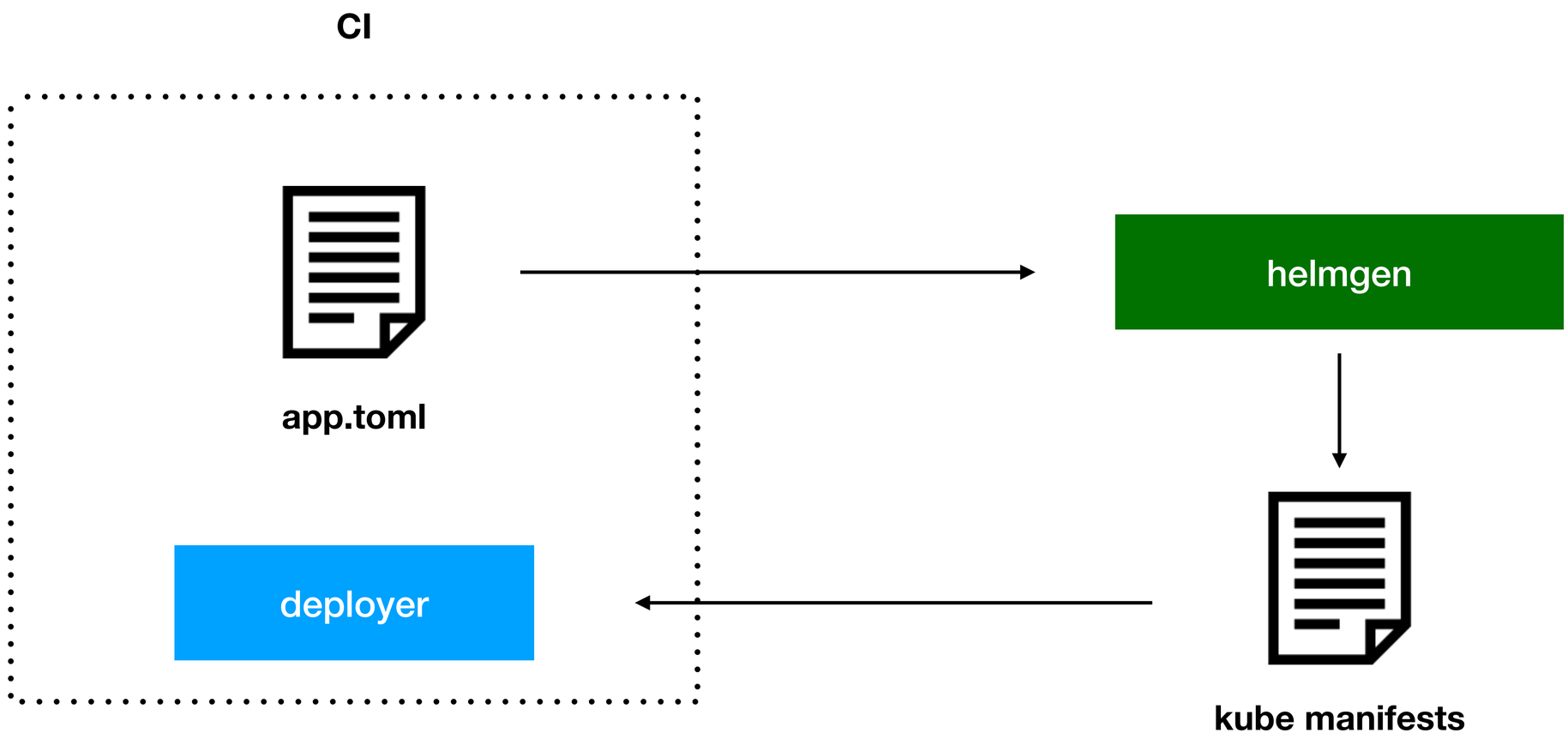
app.toml

1. Выносим только "движущиеся" части
2. Добиваемся "плоской" структуры с помощью toml

app.toml

1. Выносим только "движущиеся" части
2. Добиваемся "плоской" структуры с помощью toml
3. Конфигурируем настройки по окружениям с помощью переменных окружения

Как теперь выглядит процесс получения манифестов



helmgen

helmgen

1. Является сервисом и все новые фичи становятся доступны сразу всем

helmgen

1. Является сервисом и все новые фичи становятся доступны сразу всем
2. Получает минимальный app.toml

helmgen

1. Является сервисом и все новые фичи становятся доступны сразу всем
2. Получает минимальный app.toml
3. Отдает готовые для применения в кластер манифесты

Управление секретами 😕

Управление секретами 😕

- Vault

Управление секретами 😕

- Vault
- Helm манифесты с init контейнерами для похода в runtime при старте pod'ов

Управление секретами 😕

Управление секретами 😕

1. Необходимо понимание схемы работы с vault

Управление секретами 😕

1. Необходимо понимание схемы работы с vault
2. Руками разложить по нужным путям ключи

Управление секретами 😕

1. Необходимо понимание схемы работы с vault
2. Руками разложить по нужным путям ключи
3. Правильно подключить интеграцию в helm манифестах

Управление секретами 😕

1. Необходимо понимание схемы работы с vault
2. Руками разложить по нужным путям ключи
3. Правильно подключить интеграцию в helm манифестах
4. Узнать о корректности уже в production

Управление секретами 😎

Управление секретами 😎

1. Секреты – часть конфигурации

Управление секретами 😎

1. Секреты – часть конфигурации
2. С помощью PaaS Dashboard разработчики изменяют секреты для сервиса (используя dex авторизацию)

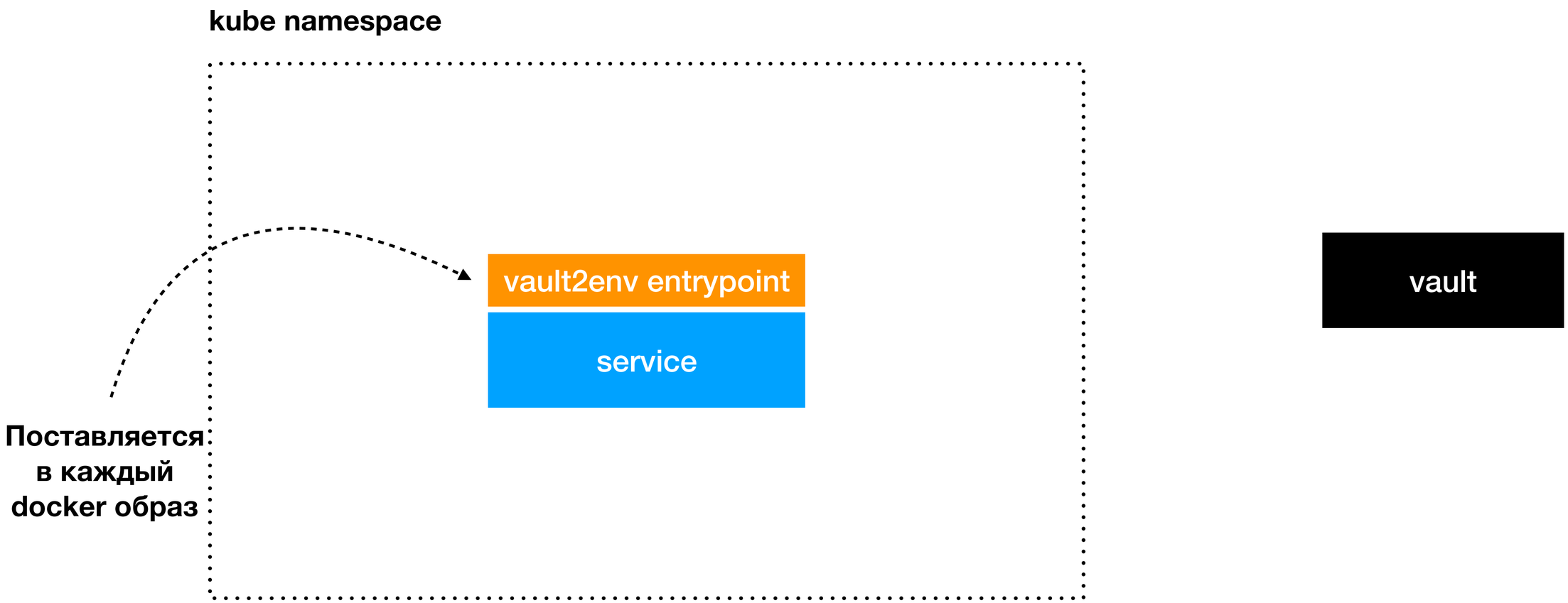
Управление секретами 😎

1. Секреты – часть конфигурации
2. С помощью PaaS Dashboard разработчики изменяют секреты для сервиса (используя dex авторизацию)
3. Каждый сервис раскатывается в свой namespace – security единица

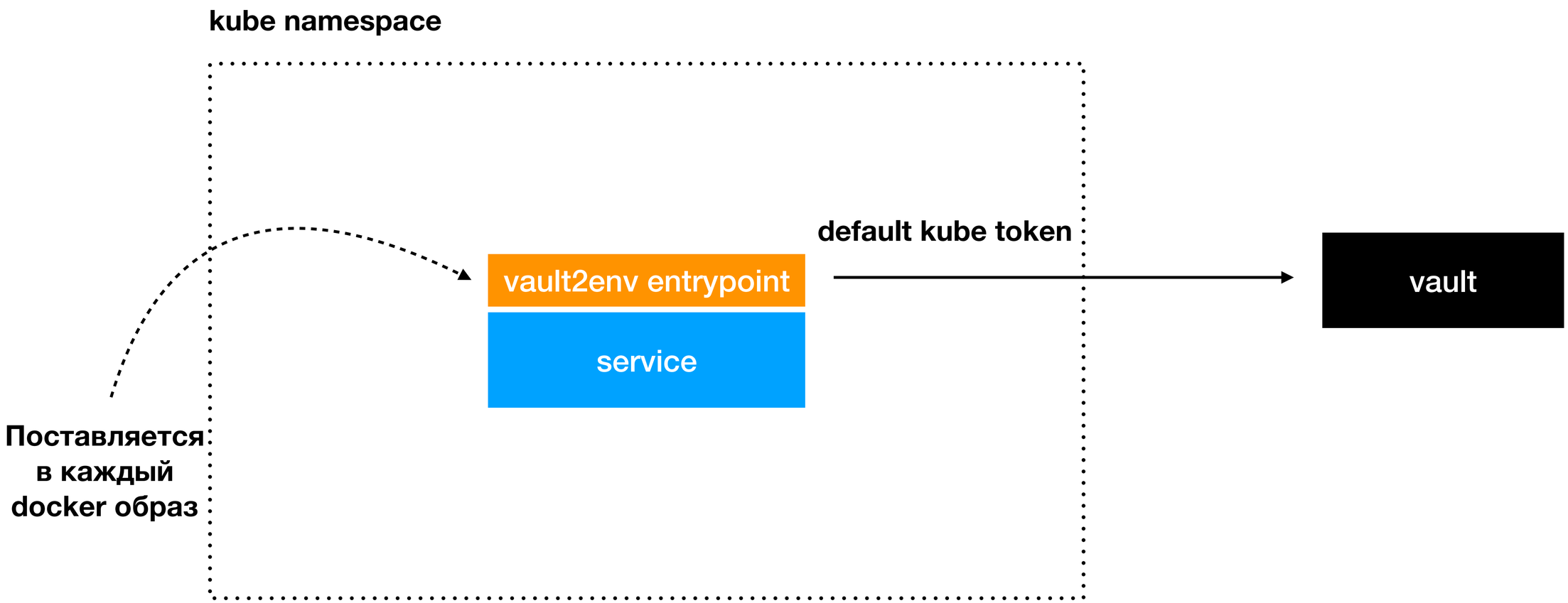
Управление секретами 😎

1. Секреты – часть конфигурации
2. С помощью PaaS Dashboard разработчики изменяют секреты для сервиса (используя dex авторизацию)
3. Каждый сервис раскатывается в свой namespace – security единица
4. В runtime все сервисы получают секреты с помощью vault2env утилиты

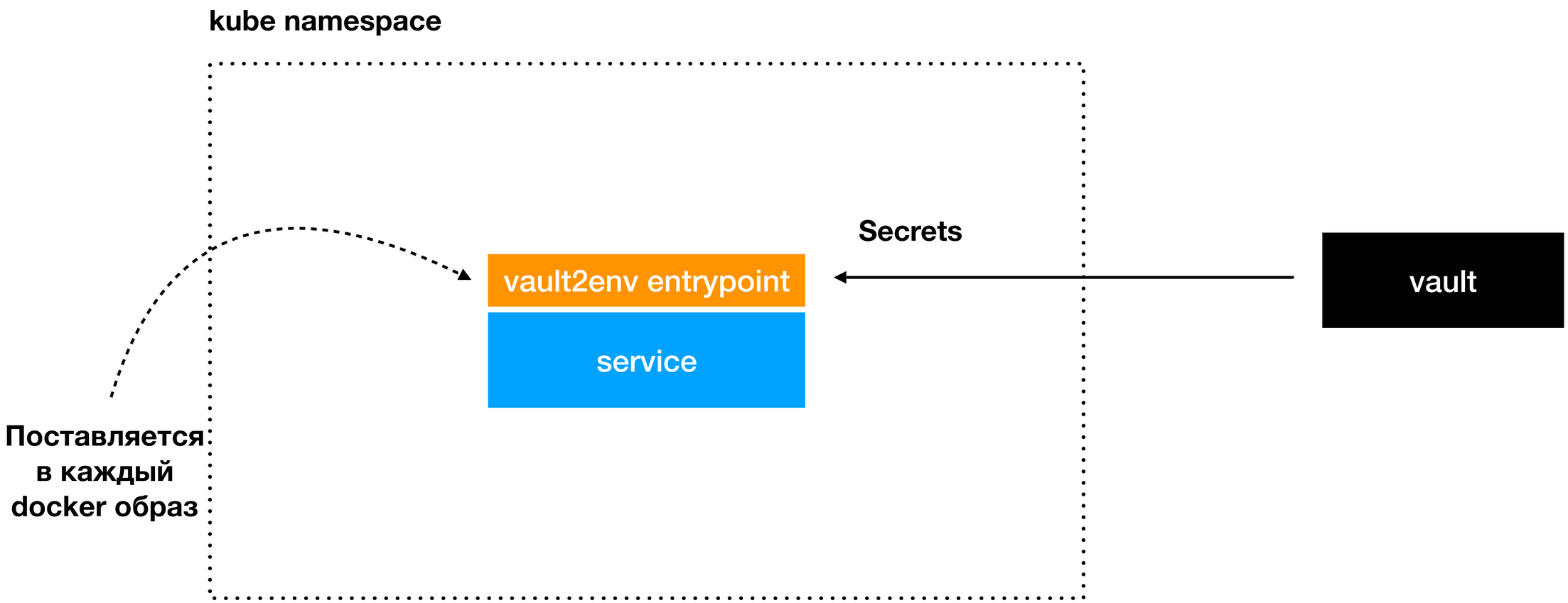
Управление секретами 😎



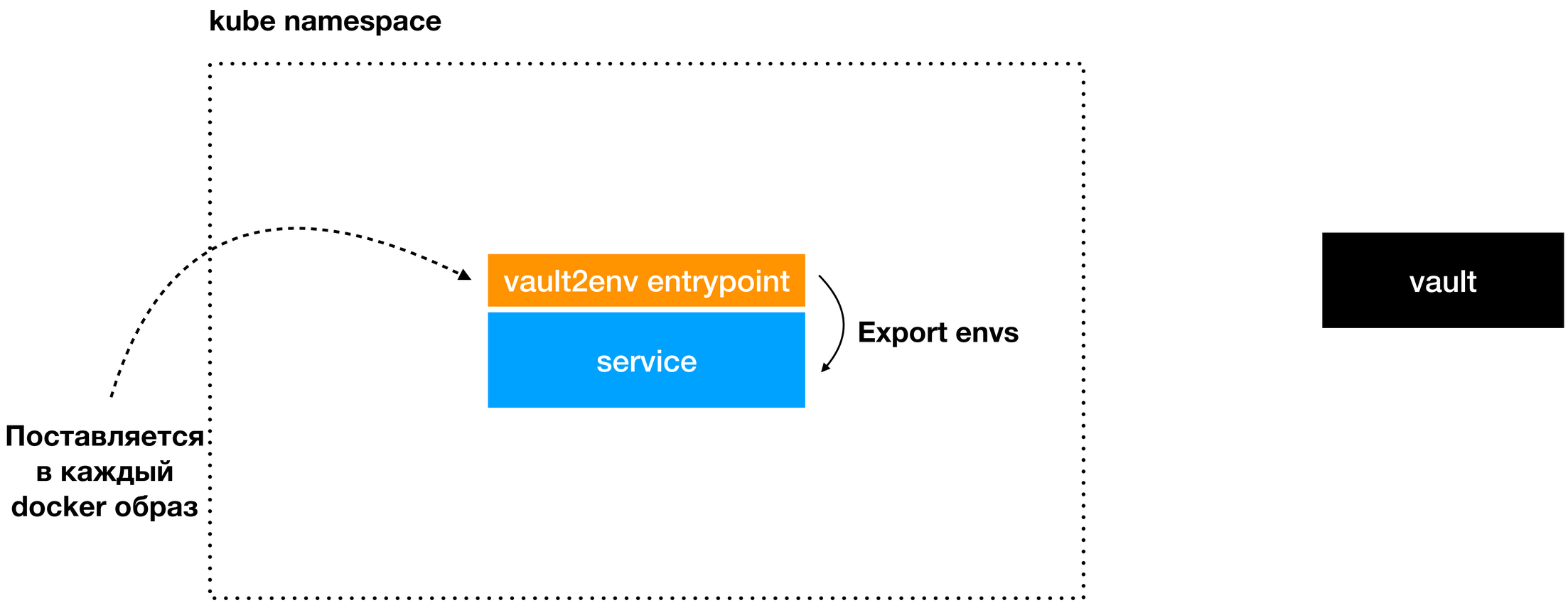
Управление секретами 😎



Управление секретами 😎



Управление секретами 😎



Взаимодействие сервисов 😕

Взаимодействие сервисов 😕

1. Нужно сходить в сервис – пиши клиента

Взаимодействие сервисов 😕

1. Нужно сходить в сервис – пиши клиента
2. Не забудь как обработать все ошибки

Взаимодействие сервисов 😕

1. Нужно сходить в сервис – пиши клиента
2. Не забудь как обработать все ошибки
3. Circuit breaker библиотеку подключил?

Взаимодействие сервисов 😕

1. Нужно сходить в сервис – пиши клиента
2. Не забудь как обработать все ошибки
3. Circuit breaker библиотеку подключил?
4. Все ли нужные header'ы прокинуты?

Взаимодействие сервисов 😕

1. Нужно сходить в сервис – пиши клиента
2. Не забудь как обработать все ошибки
3. Circuit breaker библиотеку подключил?
4. Все ли нужные header'ы прокинуты?
5. Timeout'ы соответствуют NFR сервера?

Взаимодействие сервисов 😎

Взаимодействие сервисов 😎

1. Контракты с двух сторон (клиент и сервер)

Взаимодействие сервисов 😎

1. Контракты с двух сторон (клиент и сервер)
2. Автогенерация кода

Взаимодействие сервисов 😎

1. Контракты с двух сторон (клиент и сервер)
2. Автогенерация кода
3. Поддержка всех основных паттернов взаимодействия из коробки

Взаимодействие сервисов 😎

```
service "summer"

rpc sum (SumIn) SumOut `A sum method`

message SumIn {
    a    int    `A first number`
    b    int    `A second number`
}

message SumOut {
    sum    int    `A sum of the numbers`
    info   string `Additional info`
}
```


Контракты

./rpc/summer.brief

```
service "summer"

rpc sum (SumIn) SumOut `A sum method`

message SumIn {
    a    int    `A first number`
    b    int    `A second number`
}

message SumOut {
    sum    int    `A sum of the numbers`
}
```


service.brief

```
service "summer"

rpc sum (SumIn) SumOut `A sum method`

message SumIn {
    a    int    `A first number`
    b    int    `A second number`
}

message SumOut {
    sum    int    `A sum of the numbers`
    info   string `Additional info`
}
```


Контракты

./rpc/summer.brief

```
service "summer"

rpc sum (SumIn) SumOut `A sum method`

message SumIn {
    a    int    `A first number`
    b    int    `A second number`
}

message SumOut {
    sum    int    `A sum of the numbers`
}
```


service.brief

```
service "summer"

rpc sum (SumIn) SumOut `A sum method`

message SumIn {
    a    int    `A first number`
    b    int    `A second number`
}

message SumOut {
    sum    int    `A sum of the numbers`
    info   string `Additional info`
}
```

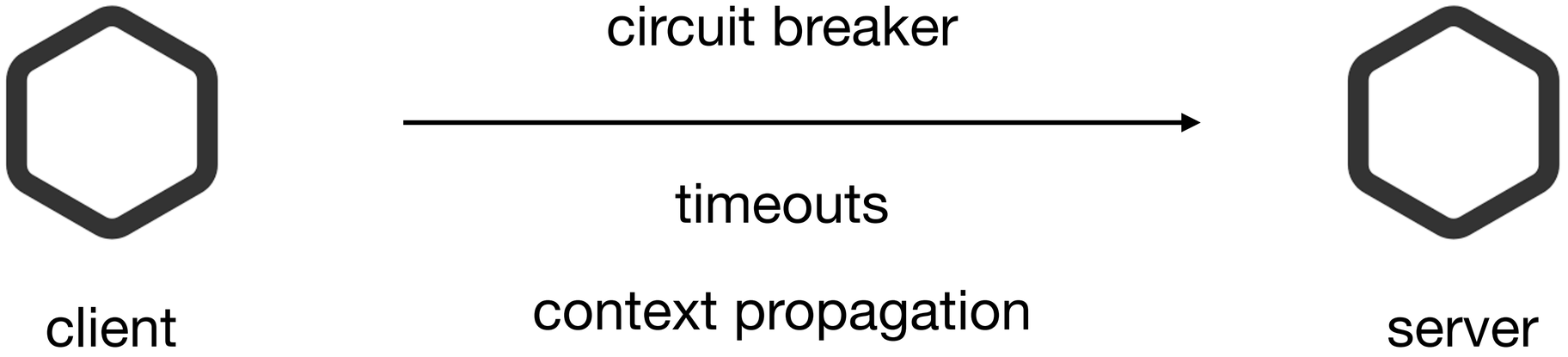
Кодогенерация

./rpc/summer.brief

```
service "summer"

rpc sum (SumIn) SumOut `A sum method`

message SumIn {
    a    int    `A first number`
    b    int    `A second number`
}

message SumOut {
    sum    int    `A sum of the numbers`
}
```

+ $ avito codegen ⟶

Кодогенерация



Синхронное 😎

Синхронное 😎

1. Межсервисное взаимодействие с помощью внутреннего RPC протокола

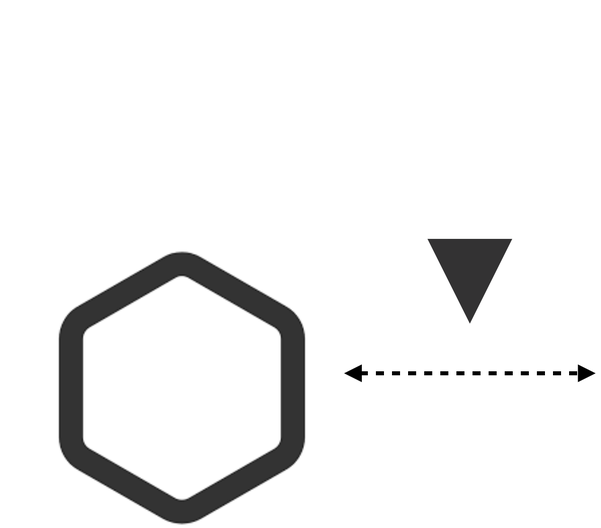
Синхронное 😎

1. Межсервисное взаимодействие с помощью внутреннего RPC протокола
2. Со стороны разработчика подробности реализации скрыты (нет доступа к транспорту)

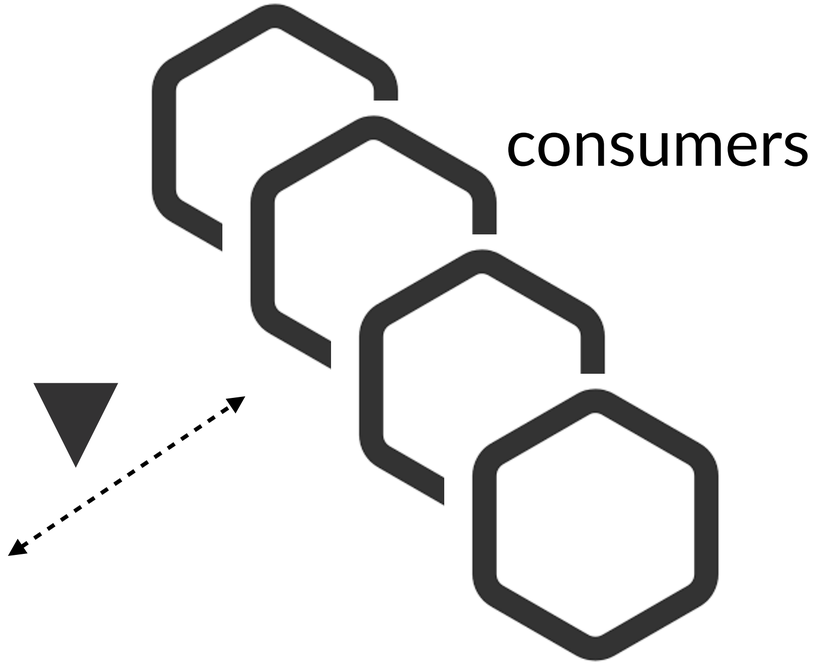
Синхронное 😎

1. Межсервисное взаимодействие с помощью внутреннего RPC протокола
2. Со стороны разработчика подробности реализации скрыты (нет доступа к транспорту)
3. В автогенеренных клиентах сразу реализованы circuit breaker, retry на идемпотентные запросы, проброс контекста

Асинхронное 😎



service-item



Асинхронное 😎

```
schema "service.create" ServiceCreate `Создание сервиса`

message ServiceCreate {
    serviceId  int
    userId     int
}
```

Асинхронное 😎

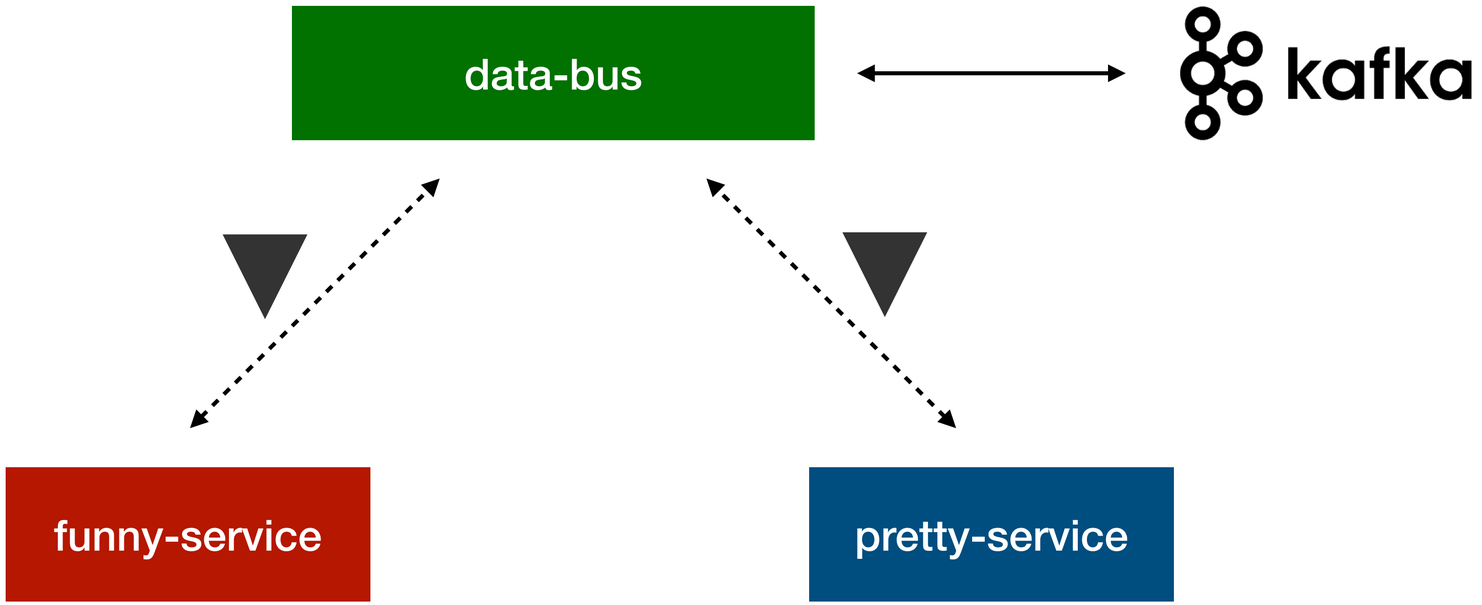
Асинхронное 😎

1. Используем тот же формат описания контракта взаимодействия

Асинхронное 😎

1. Используем тот же формат описания контракта взаимодействия
2. Сервисы автоматически интегрируются с сервисом "шина данных"

Асинхронное 😎



1. Создание сервисов
2. Разработка сервисов
3. Тестирование сервисов
4. Доставка сервисов
5. Эксплуатация сервисов

Avito

Тесты 😕

Тесты 😕

1. Тестирование в микросервисах – непросто

Тесты 😕

1. Тестирование в микросервисах – непросто
2. Миграция тех же подходов как в монолите приводит к плохим результатам

Тесты 😕

1. Тестирование в микросервисах – непросто
2. Миграция тех же подходов как в монолите приводит к плохим результатам
3. Но мы все еще хотим прогнозируемое качество всех основных сценариев и нам неважно какие сервисы в этом участвуют

Тесты 

Тесты 😎

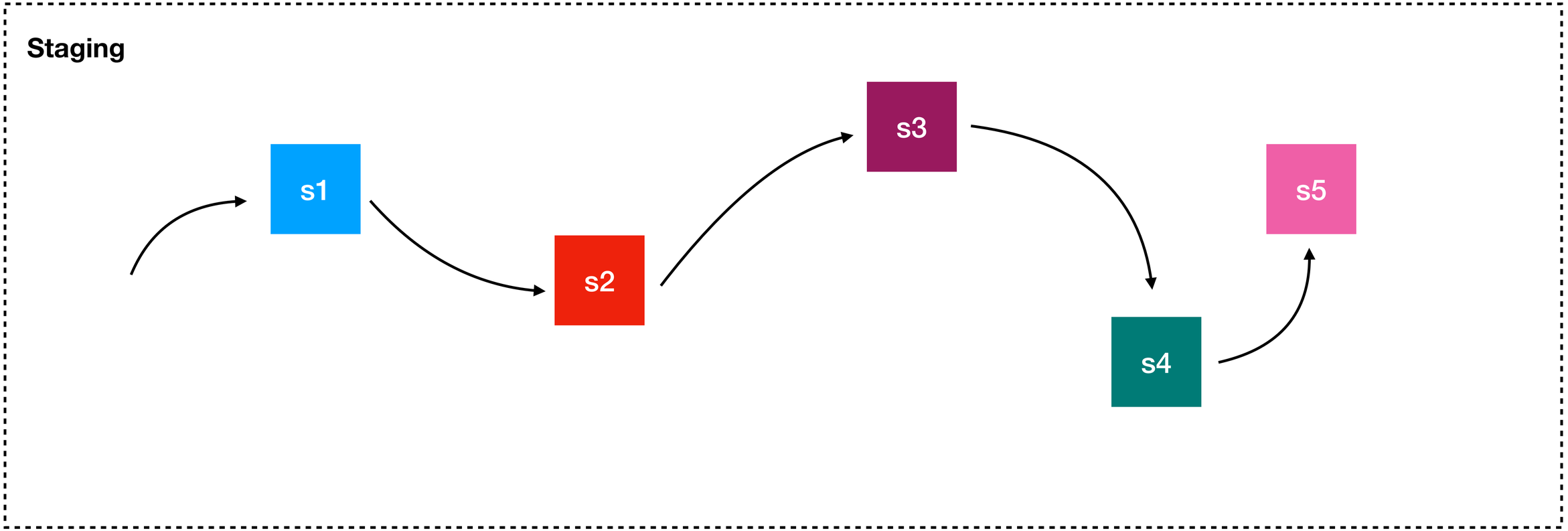
1. Всю основную бизнес логику тестируем unit тестами

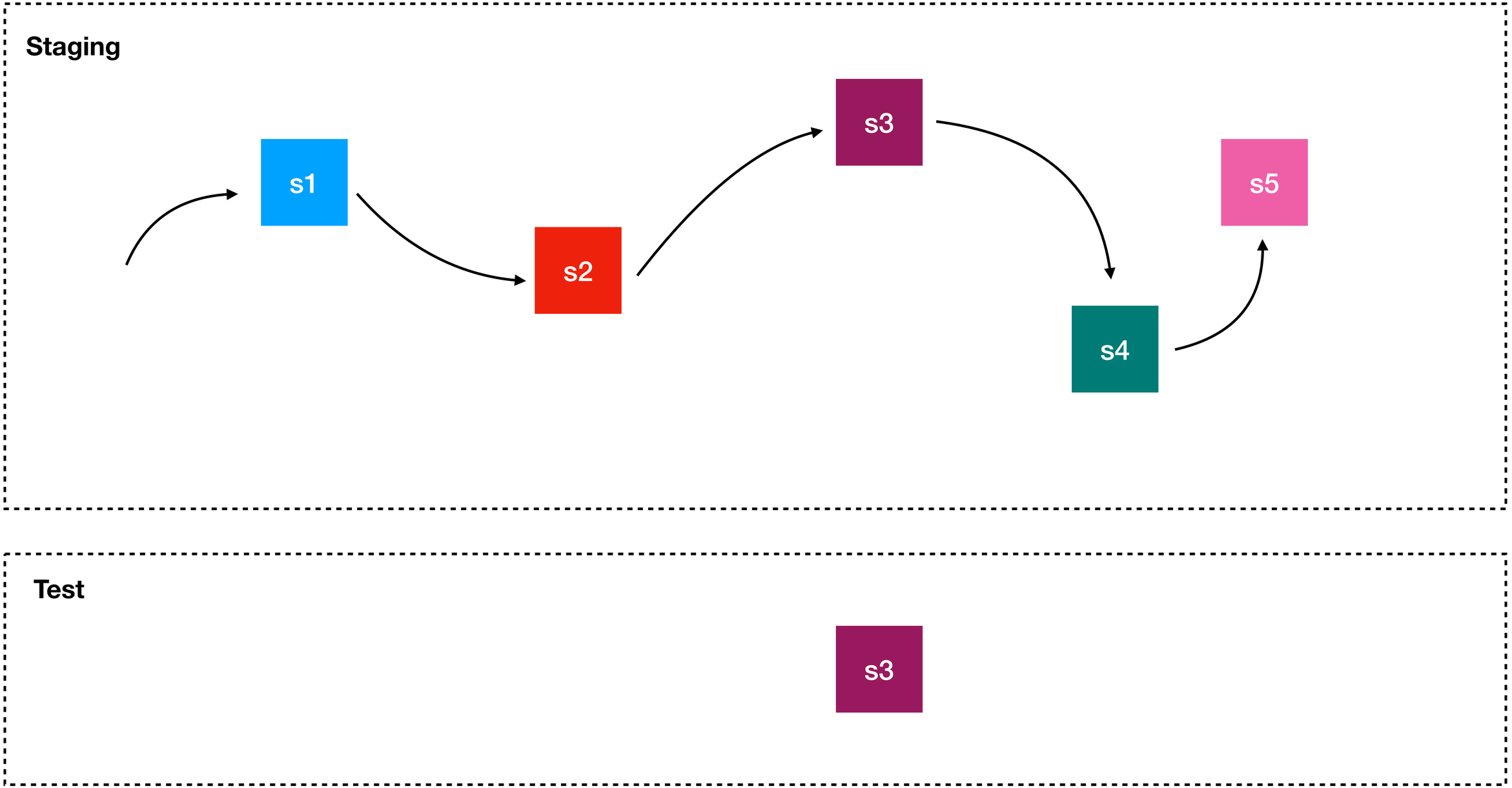
Тесты 😎

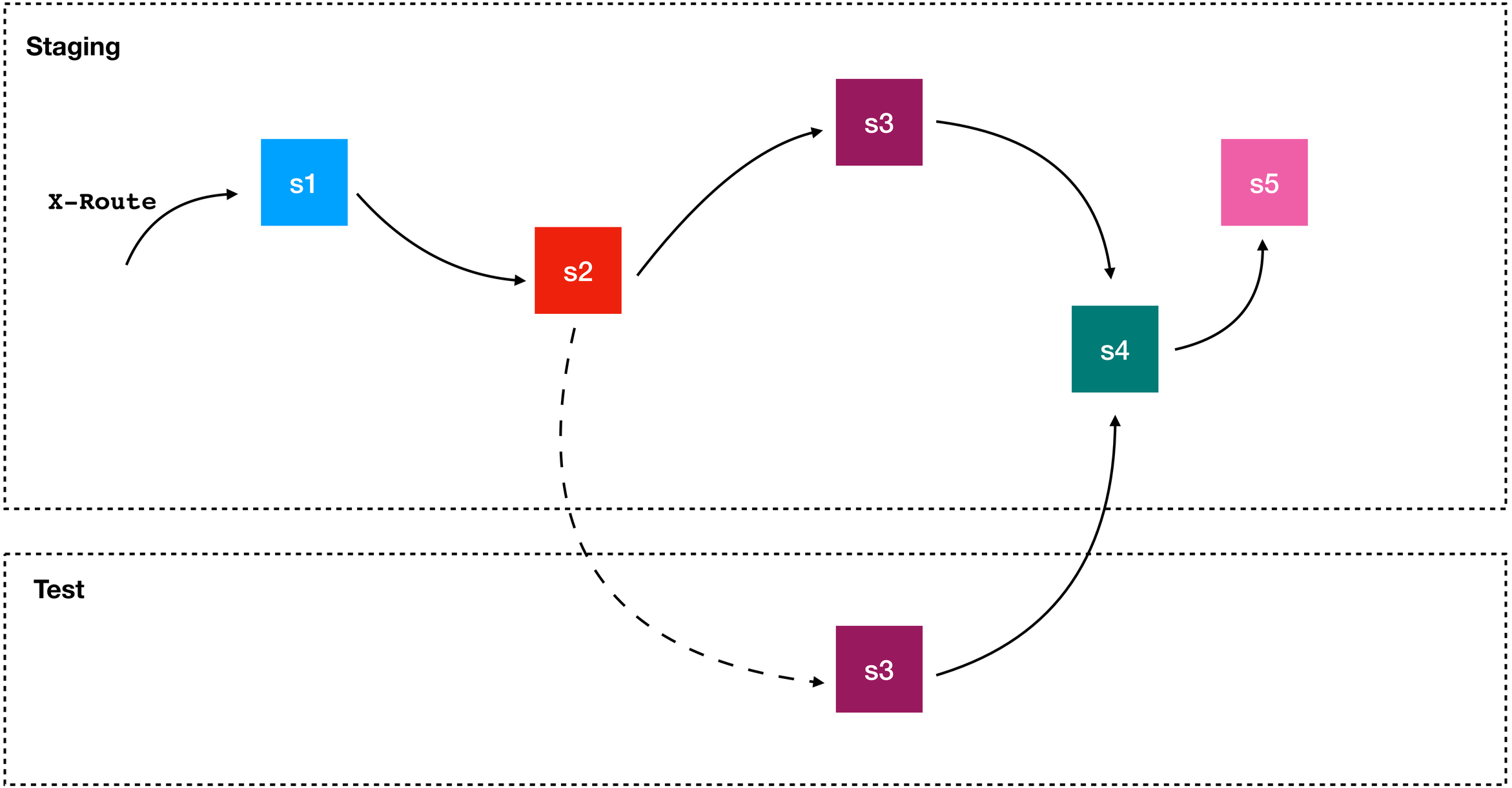
1. Всю основную бизнес логику тестируем unit тестами
2. e2e тесты оставляем только на критичные бизнес пути

Тесты 😎

1. Всю основную бизнес логику тестируем unit тестами
2. e2e тесты оставляем только на критичные бизнес пути
3. Для e2e применяем новый подход


Staging
s1
s2
s3
s4
s5
Test


Staging
s1
s2
s3
s4
s5
Test
s3


Staging
X-Route
s1
s2
s3
s4
s5
Test
s3

1. Создание сервисов
2. Разработка сервисов
3. Тестирование сервисов
4. Доставка сервисов
5. Эксплуатация сервисов

Avito

Deploy 😕

Deploy 😕

1. Helm – хороший инструмент с плохим UX

Deploy 😕

1. Helm – хороший инструмент с плохим UX
2. Несколько кластеров в одном окружении все ломает

Deploy 😕

1. Helm – хороший инструмент с плохим UX
2. Несколько кластеров в одном окружении все ломает
3. Транзакционность деплоев – боль

Deploy

Deploy

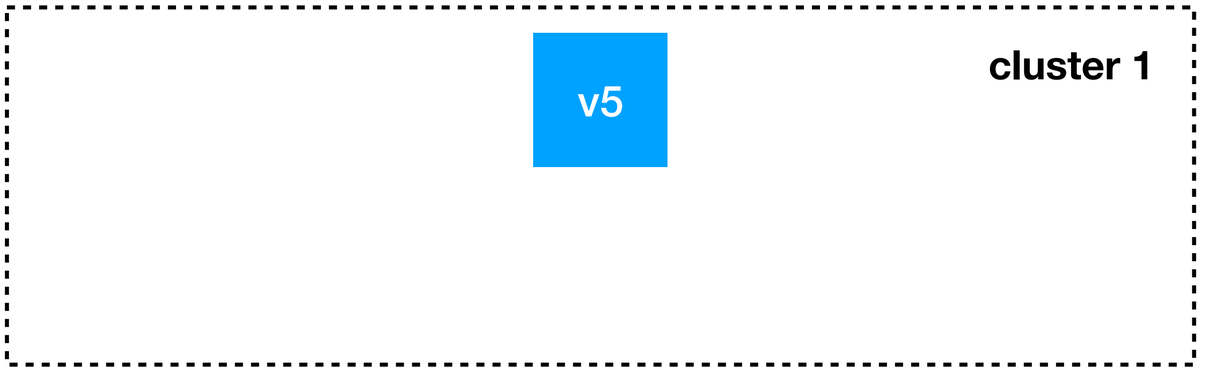
1. Мы уже применяем готовые манифесты для кластеров

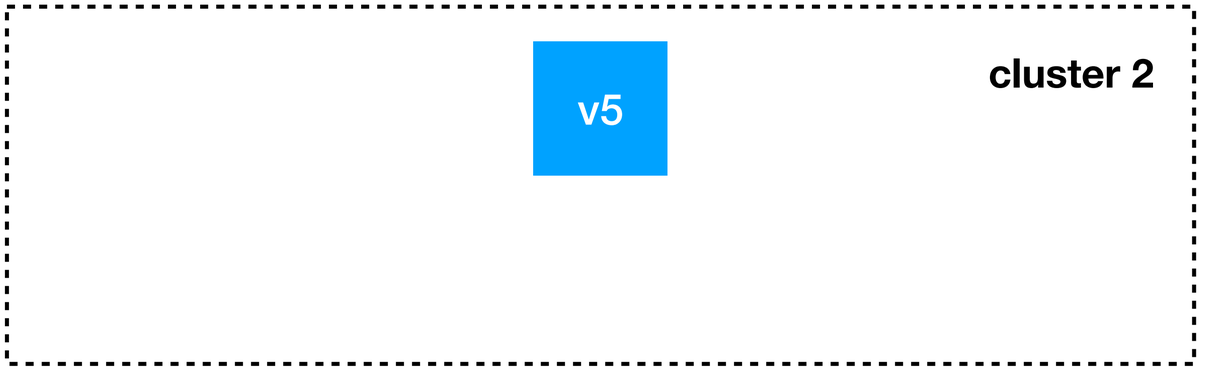
Deploy

1. Мы уже применяем готовые манифесты для кластеров
2. Меняем подход к deploy!

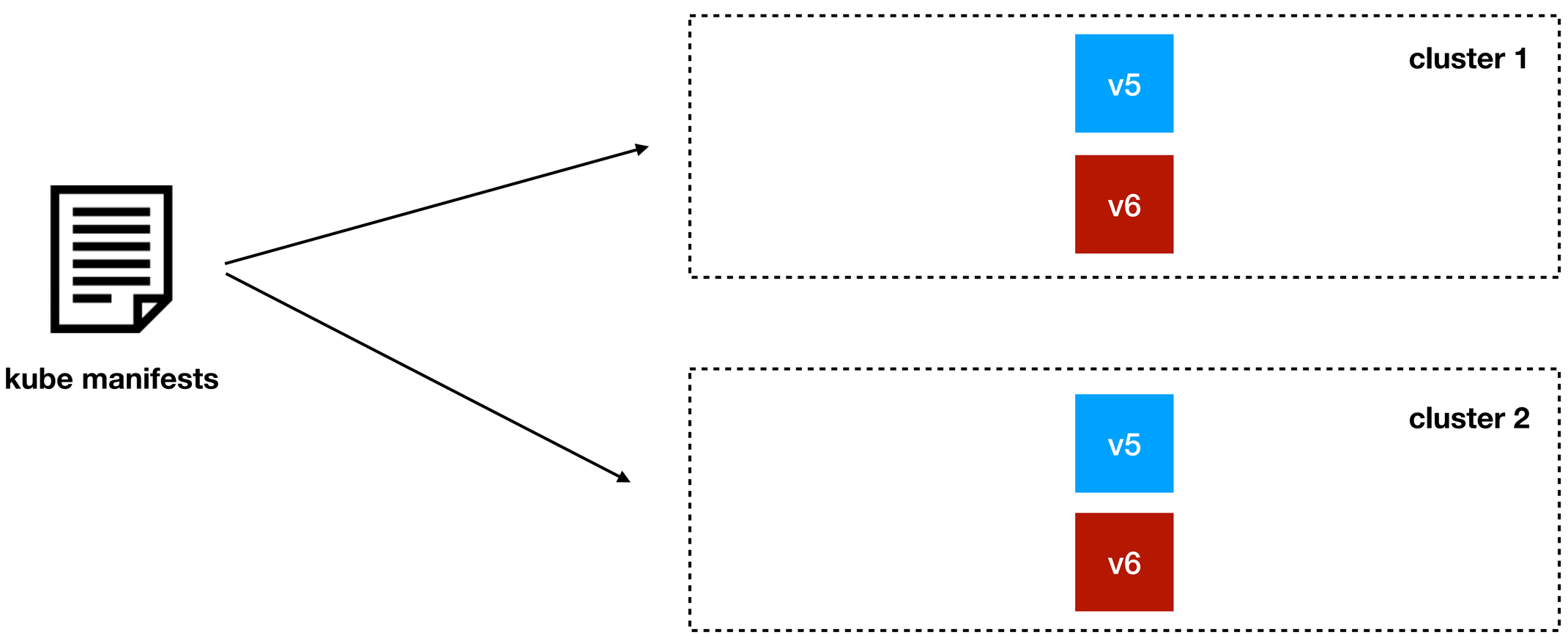
Jibe 😎

kube manifests

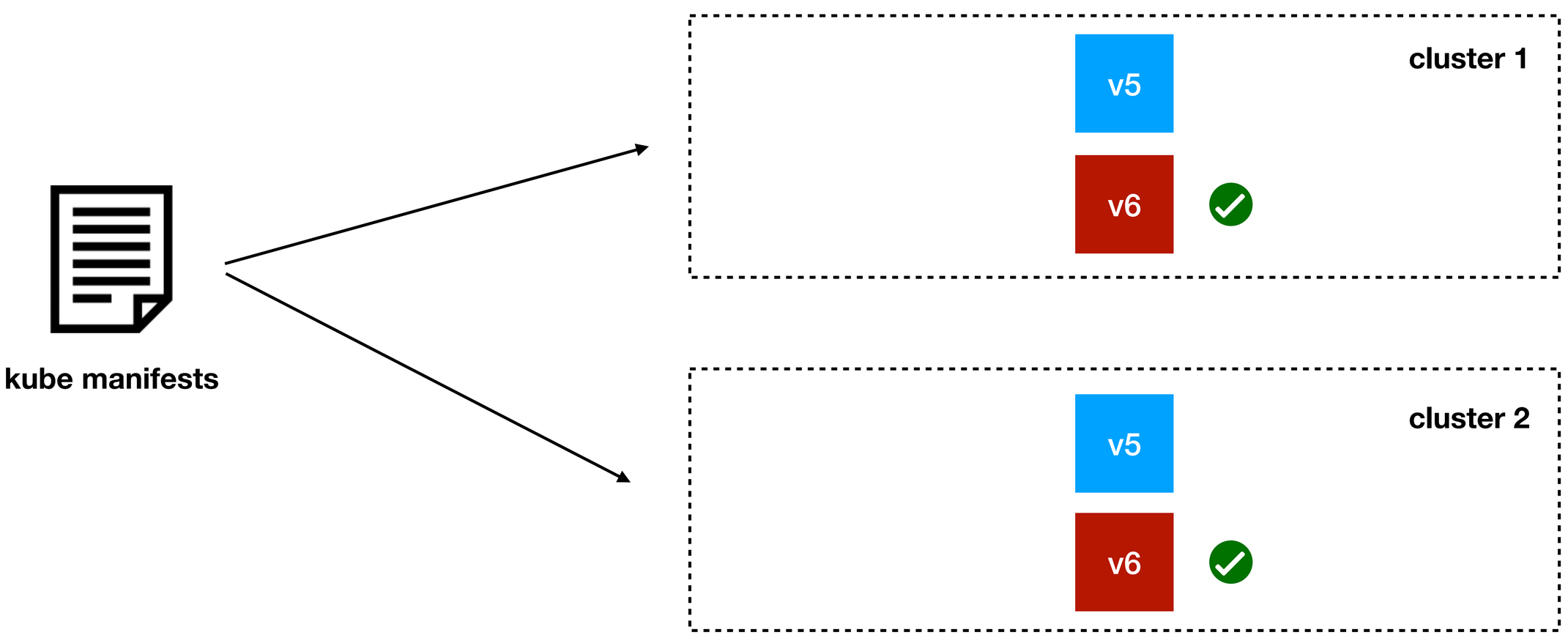




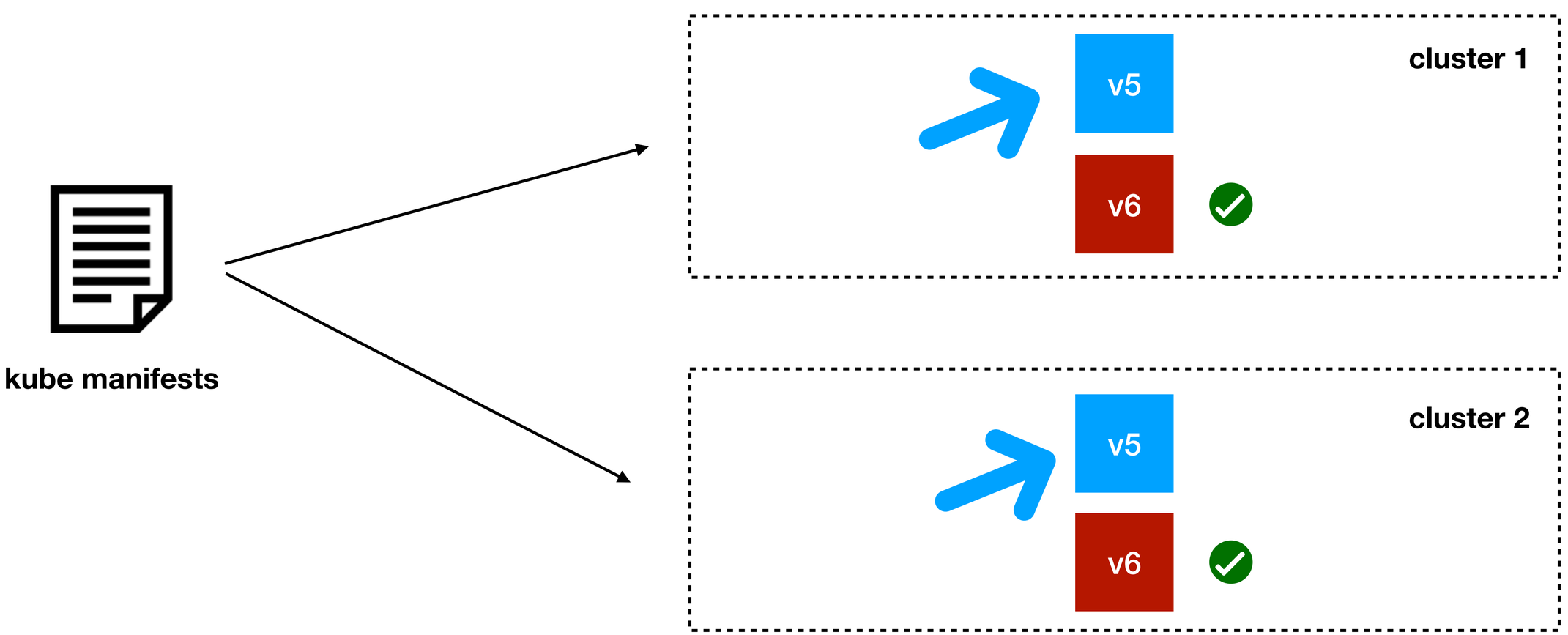
Jibe 😎



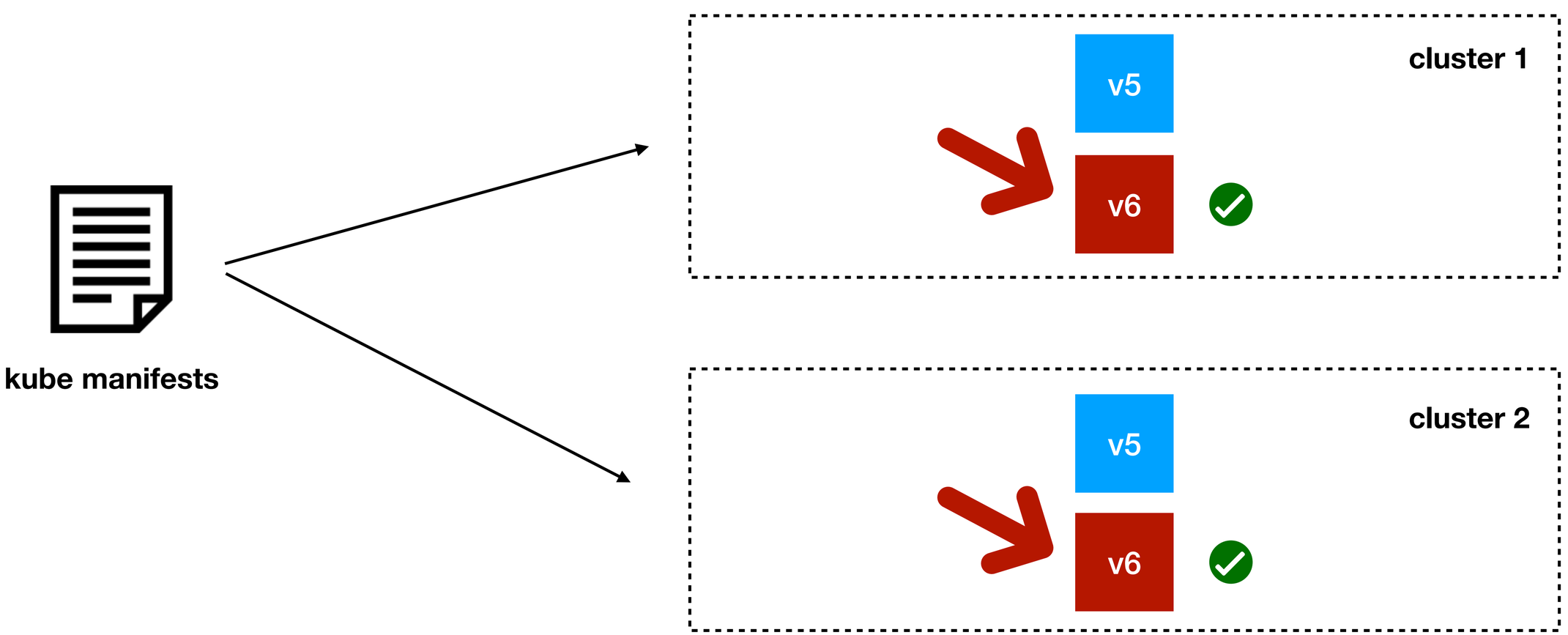
Jibe 😎



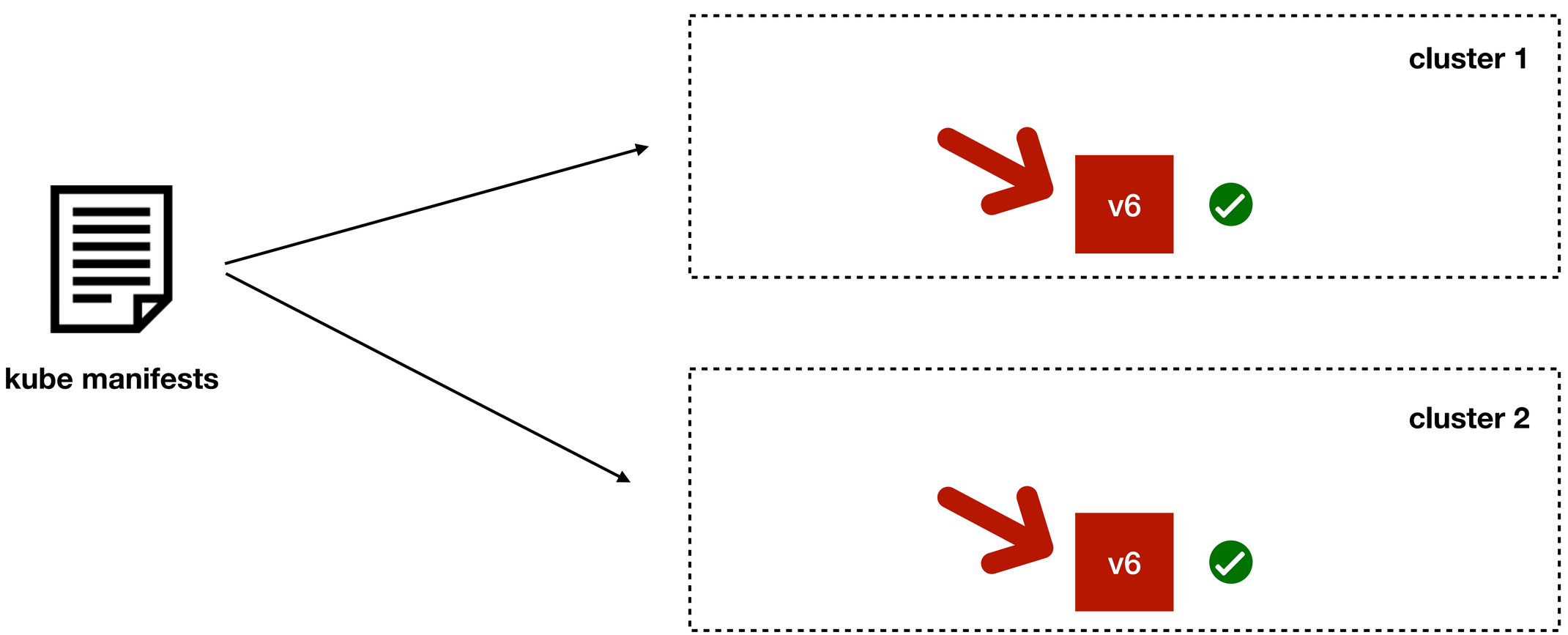
Jibe 😎



Jibe 😎



Jibe 😎



Jibe 😎

Jibe 😎

1. Multistage deploy (init, db, app, balancing phases)

Jibe 😎

1. Multistage deploy (init, db, app, balancing phases)
2. Возможность deploy с ручным контролем (canary, blue-green)

Jibe 😎

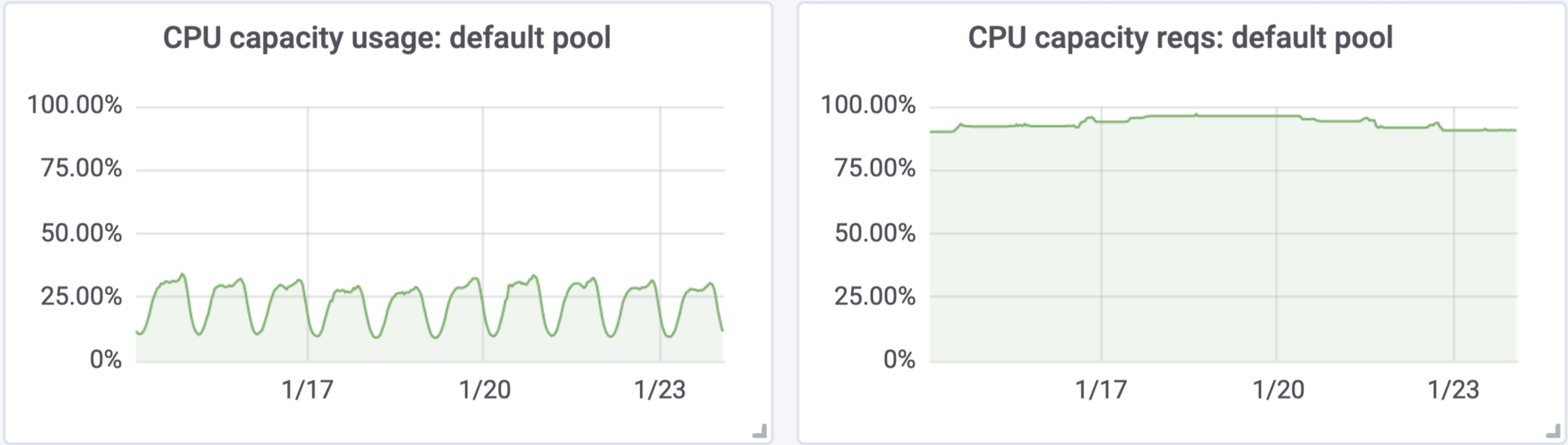
1. Multistage deploy (init, db, app, balancing phases)
2. Возможность deploy с ручным контролем (canary, blue-green)
3. Гарантия консистентности

1. Создание сервисов
2. Разработка сервисов
3. Тестирование сервисов
4. Доставка сервисов
5. Эксплуатация сервисов

Avito

Управление ресурсами

Capacity management 😕



Capacity management 😕

Capacity management 😕

1. Разработчикам сложно определять и планировать нужное количество ресурсов

Capacity management 😕

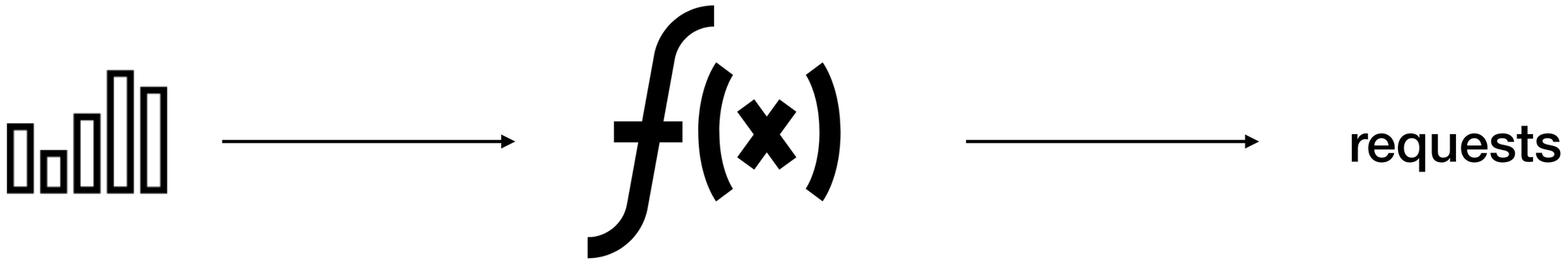
1. Разработчикам сложно определять и планировать нужное количество ресурсов
2. Значения устаревают со временем

Capacity management 😕

1. Разработчикам сложно определять и планировать нужное количество ресурсов
2. Значения устаревают со временем
3. В итоге получаем плохой scheduling, шумных соседей и неэффективное использование ресурсов

```
No nodes are available that match all of the predicates:
Insufficient cpu (13), Insufficient pods (58), NodeNotReady (2),
NodeUnschedulable (5), PodToleratesNodeTaints (67).
```

VPA 😎



VPA 😎

VPA 😎

1. Requests вычисляются на основании статистики предыдущего использования

VPA 😎

1. Requests вычисляются на основании статистики предыдущего использования
2. В начале сервисам дается большой запас для обработки большого количества нагрузки

VPA 😎

1. Requests вычисляются на основании статистики предыдущего использования
2. В начале сервисам дается большой запас для обработки большого количества нагрузки
3. Для резких всплесков мы выдаем лимиты, превышающие запросы в несколько раз

VPA 😎

1. Requests вычисляются на основании статистики предыдущего использования
2. В начале сервисам дается большой запас для обработки большого количества нагрузки
3. Для резких всплесков мы выдаем лимиты, превышающие запросы в несколько раз
4. На серверах поддерживаем всегда запас для выхода за запрошенные ресурсы

VPA 😎

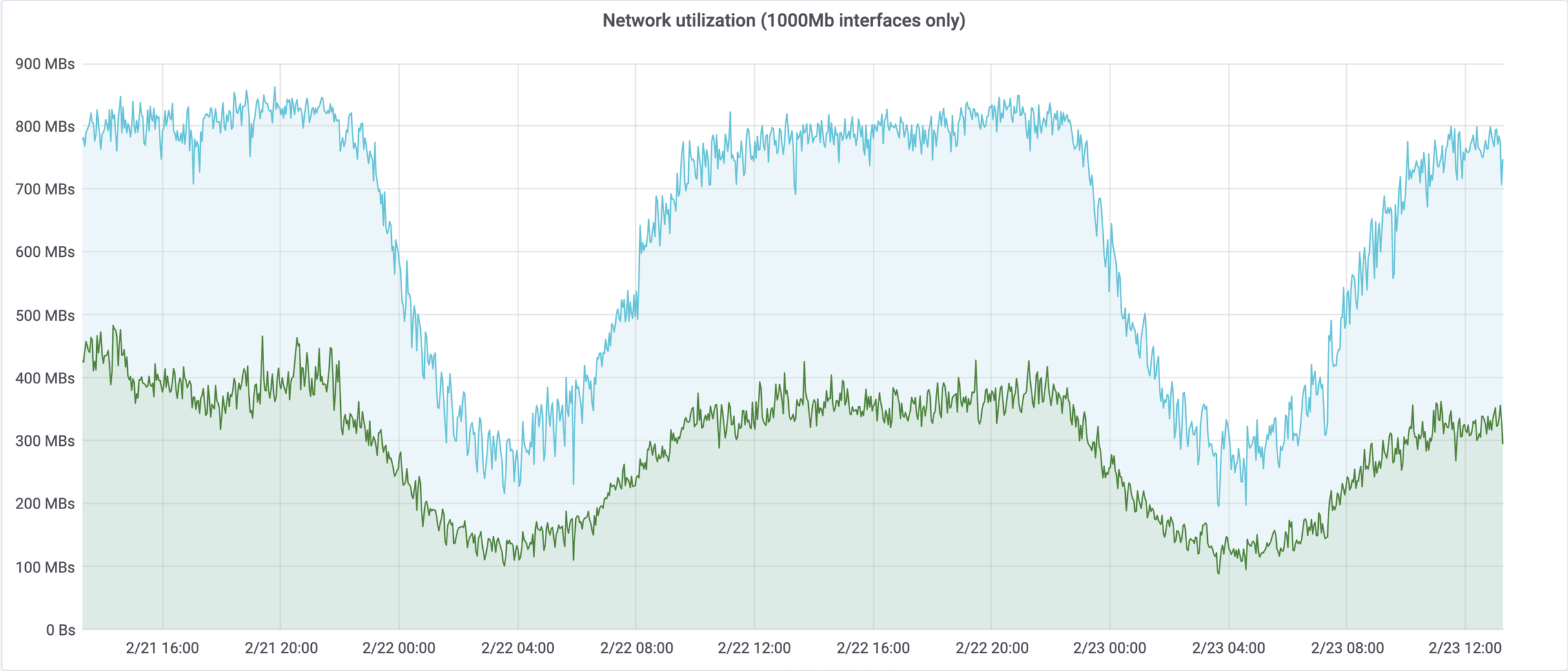
VPA 😎

1. В итоге запросы соответствуют потреблению

VPA 😎

1. В итоге запросы соответствуют потреблению
2. Scheduling предсказуемый без переутилизации физических нод

Network utilization + kubernetes 😕



Network utilization + kubernetes 😕

Network utilization + kubernetes 😕

1. Нативная поддержка только cpu и memory

Network utilization + kubernetes 😕

1. Нативная поддержка только cpu и memory
2. Сетевая утилизация часто не зависит от утилизации процессора

Network utilization + kubernetes 😕

1. Нативная поддержка только cpu и memory
2. Сетевая утилизация часто не зависит от утилизации процессора
3. В итоге получаем утилизированные в полку физические ноды по сети

Network utilization + kubernetes + VPA 😎

Network utilization + kubernetes + VPA 😎

1. https://kubernetes.io/docs/tasks/configure-pod-container/extended-resource/

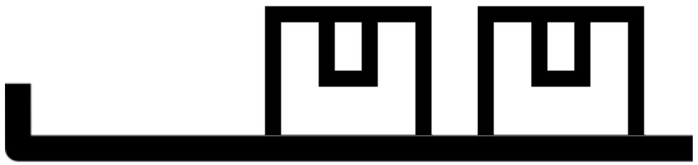
Network utilization + kubernetes + VPA 😎

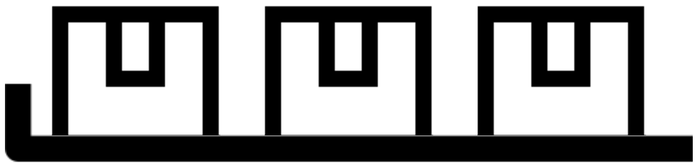
1. https://kubernetes.io/docs/tasks/configure-pod-container/extended-resource/
2. Вводим extended resources – avito.ru/netutil

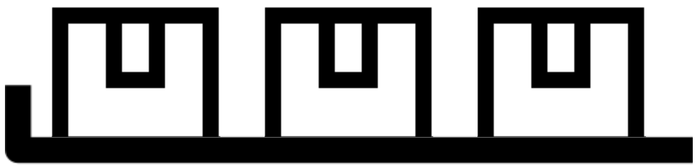
Network utilization + kubernetes + VPA 😎

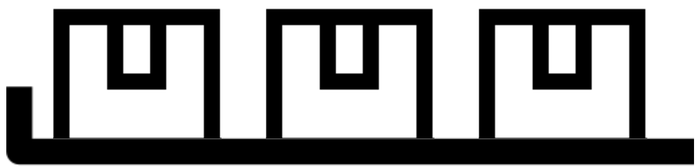
1. https://kubernetes.io/docs/tasks/configure-pod-container/extended-resource/
2. Вводим extended resources – avito.ru/netutil
3. Заполняем тем же подходом, что и cpu, mem

descheduler

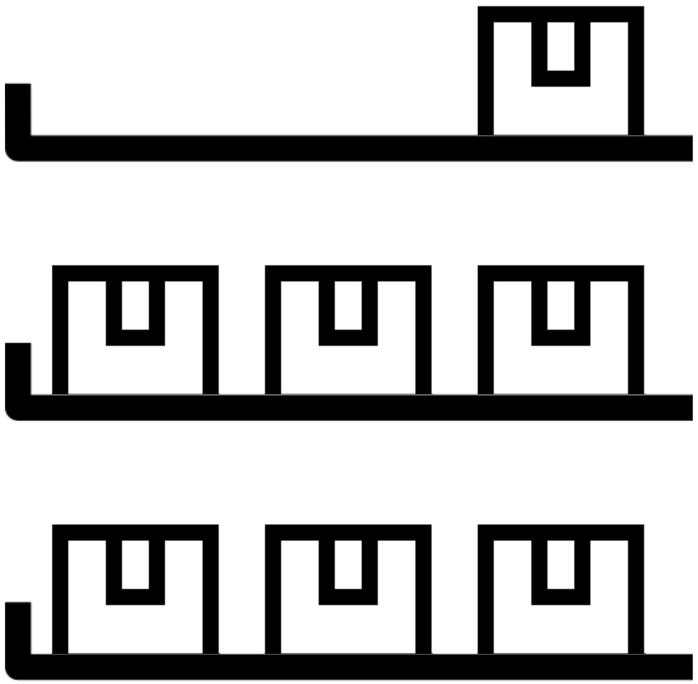


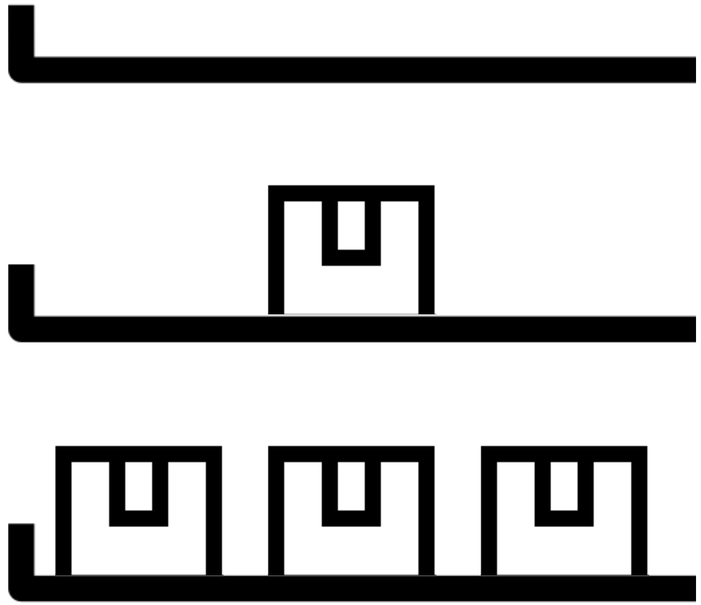




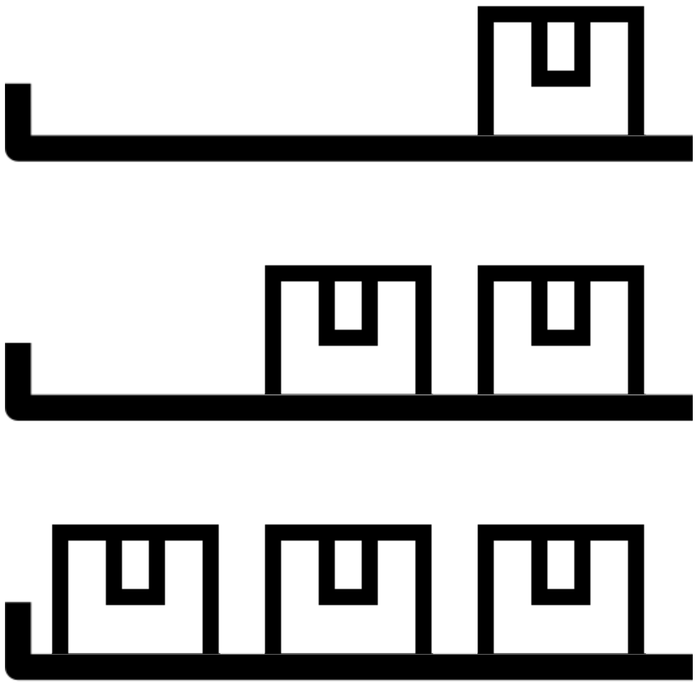


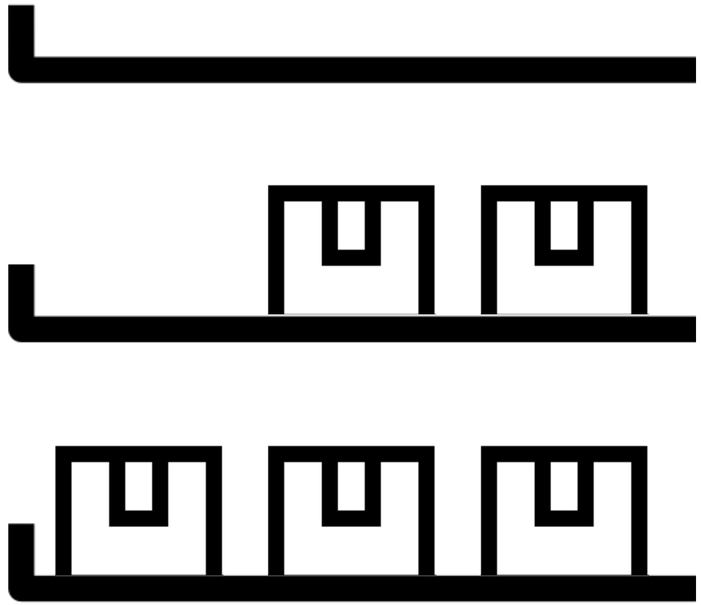
descheduler





descheduler





descheduler

descheduler

1. Смотрим на метрики нод в кластере

descheduler

1. Смотрим на метрики нод в кластере
2. На тех, которых сильно утилизирован хотя бы один параметр, ищем pod с самым высоким потреблением по той же метрике

descheduler

1. Смотрим на метрики нод в кластере
2. На тех, которых сильно утилизирован хотя бы один параметр, ищем pod с самым высоким потреблением по той же метрике
3. Переселяем этот pod

Runtime configuration

Service discovery 😕

Service discovery 😕

1. Хардкод URL'ов в конфигурации

Service discovery 😕

1. Хардкод URL'ов в конфигурации
2. Кто-то пошел через Ingress controller

Service discovery 😕

1. Хардкод URL'ов в конфигурации
2. Кто-то пошел через Ingress controller
3. Кто-то по внутреннему fqdn адресу

Service discovery 😕

1. Хардкод URL'ов в конфигурации
2. Кто-то пошел через Ingress controller
3. Кто-то по внутреннему fqdn адресу
4. Случайно из прода пошел в staging

Service discovery 😕

Пришло время переезжать в новый кластер – что делать?

```
[[dependencies]]
name = "data-bus"

[[dependencies]]
name = "image-storage"
```

```
[[dependencies]]
name = "data-bus"
```

SERVICE_DATA_BUS_URL

```
[[dependencies]]
name = "image-storage"
```

SERVICE_IMAGE_STORAGE_URL

Service discovery 😎

Service discovery 😎

1. Разработчик не указывает никаких URL'ов

Service discovery 😎

1. Разработчик не указывает никаких URL'ов
2. Система сама подкладывает нужные гейты в нужных окружениях

Service discovery 😎

1. Разработчик не указывает никаких URL'ов
2. Система сама подкладывает нужные гейты в нужных окружениях
3. Автогенерируемые клиенты берут значения автоматически

Network & Observability

Синхронное взаимодействие и сеть 😕

Синхронное взаимодействие и сеть 😕

1. Connection timeout несколько раз в час – когда почините?

Синхронное взаимодействие и сеть 😕

1. Connection timeout несколько раз в час – когда почините?
2. Почему у меня запрос улетел в уже мертвый инстанс?

Синхронное взаимодействие и сеть 😕

1. Connection timeout несколько раз в час – когда почините?
2. Почему у меня запрос улетел в уже мертвый инстанс?
3. Как мне понять всю цепочку взаимодействия?

Синхронное взаимодействие и сеть 😎

Синхронное взаимодействие и сеть 😎

1. Navigator service mesh

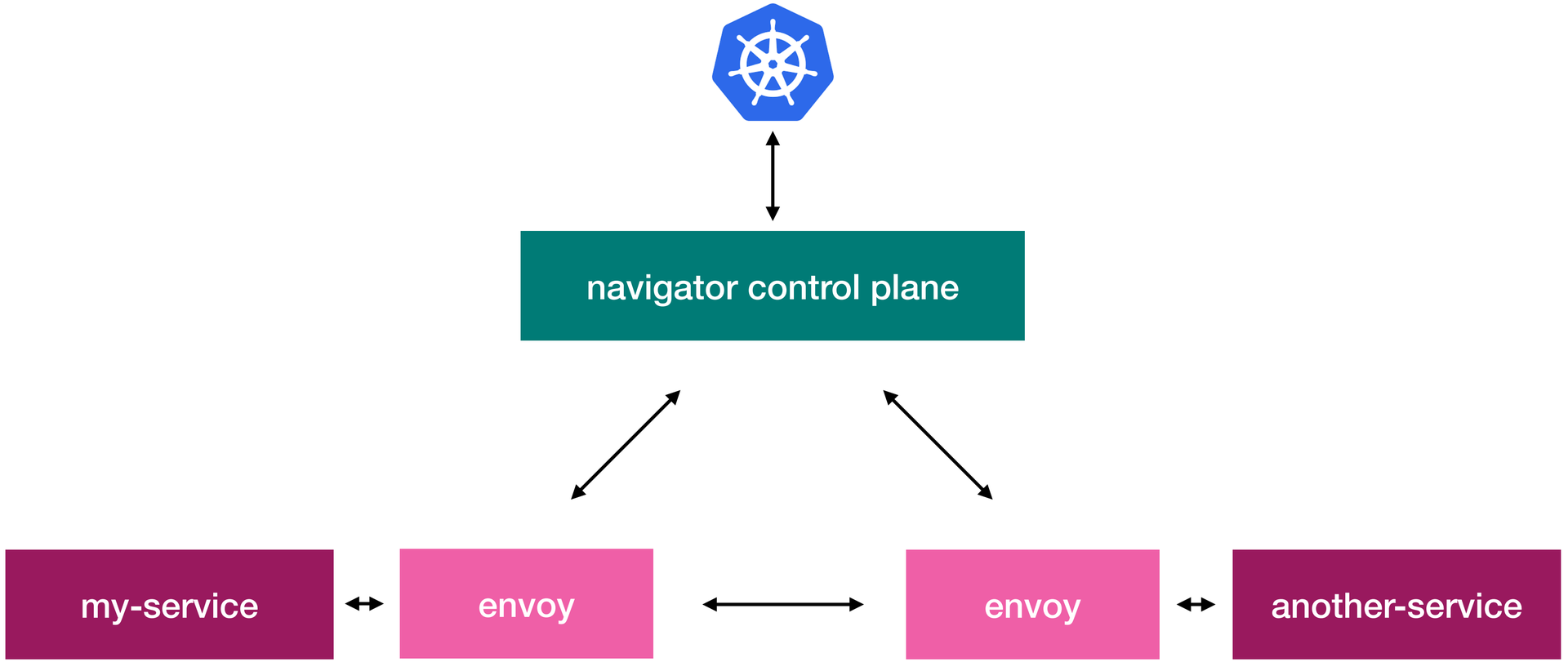
Синхронное взаимодействие и сеть 😎

1. Navigator service mesh
2. Собираем унифицированные метрики по всем взаимодействиям

Синхронное взаимодействие и сеть 😎

1. Navigator service mesh
2. Собираем унифицированные метрики по всем взаимодействиям
3. Для всех включаем outlier detection, connect retries

Синхронное взаимодействие и сеть 😎



Observability 😎

Observability 😎

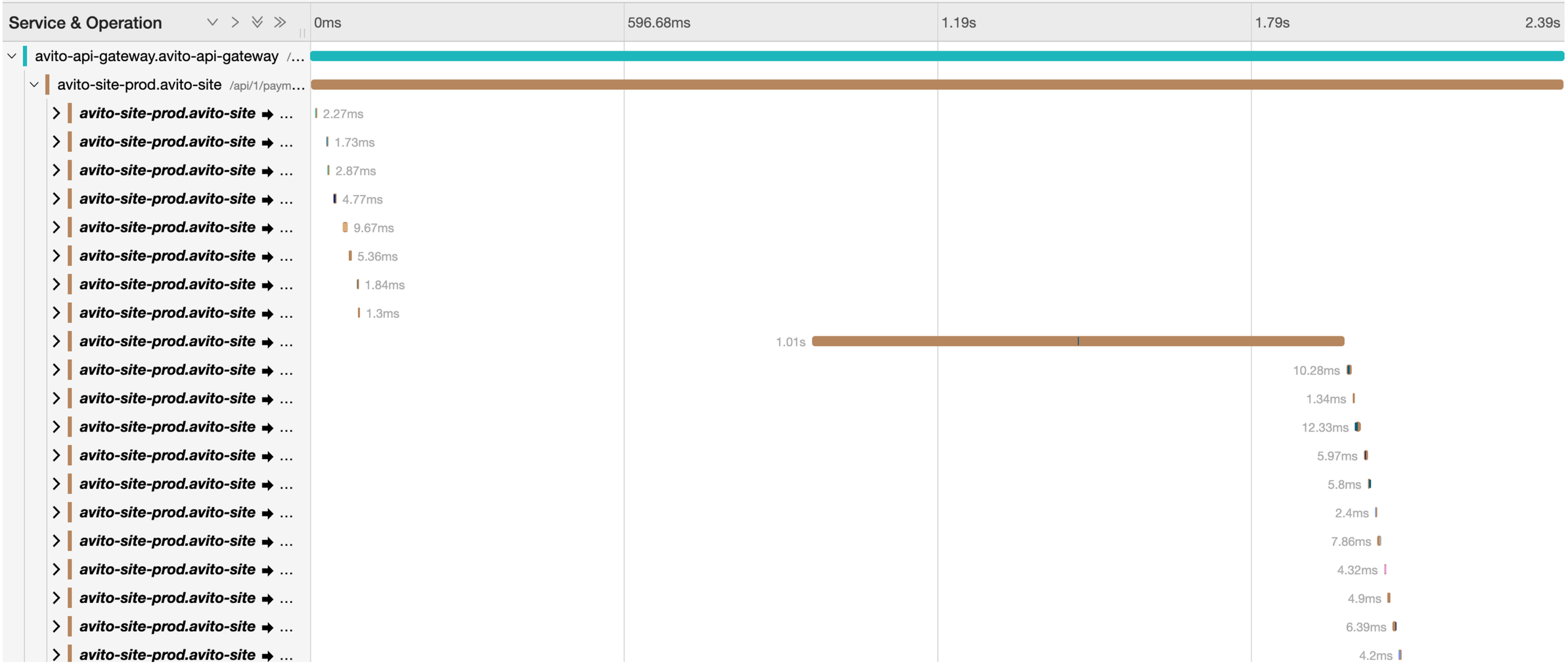
Observability 😎

1. Prometheus endpoint у envoy предоставляет метрики по взаимодействию между сервисами в унифицированном виде

Observability 😎

1. Prometheus endpoint у envoy предоставляет метрики по взаимодействию между сервисами в унифицированном виде
2. Envoy отправляет tracing span'ы, с которыми можно работать через UI tracing системы, либо строить свою логику поверх собранных данных

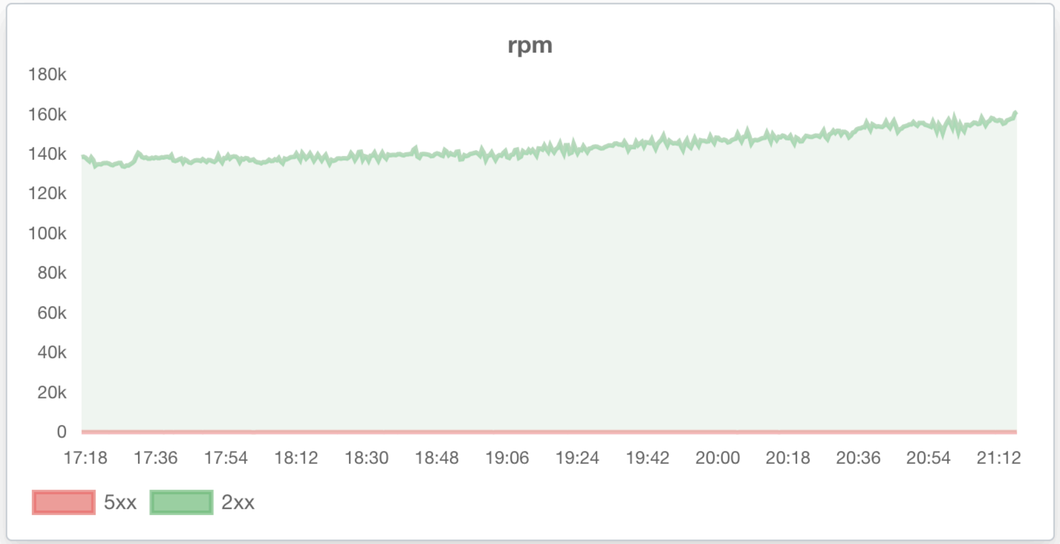
Observability 😎

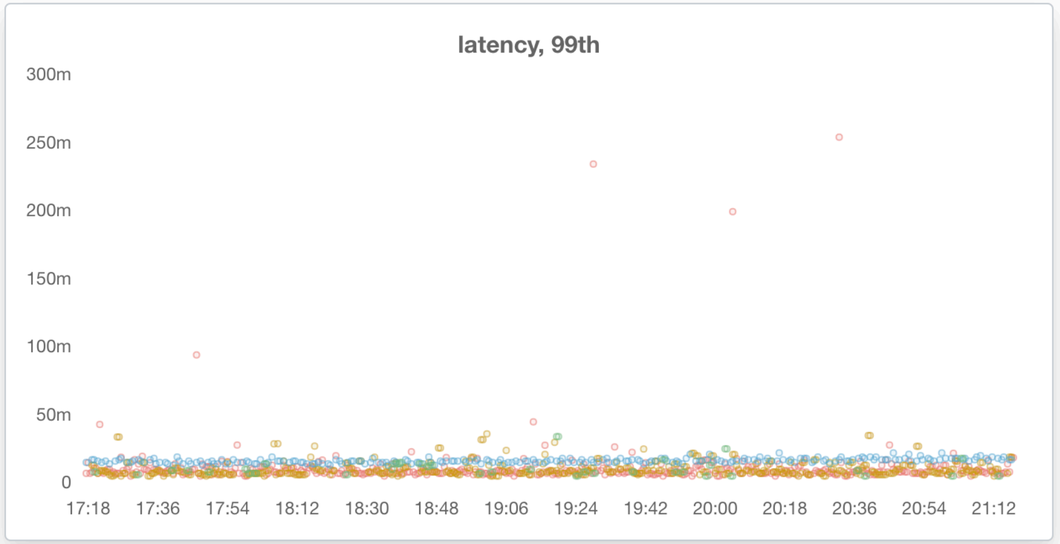


notes **low** prod deploy rollback

Slack Stash Sentry Kubernetes Grafana Teamcity Swagger

Основная информация | Troubleshooting | Библиотеки | Секреты | Настройки





Issues from sentry

5 unresolved errors! unresolved

Issue	Graph 24h 14d	Events	Users
*errors.errorString: cannot find notes Первое срабатывание: 23 апр. 08:58		714912	57
*errors.errorString: problem while iterating rows Первое срабатывание: 24 апр. 13:26		12446	33
*errors.errorString: cannot save notes Первое срабатывание: 23 апр. 12:20		5177	53

PaaS Dashboard 😎

PaaS Dashboard 😎

1. Как найти ответственного за сервис?

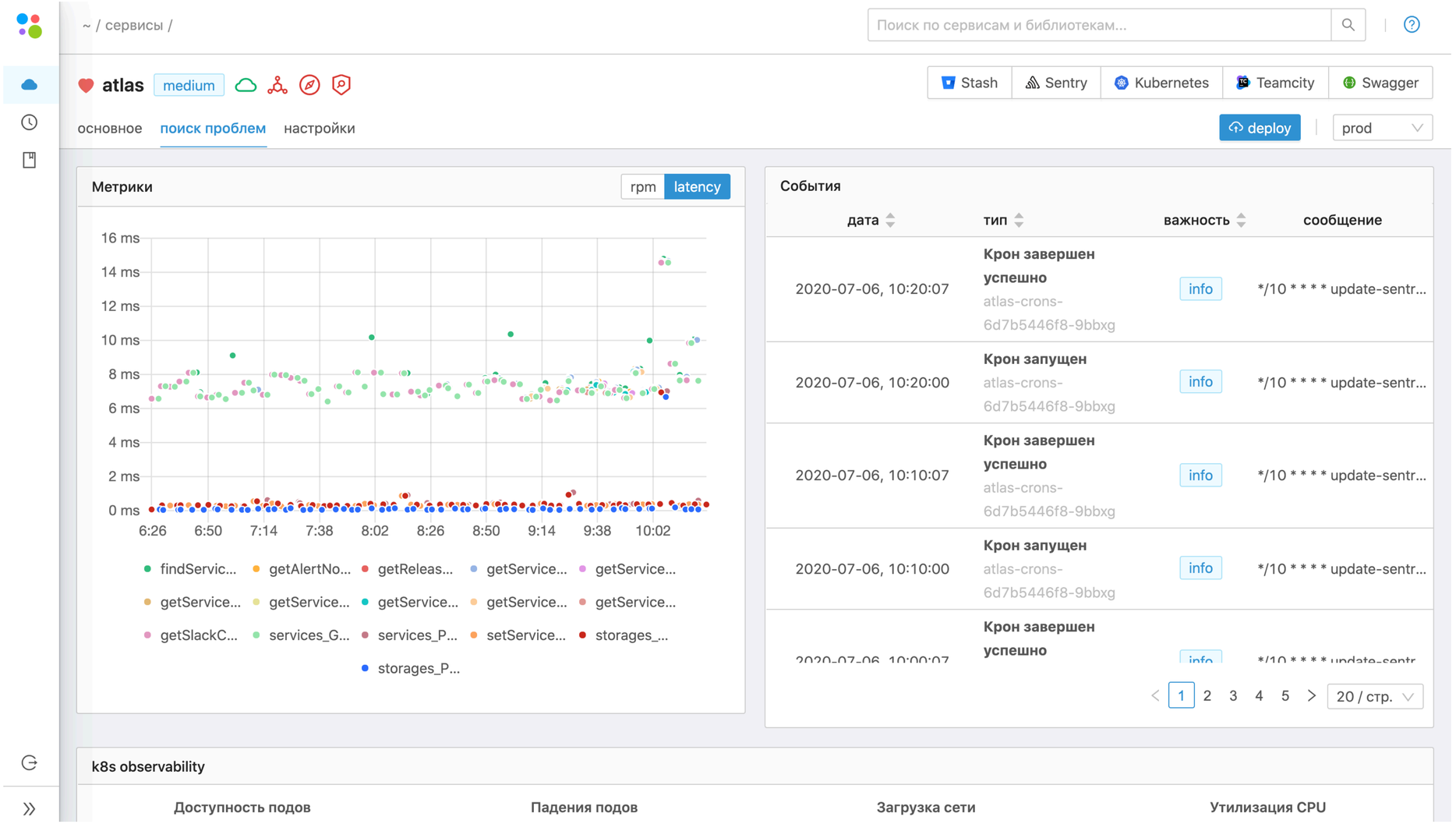
PaaS Dashboard 😎

1. Как найти ответственного за сервис?
2. Почему сервис деградирует?

PaaS Dashboard 😎

1. Как найти ответственного за сервис?
2. Почему сервис деградирует?
3. Как сейчас чувствует себя система в целом?

PaaS Dashboard 😎



PaaS недостатки 😕

PaaS недостатки 😕

1. Миграция существующих сервисов может быть долгой (очень)

PaaS недостатки 😕

1. Миграция существующих сервисов может быть долгой (очень)
2. Те разработчики, которые уже пользуются низкоуровневыми инструментами и имеют с ними опыт, могут быть против миграции

PaaS недостатки 😕

1. Миграция существующих сервисов может быть долгой (очень)
2. Те разработчики, которые уже пользуются низкоуровневыми инструментами и имеют с ними опыт, могут быть против миграции
3. Нужно находить правильный баланс между автоматизацией и гибкостью

PaaS преимущества 😎

PaaS преимущества 😎

1. Экономия времени и ресурсов со стороны продукта

PaaS преимущества 😎

1. Экономия времени и ресурсов со стороны продукта
2. "Зоопарк" под контролем

PaaS преимущества 😎

1. Экономия времени и ресурсов со стороны продукта
2. "Зоопарк" под контролем
3. Возможность со стороны платформы проводить любые операции без необходимости влезать в рабочие спринты продуктовых команд

Итоги

Итоги

1. PaaS необходим при масштабировании команды разработки

Итоги

1. PaaS необходим при масштабировании команды разработки
2. Количество времени на интеграцию с инфраструктурой можно снизить практически до нуля

Итоги

1. PaaS необходим при масштабировании команды разработки
2. Количество времени на интеграцию с инфраструктурой можно снизить практически до нуля
3. Если рассматривать PaaS как продукт, можно увидеть и решить реальные проблемы пользователей и дать возможность расти бизнесу с большей скоростью

Спасибо

tg: @lookyan
fb.com/alex.lukyanchenko

Лукьянченко Александр
Team Lead, Architecture

Москва, 2020

Avito