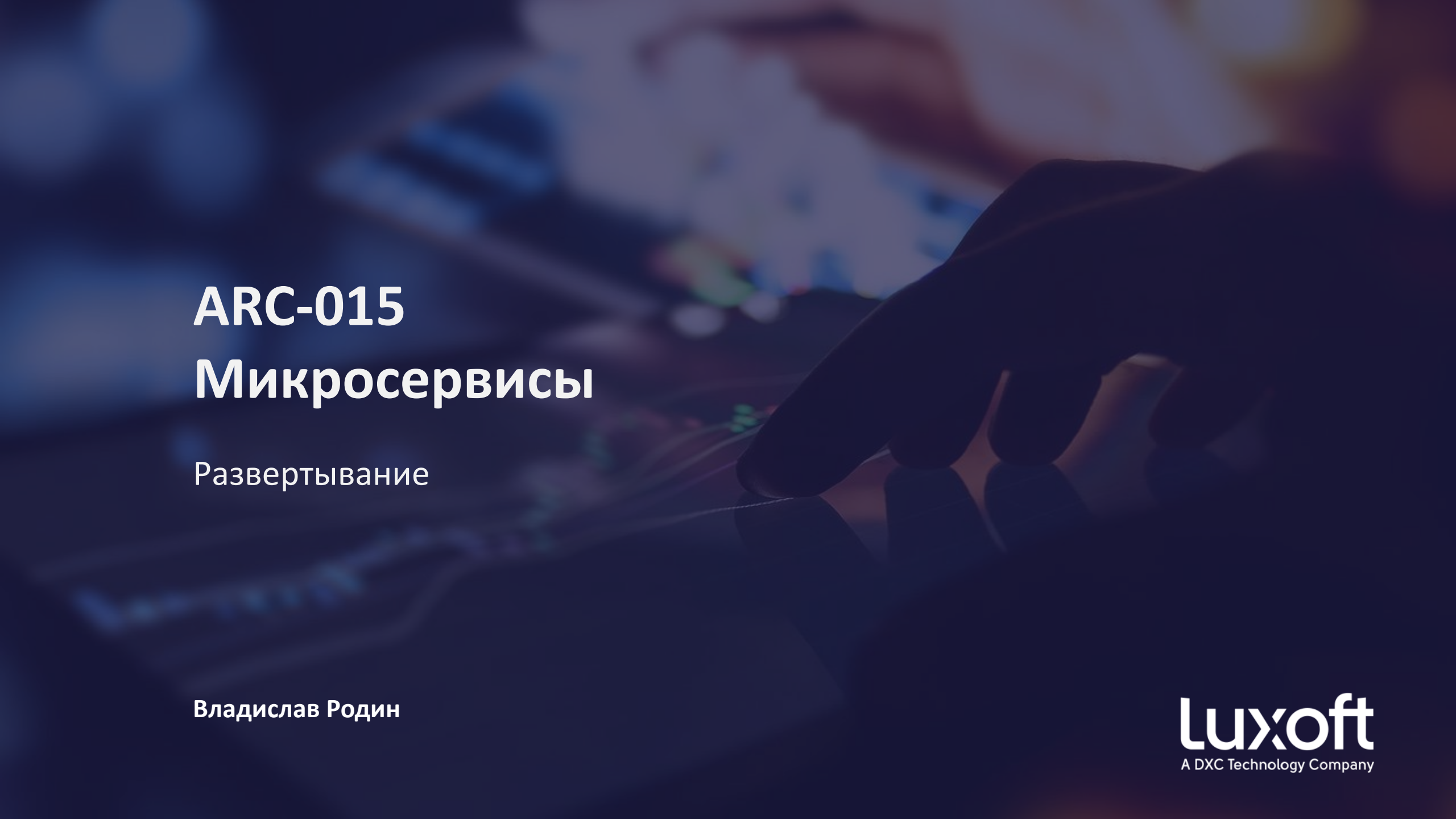


ARC-015
Микросервисы

Развертывание

Владислав Родин

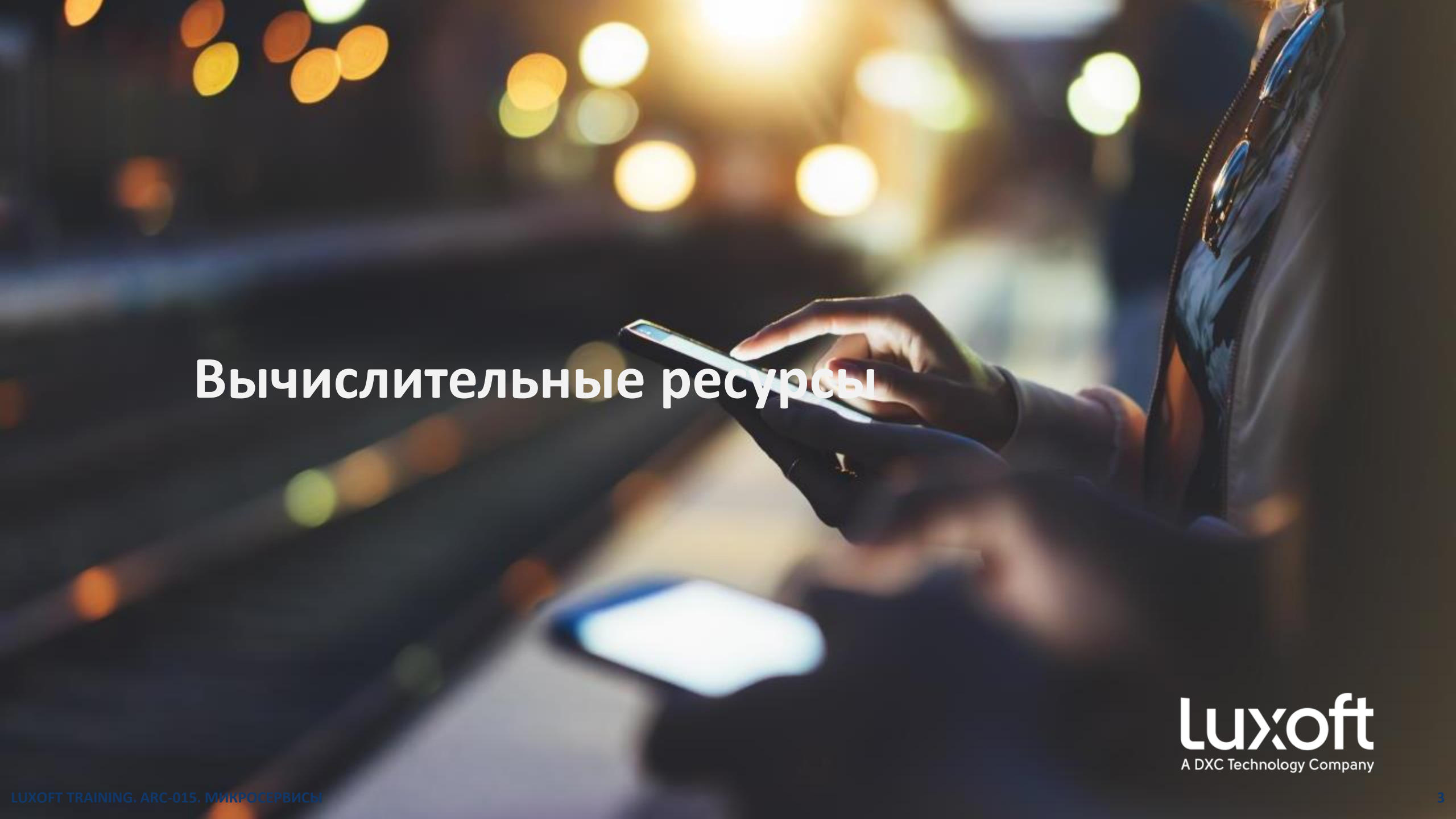
Luxoft
A DXC Technology Company

Развертывание

- ***Развертывание*** **— это сочетание двух взаимосвязанных концепций — процесса и архитектуры.**
  - ***Процесс*** **развертывания заключается в доставке кода в промышленную среду и состоит из этапов, которые должны выполнить люди — разработчики или системные администраторы.**
  - ***Архитектура*** **развертывания определяет структуру среды, в которой этот код будет выполняться.**

Решаемые вопросы

- **Какие вычислительные ресурсы используем**
  - ***Выделенные сервера***
  - ***Виртуальные машины***
  - ***Контейнера***
  - ***Не используем вычислительные ресурсы (serverless)***
- ***Как управляем вычислительными ресурсами (платформа)***
  - ***Платформы как службы (PaaS)***
  - ***Оркестровка***
- ***Как организуем процесс развертывания (стратегия и реализация)***



Вычислительные ресурсы

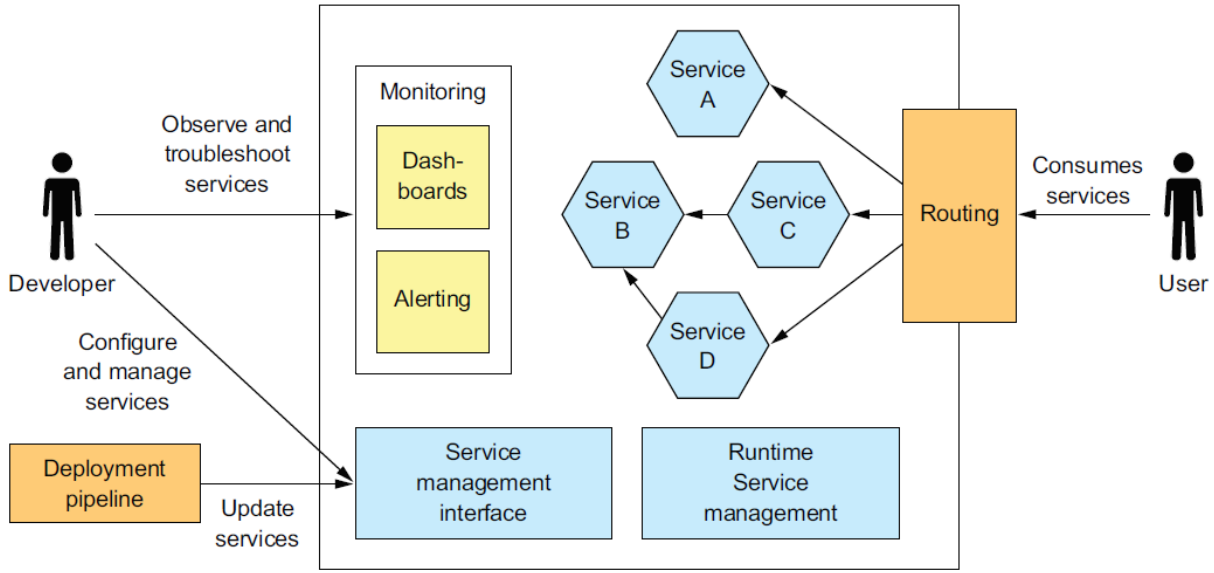
Вычислительные ресурсы

- ***Выделенные сервера***
- ***Виртуальные машины***
  - **Запущенные в высокоавтоматизированных облаках**
- ***Контейнеры*** **— легковесный слой абстракции поверх виртуальных машин**
- ***Бессерверные платформы (serverless)*** **развертывания (AWS Lambda)**

Промышленная среда (Production environment)

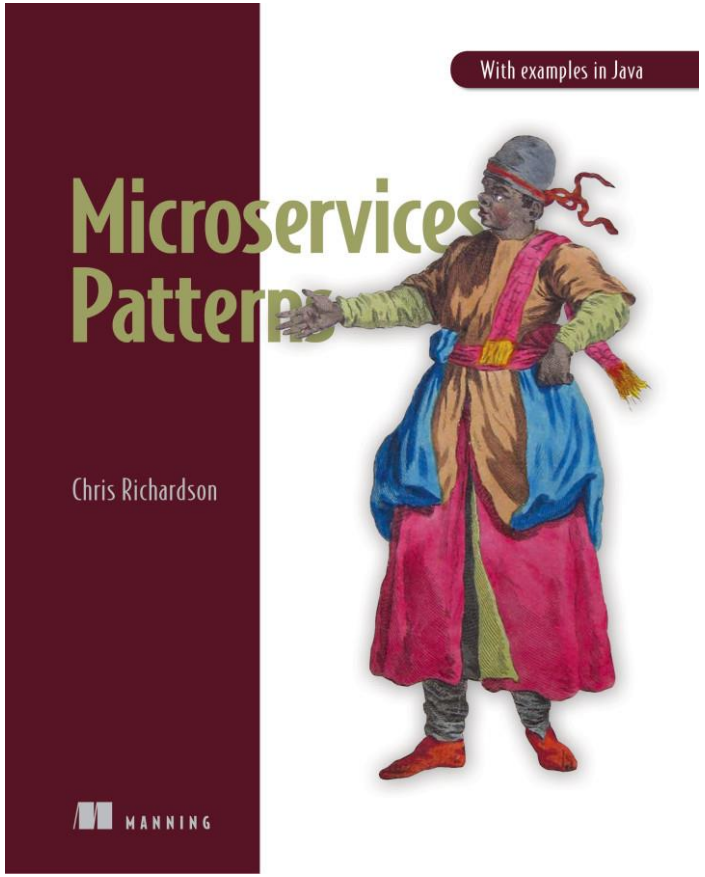
Должна поддерживать следующие возможности:

- **Интерфейс управления сервисами**
  позволяет разработчикам создавать, обновлять и конфигурировать сервисы.
- **Управление запущенными сервисами**
  пытается следить за тем, чтобы в промышленной среде всегда выполнялось желаемое количество экземпляров сервиса.
- **Мониторинг**
  предоставляет разработчикам сведения о работе их сервисов, включая журнальные файлы и показатели.
- **Маршрутизация**
  направляет запросы от пользователей к сервисам.



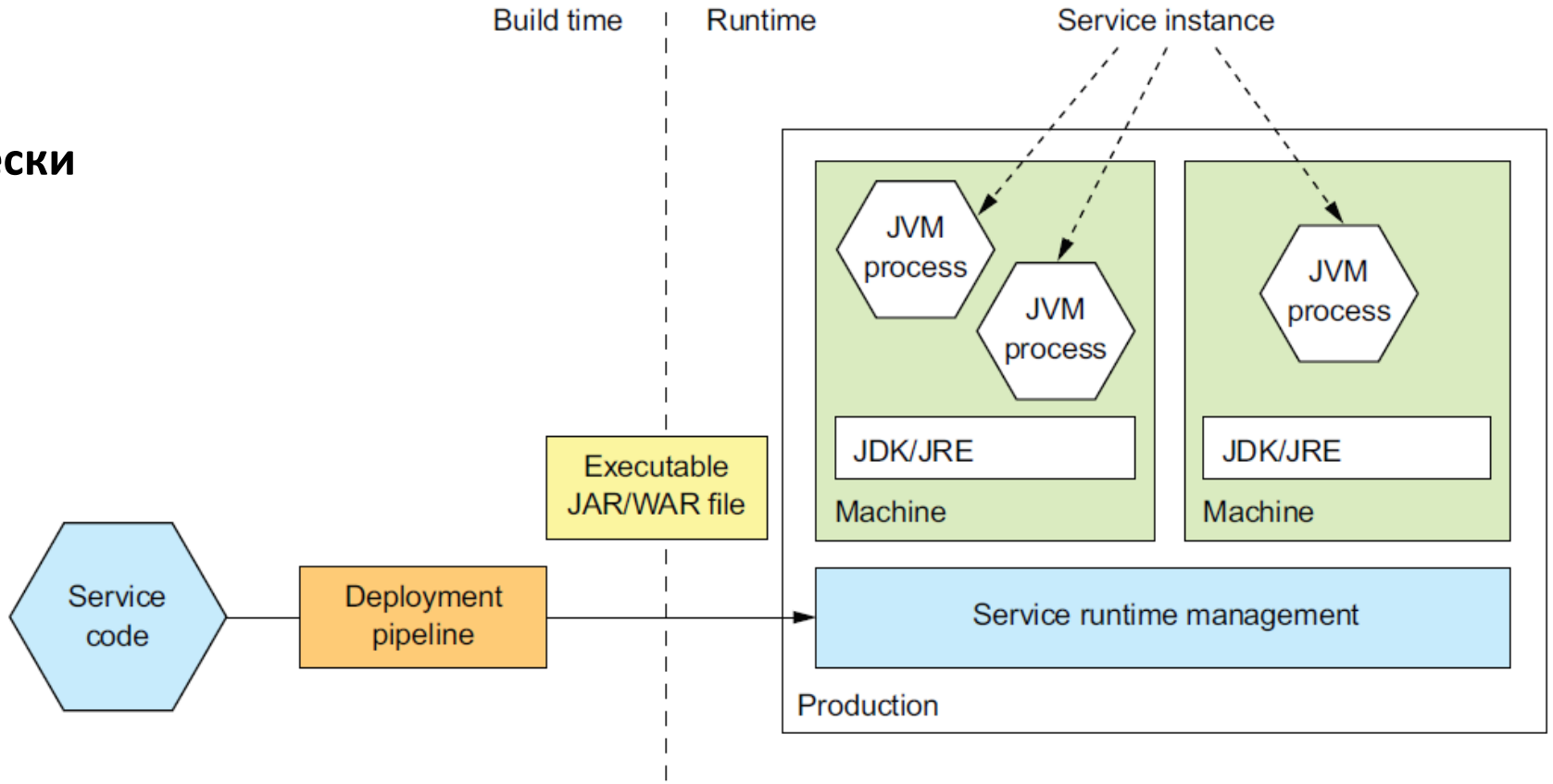
Паттерны развертывания
Крис Ричардсон

- Развертывание сервисов с помощью пакетов для отдельных языков
  (Deploying services using the Language-specific packaging format pattern)
- Развертывание сервисов в виде виртуальных машин
  (Service instance per VM)
- Развертывание сервисов в виде контейнеров
  (Service instance per Container)
- Бессерверное развертывание сервисов (Serverless deployment)
- Платформа развертывания сервисов (Service deployment platform)



Развертывание сервисов с помощью пакетов для отдельных языков

- Устанавливается необходимая среда
- Доставляется пакет
  - В идеале – автоматически
- Пакет развертывается



Развертывание сервисов с помощью пакетов для отдельных языков

- **Варианты развертывания**
- **Несколько экземпляров на узел**

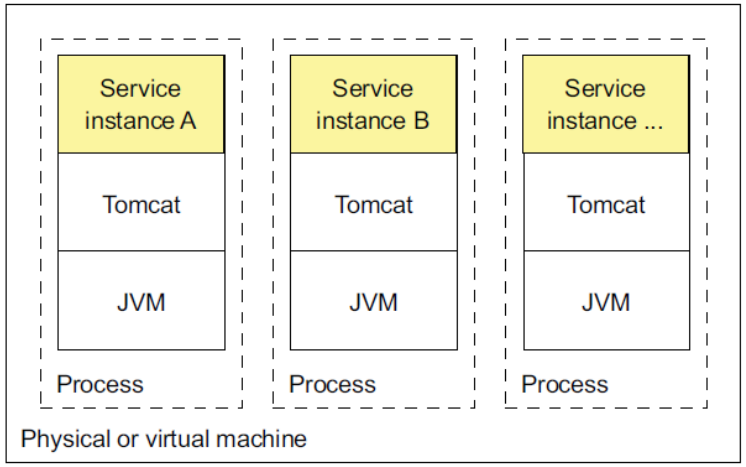
  (Multiple service instances per host)
- **Один экземпляр на узел**

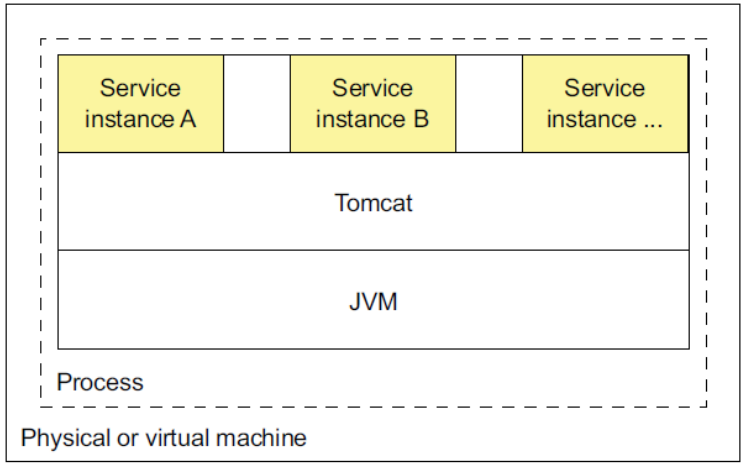
  (Service instance per host)

Несколько экземпляров на узел (Multiple service instances per host)

Запустите несколько экземпляров различных служб на узле (физическая или виртуальная машина).

- **Способы:**
  - **Разверните каждый экземпляр службы как процесс.**
  - **Разверните несколько экземпляров службы в одном процессе.**





Несколько экземпляров на узел: Результат

- **Преимущества:**
  - **Более эффективное использование ресурсов по сравнению с вариантом «один экземпляр на узел»**
- **Недостатки:**
  - **Риск противоречивых потребностей в ресурсах**
  - **Риск конфликта версий зависимости**
  - **Сложно ограничить ресурсы, потребляемые экземпляром службы**
  - **Если несколько экземпляров служб развертываются в одном и том же процессе, то сложно отслеживать потребление ресурсов каждым экземпляром службы**
  - **Невозможно изолировать каждый экземпляр**

Один экземпляр на машину (Service instance per host)

Развернуть каждый экземпляр службы на своем собственном хосте

- **Преимущества:**
  - **Экземпляры сервисов изолированы друг от друга**
  - **Нет возможности противоречивых требований к ресурсам или версий зависимостей**
  - **Экземпляр службы может использовать только ресурсы одного хоста**
  - **Просто для мониторинга, управления и повторного развертывания каждого экземпляра службы**
- **Недостатки:**
  - **менее эффективное использование ресурсов по сравнению с несколькими сервисами на узел**

Развертывание сервисов с помощью пакетов для отдельных языков: Результат

- **Преимущества:**
  - **Быстрое развертывание**

    зачастую требуется только копирование пакета
  - **Эффективное задействование ресурсов, особенно при запуске нескольких экземпляров на одном компьютере или внутри одного процесса.**
- **Недостатки:**
  - **Отсутствие инкапсуляции стека технологий**

Привязка портов (Port binding)

Экспортируйте сервисы через привязку портов

- **Иногда веб-приложения запускают внутри контейнера веб-сервера. Например, PHP-приложение может быть запущено как модуль внутри Apache HTTPD или Java-приложение может быть запущено внутри Tomcat.**
- **Приложение двенадцати факторов является полностью самодостаточным и не полагается на инъекцию веб-сервера во время выполнения для того, чтобы создать веб-сервис.**

  Веб-приложение экспортирует HTTP-сервис путём привязки к порту и прослушивает запросы, поступающих на этот порт.

The twelve factor app

Зависимости в MSA

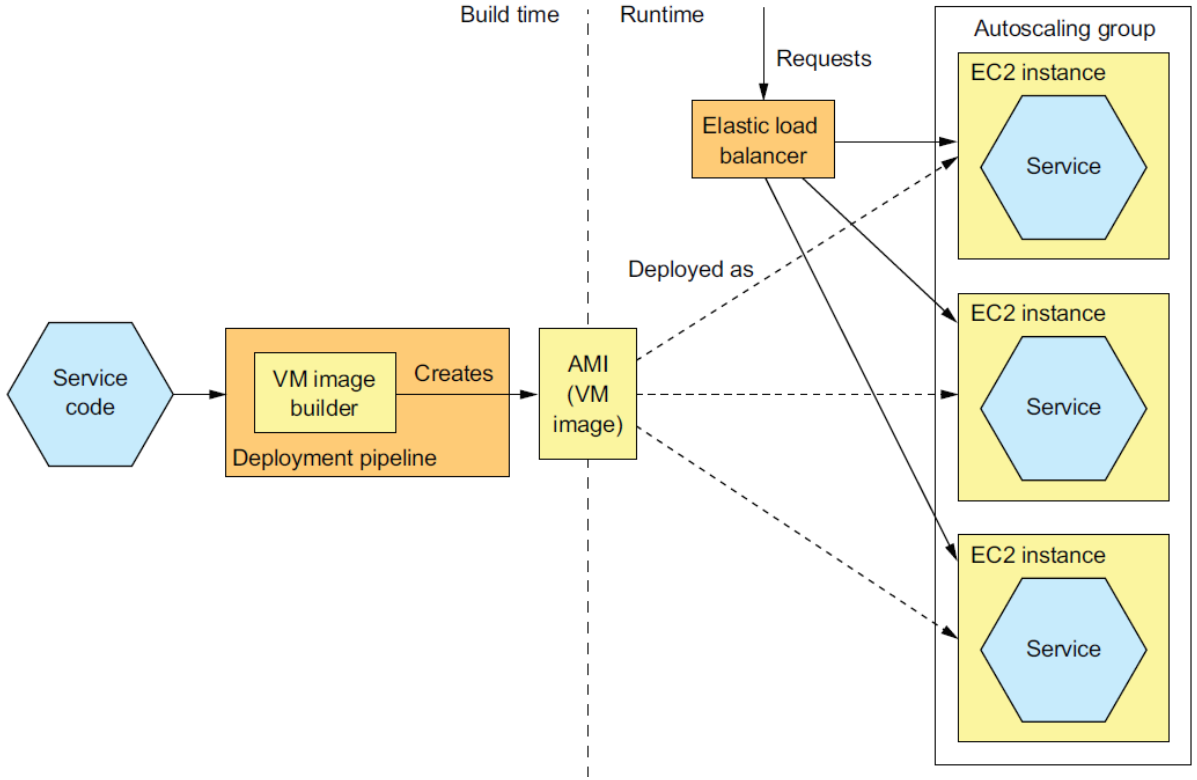
- **Явно объявляйте и изолируйте зависимости**

  Приложение двенадцати факторов никогда не зависит от неявно существующих, доступных всей системе пакетов.

The twelve factor app

Развертывание сервисов в виде виртуальных машин

- Развертывает сервисы упакованные в виде образов виртуальных машин
- Примеры утилит сборки
  - Aminator от Netflix (AWS)
  - Packer (EC2, Digital Ocean, Virtual Box, VMware)
  - Elastic Beanstalk (AWS)



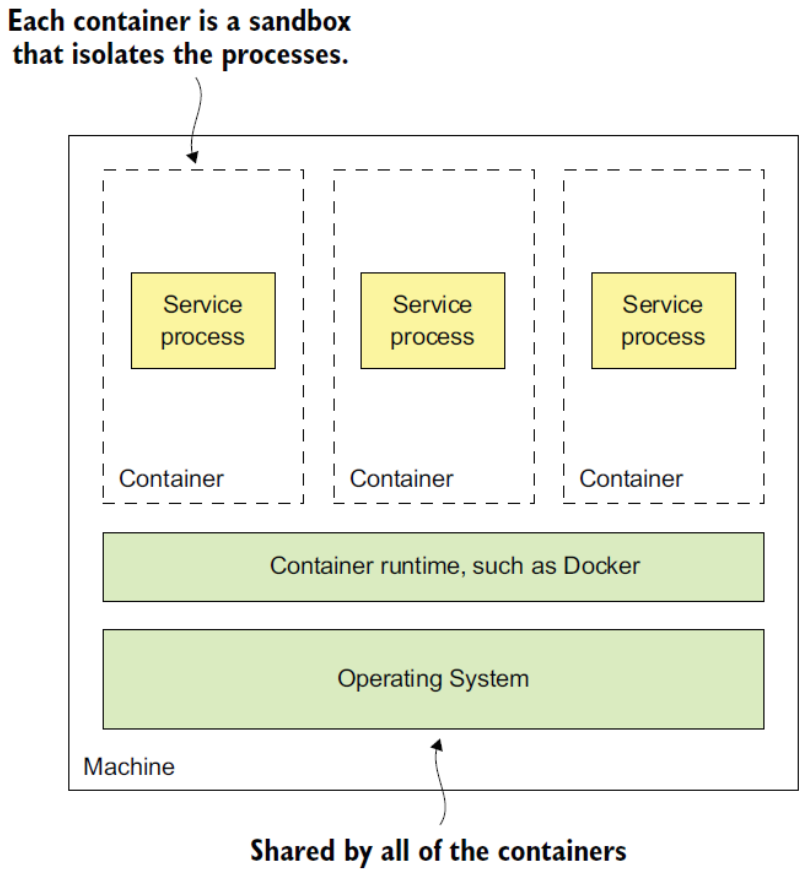
Развертывание сервисов в виде виртуальных машин: Результат

- **Преимущества:**
  - **Образ ВМ инкапсулирует стек технологий.**

    избавляет от необходимости устанавливать и конфигурировать ПО
  - **Экземпляры сервиса изолированы.**
  - **Используется зрелая облачная инфраструктура.**
- **Недостатки:**
  - **Менее эффективно используются ресурсы.**
  - **Развертывание протекает довольно медленно.**
  - **Требуются дополнительные расходы на системное администрирование.**

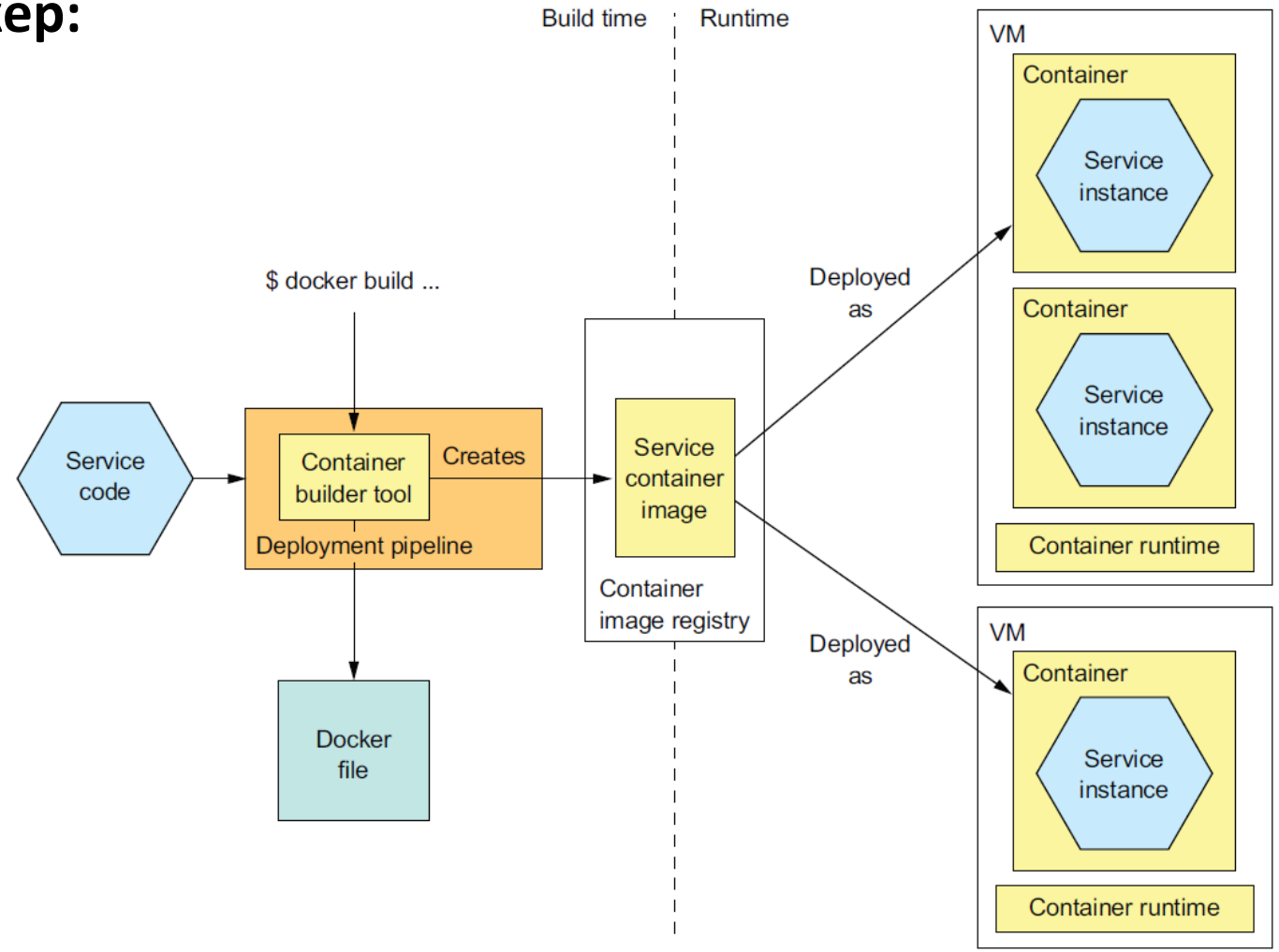
Развертывание сервисов в виде контейнеров

- Контейнеры — это более легковесный современный механизм развертывания.
- Используют механизм виртуализации на уровне операционной системы.
- Контейнер обычно состоит из одного процесса (хотя их может быть несколько), запущенного в среде, изолированной от других контейнеров



Развертывание сервисов в виде контейнеров: Процесс

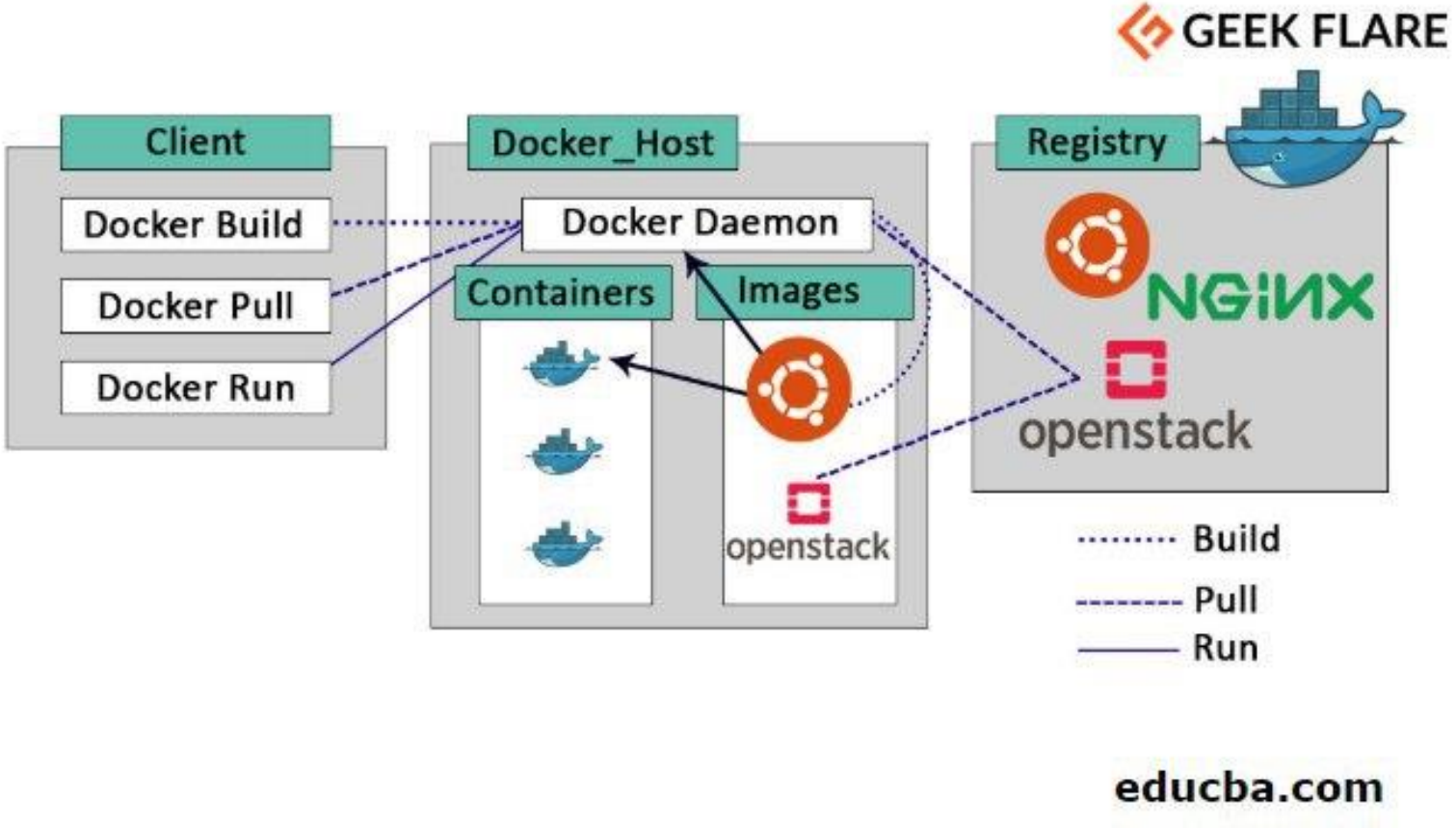
- **Реализации оркестраторов докер:**
  - **Kubernetes**
  - **Marathon/Mesos**
  - **Amazon EC2 Container Service**



Развертывание сервисов в виде контейнеров: Докер

Dockerfile

```
FROM postgres:latest COPY
init.sql /docker-entrypoint-initdb.d/
```



Развертывание сервисов в виде контейнеров: Результат

- **Преимущества:**
  - **Инкапсуляция стека технологий, благодаря которой API для управления сервисом превращается в API контейнера**
  - **Экземпляры сервиса изолированы**
  - **Ресурсы экземпляров сервиса ограничены**
  - **Скорость развертывания**
- **Недостатки:**
  - **Вы должны постоянно заниматься управлением образами контейнеров и обновлением операционной системы вместе со средой выполнения**
  - **Если вы не применяете облачные контейнерные решения наподобие Google Container Engine или AWS ECS, на вас ложится администрирование контейнерной инфраструктуры и, возможно, инфраструктуры виртуальных машин, поверх которой она работает**

Бессерверное развертывание сервисов (Serverless deployment)

- **Проблемы предыдущих решений:**
  - **Стоимость:**
    - **Предыдущие шаблоны развертывания должны заранее выделять определенные вычислительные ресурсы — физические серверы, виртуальные машины или контейнеры.**
    - **Некоторые платформы развертывания поддерживают авто-масштабирование, динамически регулируя количество ВМ или контейнеров в зависимости от нагрузки.**
    - **Тем не менее вам постоянно нужно платить за какие-то ресурсы, даже если они простаивают.**
  - **Ответственность за системное администрирование ложится на вас**
- **Решение: использование инфраструктуры развертывания, которая скрывает любую концепцию серверов (то есть зарезервированных или предварительно выделенных ресурсов) – физических или виртуальных хостов или контейнеров.**

Бессерверное развертывание сервисов: Примеры

- AWS Lambda
- Google Cloud Functions
- Azure Functions

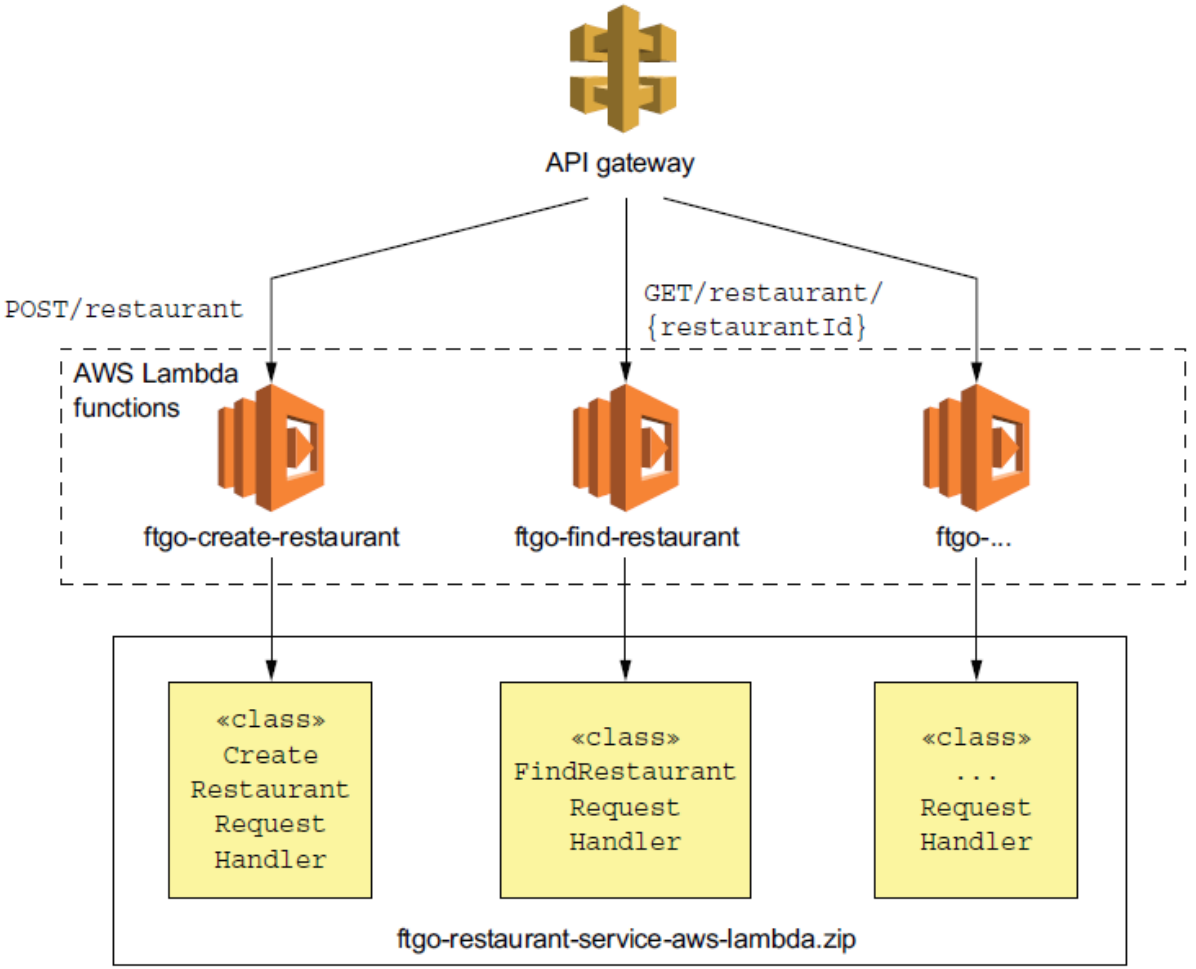
Бессерверное развертывание сервисов: Результат

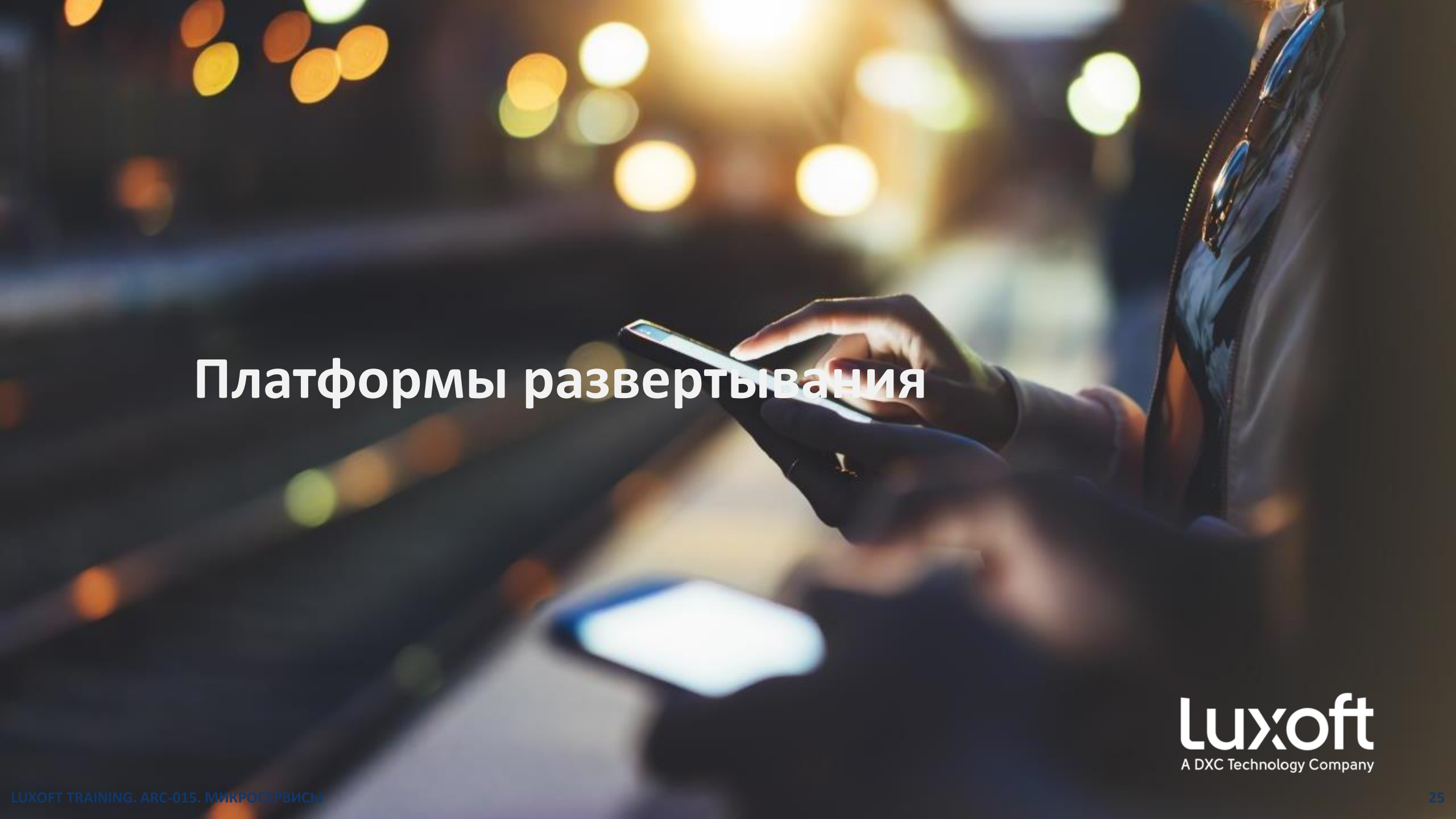
- **Преимущества:**
  - **Интеграция со многими облачными сервисами (AWS).**
  - **Избавление от многих задач системного администрирования.**

    Вы больше не отвечаете за низкоуровневое системное администрирование.
  - **Эластичность.**

    AWS Lambda запускает экземпляры вашего приложения в количестве, которого достаточно, чтобы справиться с нагрузкой.
  - **Тарифы, основанные на потреблении.**
- **Недостатки:**
  - **Периодически возникает высокая латентность**
  - **Ограниченная модель программирования, основанная на событиях/запросах**

Пример развертывание Rest сервиса на AWS Lambda





Платформы развертывания

luxoft
A DXC Technology Company

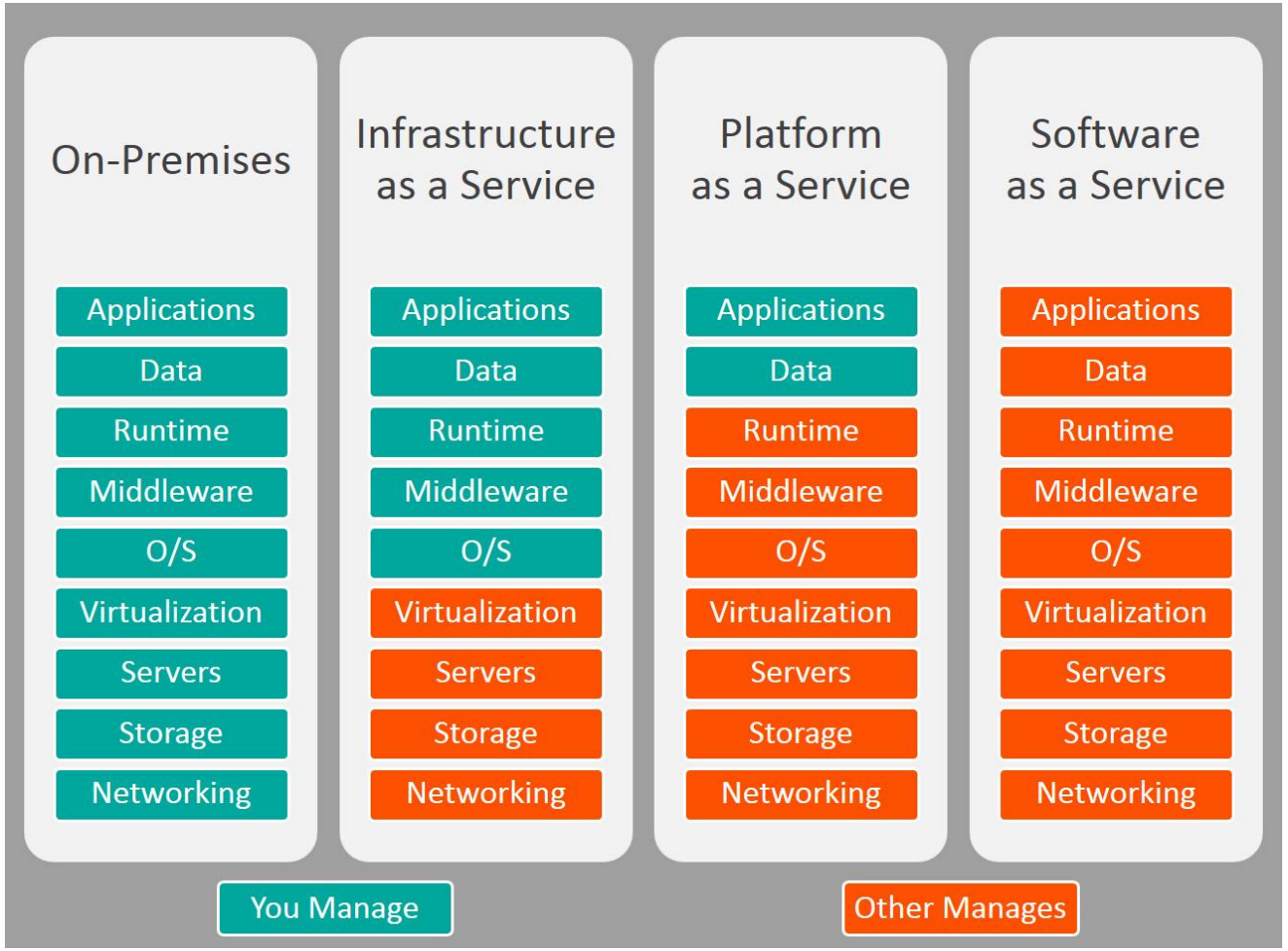
Платформа развертывания сервисов (Service deployment platform)

- **Используйте платформу развертывания, которая представляет собой автоматизированную инфраструктуру для развертывания приложений.**

  Она обеспечивает абстракцию, которая является именованным набором высокодоступных (например, с балансировкой нагрузки) экземпляров сервиса.

  - **Примеры:**
    - **Serverless platforms (AWS Lambda)**
    - **PaaS: (Cloud Foundry, AWS Elastic Beanstalk)**
    - **Docker orchestration frameworks:**
      - **Docker swarm mode**
      - **Kubernetes**

Облачные услуги
PaaS и другие облачные решения



Разделение PaaS по уровню доступа

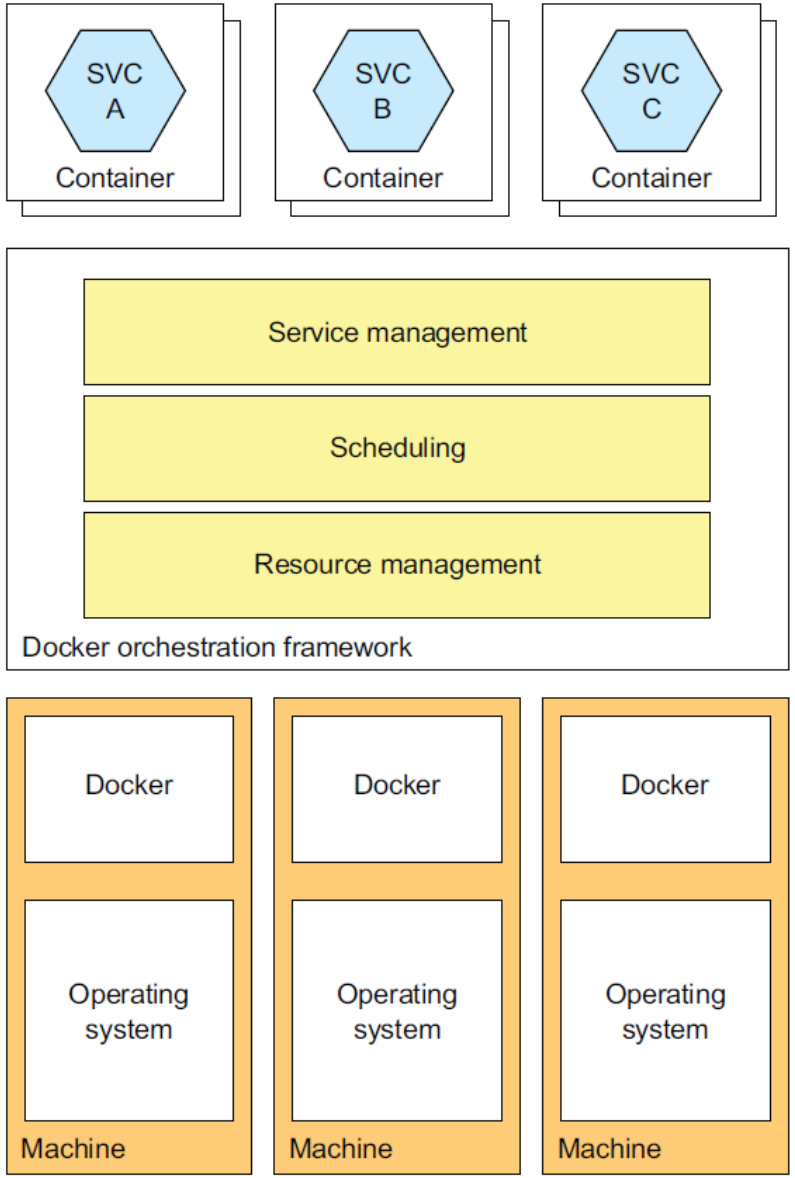
Публичные (public)	**Чаще всего используются для поддержки Open Source проектов Например Google App Engine**
Частные (private)	**Могут быть развернуты на оборудовании компании или в общедоступных облаках Например Pivotal Cloud Foundry**
Смешанные (hybrid)	**Набирают популярность в последнее время**

Kubernetes: Обзор

Фреймворк оркестрации Docker, который обращается с набором серверов под управлением Docker, как с пулом ресурсов.

- **Вы указываете, сколько экземпляров сервиса нужно запустить, а фреймворк делает все остальное.**
- **Функции:**
  - **Управление ресурсами.**
  - **Планирование.**
  - **Управление сервисом.**

kubernetes



Kubernetes: Архитектура

- ***Ведущий* узел**

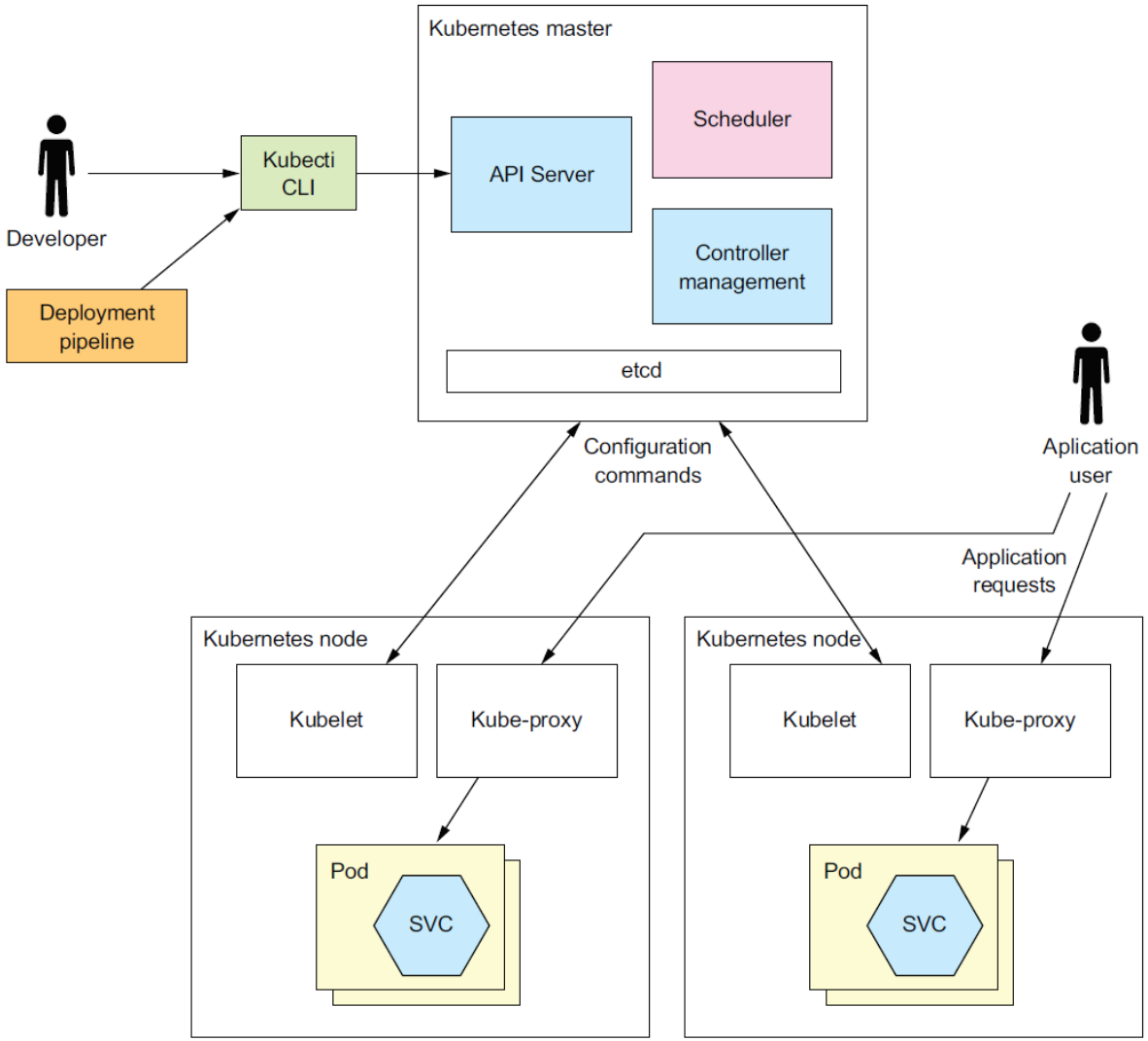
  отвечает за управление кластером.

  - **API-сервер — интерфейс REST API для развертывания и администрирования сервисов. Используется, утилитой командной строки kubectl;**
  - **Etcd — хранит данные кластера;**
  - **Планировщик — выбирает узел для запуска pod;**
  - **Диспетчер контроллеров**

- ***Рабочий узел***

  выполняет одну или несколько капсул (pod)

  - **Kubelet — создает капсулы pod и управляет их выполнением;**
  - **Kube-proxy — управляет сетевыми функциями, включая балансирование нагрузки между pod;**
  - **Капсулы pod — единицы развертывания в Kubernetes**



Kubernetes: Объекты (1 из 2)

- **Капсула (Pod) – базовая единица развертывания.**

  Состоит из одного или нескольких контейнеров с общими IP-адресом и томами хранения.

  - **Капсула экземпляра сервиса часто содержит лишь один контейнер, который, к примеру, выполняет JVM.**
  - **Но в некоторых случаях в ее состав может входить несколько дополнительных контейнеров, реализующих вспомогательные функции.**
    - **Например, у сервера NGINX может быть дополнительный контейнер, который периодически выполняет команду git pull, загружая последнюю версию веб-сайта.**
  - **Капсула является временной, поскольку ее контейнер или узел, на котором она выполняется, могут выйти из строя.**

- **Развертывание (Deployment) – декларативная спецификация капсулы.**
  - **Это контроллер, который постоянно обеспечивает нужное количество запущенных экземпляров капсул.**
  - **Для поддержки версионирования он использует плавающие обновления и откаты.**
  - **Каждый сервис в микросервисной архитектуре является развертыванием.**

Kubernetes: Объекты (2 из 2)

- Сервис (Service) – предоставляет клиентам сервиса статический/стабильный сетевой адрес.
  - Это разновидность механизма обнаружения сервисов на уровне инфраструктуры.
  - Сервис имеет IP-адрес и DNS-имя, которое на него указывает, TCP- и UDP-трафик распределяются между несколькими капсулами, если их больше одной.
  - IP-адрес и DNS-имя доступны только внутри Kubernetes.
- Конфигурация (ConfigMap) – именованный набор пар «имя — значение», который описывает внешнюю конфигурацию для одного или нескольких сервисов приложения.
  - Вы можете хранить конфиденциальную информацию, например пароли, в варианте ConfigMap под названием Secret.

Kubernetes: пример конфигурации

```
apiVersion: extensions/v1beta1
kind: Deployment
 metadata:
  name: ftgo-restaurant-service
 spec:
  replicas: 2
   template:
    metadata:
      labels:
        app: ftgo-restaurant-service
     spec:
       containers:
       - name: ftgo-restaurant-service
         image: msapatterns/ftgo-restaurant-service:latest
         imagePullPolicy: Always
         ports:
         - containerPort: 8080
           name: httpport
         env:
           - name: JAVA_OPTS
             value: "-Dsun.net.inetaddr.ttl=30"
           - name: SPRING_DATASOURCE_URL
             value: jdbc:mysql://ftgo-mysql/eventuate
           - name: SPRING DATASOURCE USERNAME
             valueFrom:
               secretKeyRef:
                 name: ftgo-db-secret
                 key: username
         - name: SPRING_DATASOURCE_PASSWORD
           valueFrom:
             secretKeyRef:
               name: ftgo-db-secret
               key: password
         - name: SPRING_DATASOURCE_DRIVER_CLASS_NAME
           value: com.mysql.jdbc.Driver
```

Specifies that this is an object of type Deployment

The name of the deployment

Number of pod replicas

Gives each pod a label called app whose value is ftgo-restaurant-service

The specification of the pod, which defines just one container

The container's port

The container's environment variables, which are read by Spring Boot

Sensitive values that are retrieved from the Kubernetes Secret called ftgo-db-secret

```
           - name: EVENTUATELOCAL_KAFKA_BOOTSTRAP_SERVERS
             value: ftgo-kafka:9092
           - name: EVENTUATELOCAL_ZOOKEEPER_CONNECTION_STRING
             value: ftgo-zookeeper:2181
       livenessProbe:
         httpGet:
           path: /actuator/health
           port: 8080
         initialDelaySeconds: 60
         periodSeconds: 20
       readinessProbe:
         httpGet:
           path: /actuator/health
           port: 8080
         initialDelaySeconds: 60
         periodSeconds: 20
```

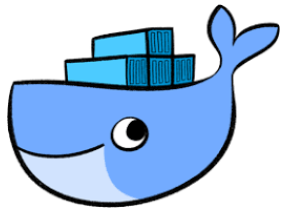
Configure Kubernetes to invoke the health check endpoint.

Развертывание без простоя

- Kubernetes превращает обновление запущенного сервиса в простой процесс, состоящий из трех шагов:

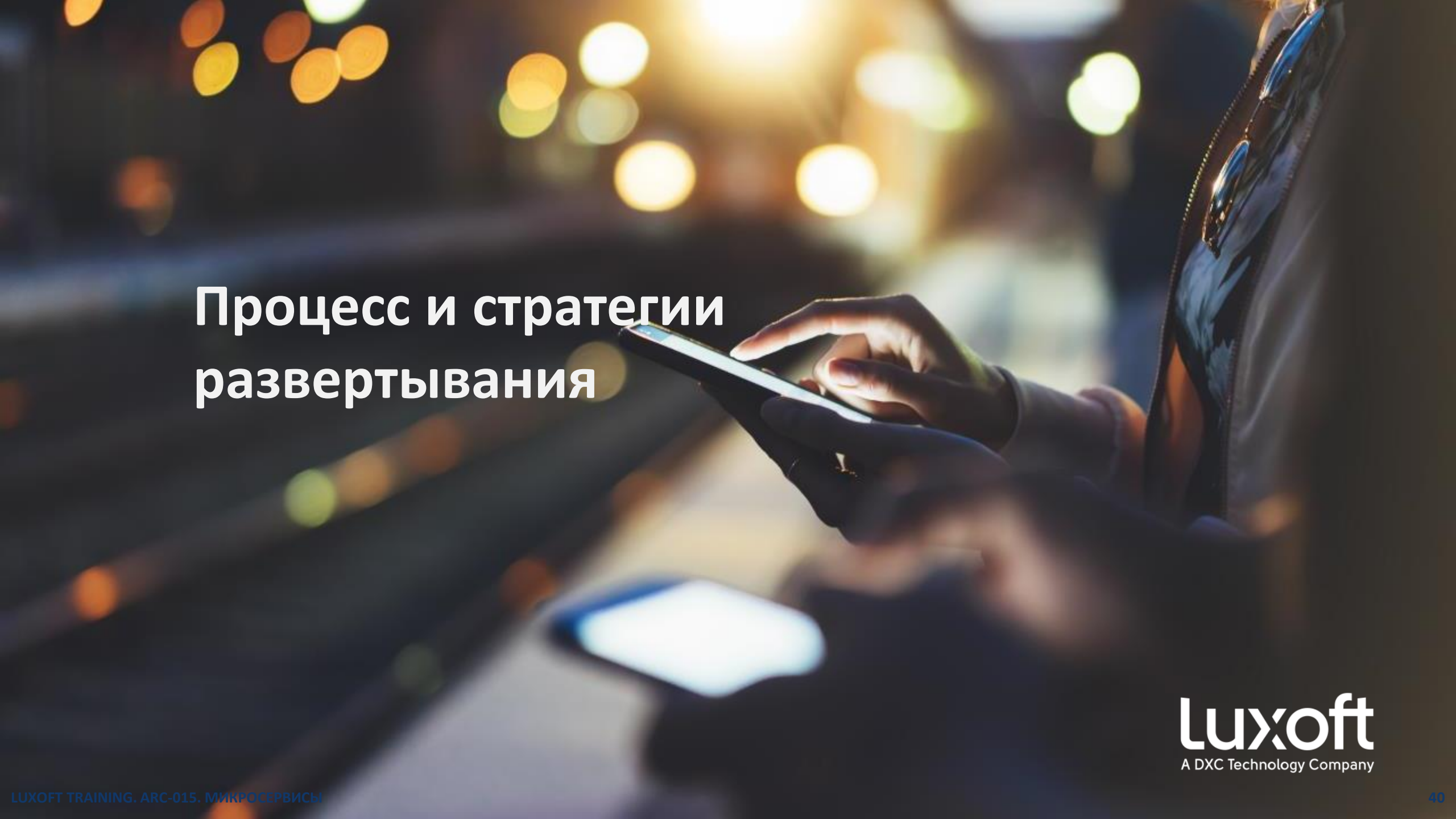
1. Сборка и загрузка в реестр нового докер-контейнера
   образ получит метку с другой версией
2. Редактирование YAML-файла с развертыванием сервиса таким образом, чтобы оно ссылалось на новый образ.
3. Обновление развертывания с помощью команды kubectl apply -f.

- Замечательной особенностью платформы Kubernetes является то, что она начинает удалять старые pod капсулы только тогда, когда их замены уже готовы к работе.
  Готовность определяется с помощью механизма проверки работоспособности *readinessProbe*
- Благодаря этому у вас всегда будут капсулы, готовые к обработке запросов.
- В итоге, если запуск капсул пройдет успешно, развертывание перейдет на новую версию.



PaaS vs CaaS

PaaS	CaaS
Развертывание и управление	Изоляция, множество команд
Одна единственная платформа	Множество платформ (на одном ядре ОС)
Ограничен набор доступных сервисов	Огромный выбор сервисов
Абстракция (buildpack) простота	Автоматизация (оркестровка) управление



Процесс и стратегии развертывания

luxoft
A DXC Technology Company

Стратегии развертывания

- Recreate – повторное создание
- Rolling (Ramped) – постепенный, «накатываемый» деплой
- Blue/Green – сине-зеленые развертывания
  - Анти-паттерн в MSA, легко приводящий к распредленному монолиту
- Canary – канареечные развертывания
  - Dark (скрытые) или A/B-развертывания

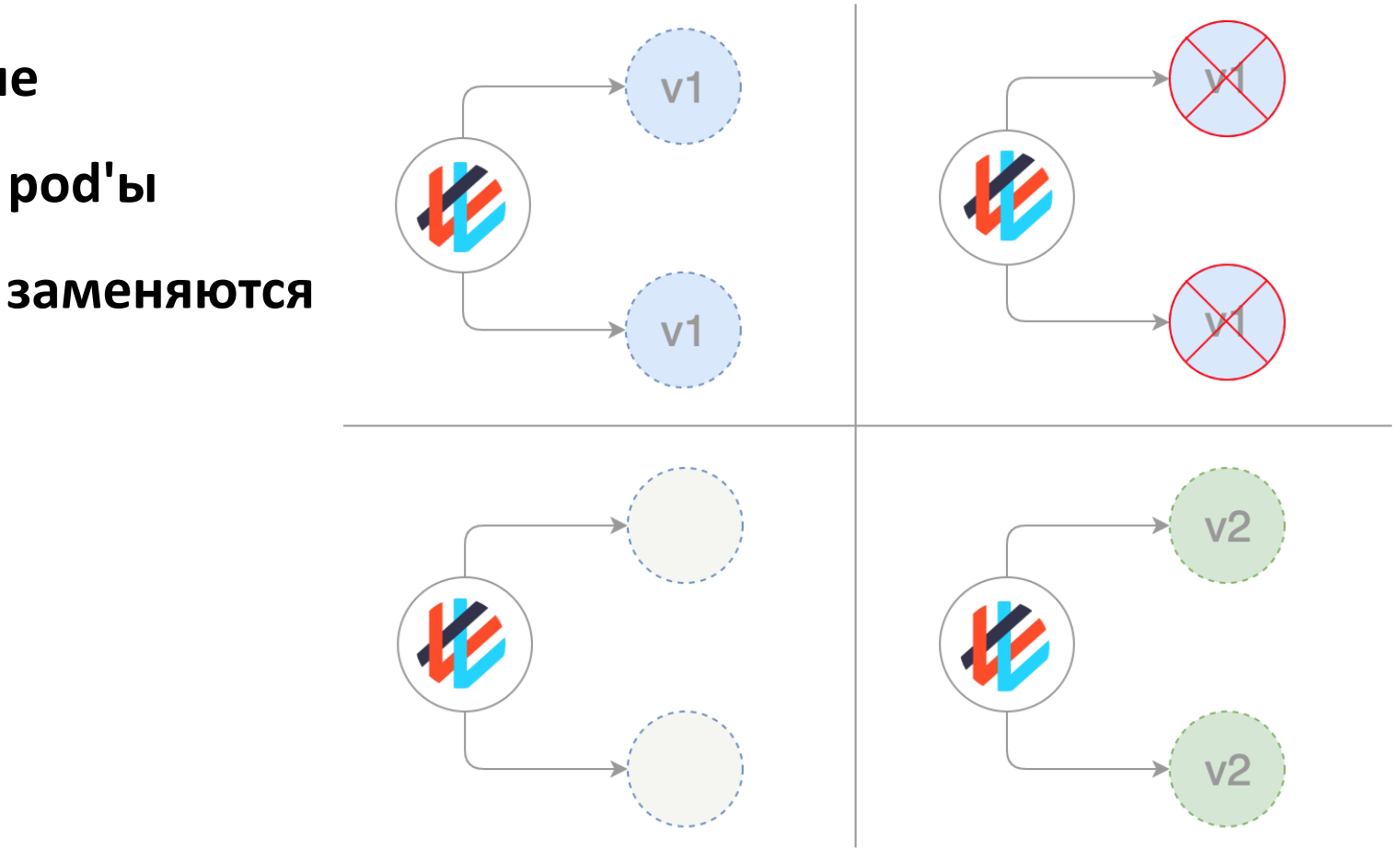
Принцип неизменяемой инфраструктуры (immutable infrastructure)

- **При каждом развертывании компоненты полностью заменяются, а не обновляются на месте.**

Один из принципов DevOps

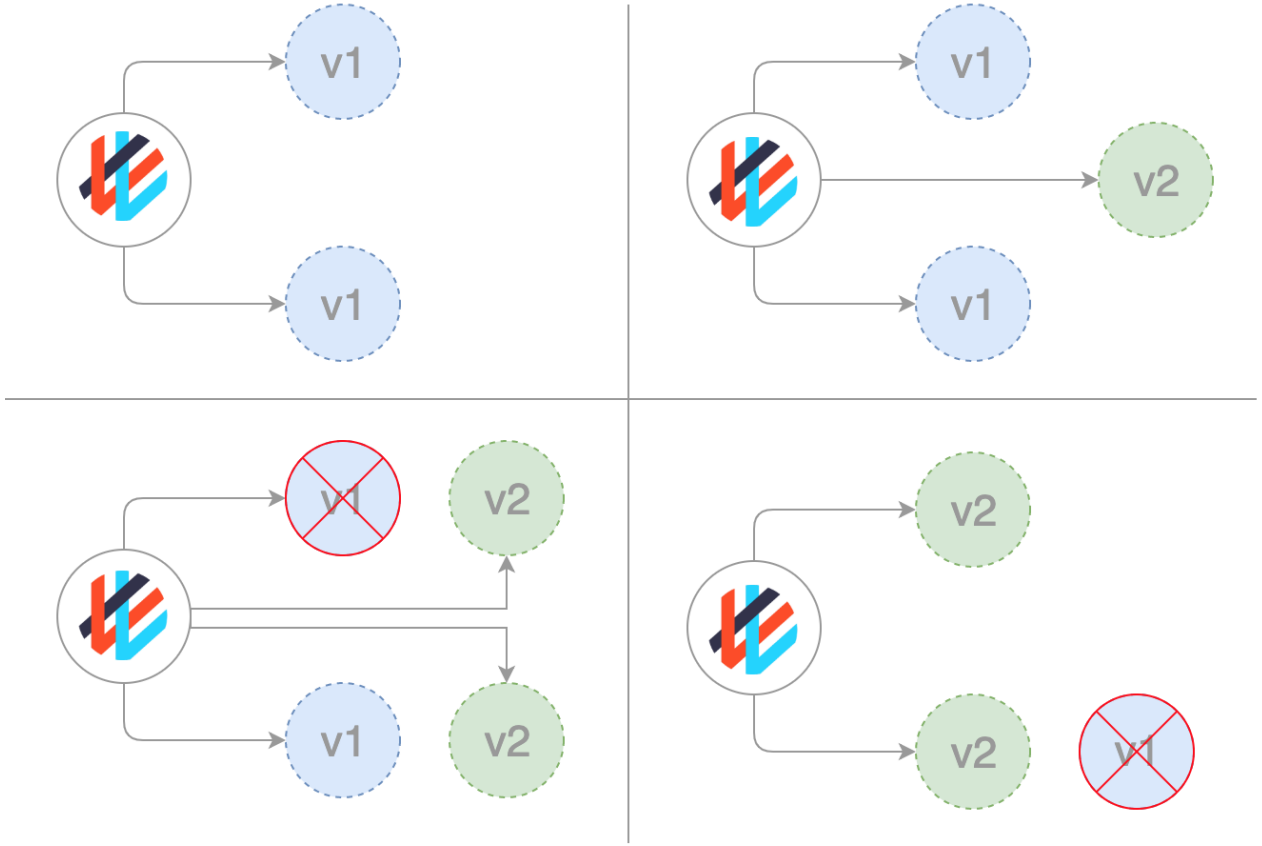
Recreate

- В этом простейшем типе развертывания старые pod'ы убиваются все разом и заменяются новыми



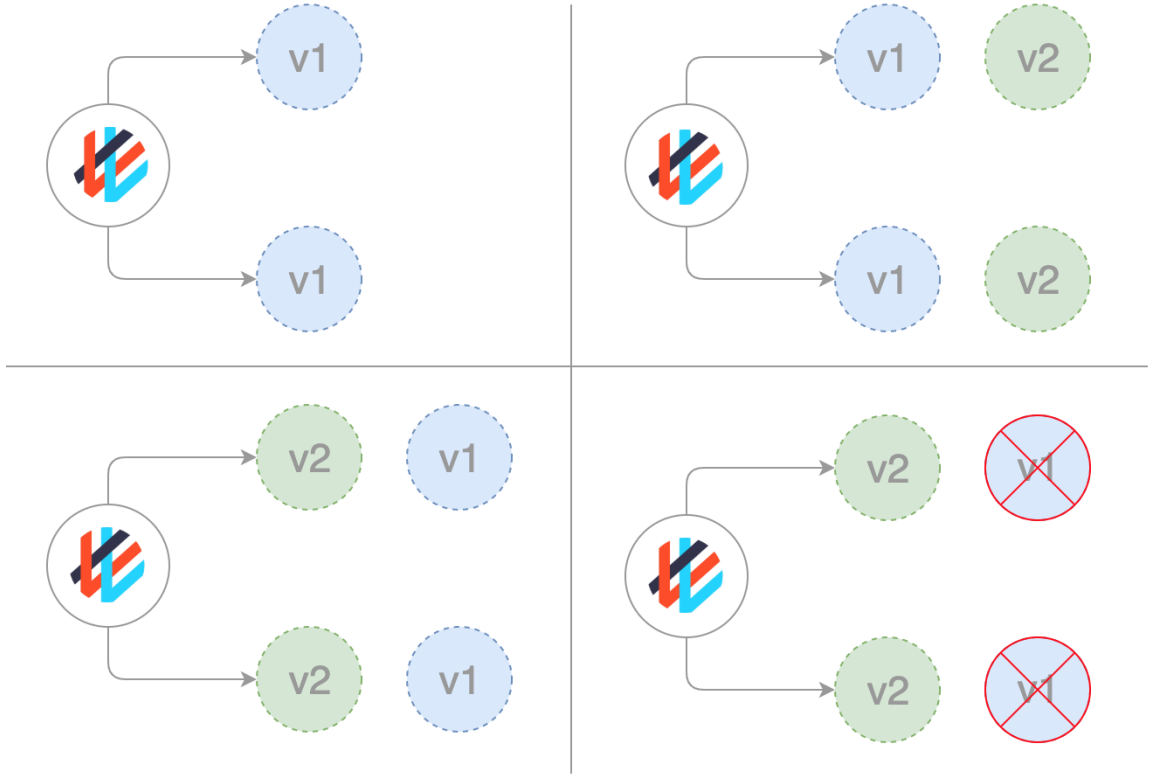
Rolling (Ramped)

- Это стандартная стратегия развертывания в Kubernetes. Она постепенно, один за другим, заменяет pod'ы со старой версией приложения на pod'ы с новой версией — без простоя кластера



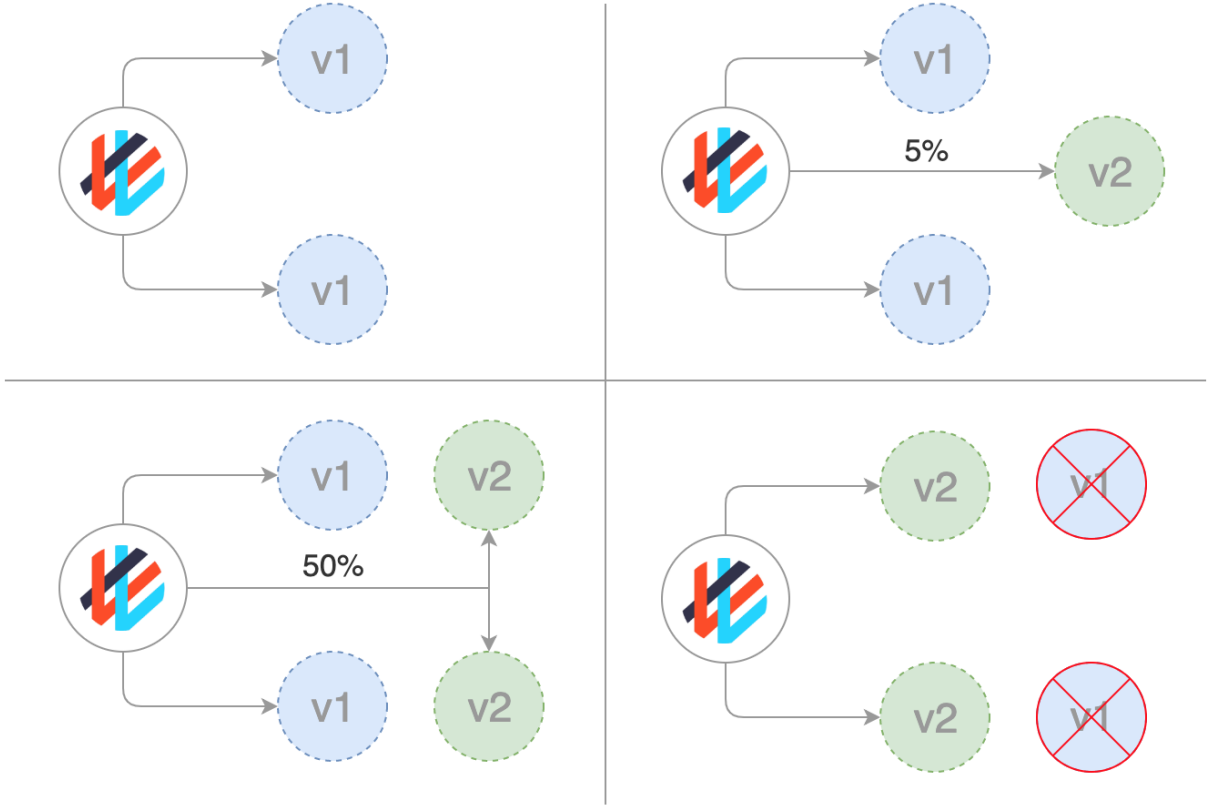
Blue/Green (red/black)

- Предусматривает одновременное развертывание старой (зеленой) и новой (синей) версий приложения.
- После размещения обеих версий обычные пользователи получают доступ к зеленой, в то время как синяя доступна для QA-команды для автоматизации тестов через отдельный сервис или прямой проброс портов
- После того, как синяя (новая) версия была протестирована и был одобрен ее релиз, сервис переключается на неё, а зеленая (старая) сворачивается



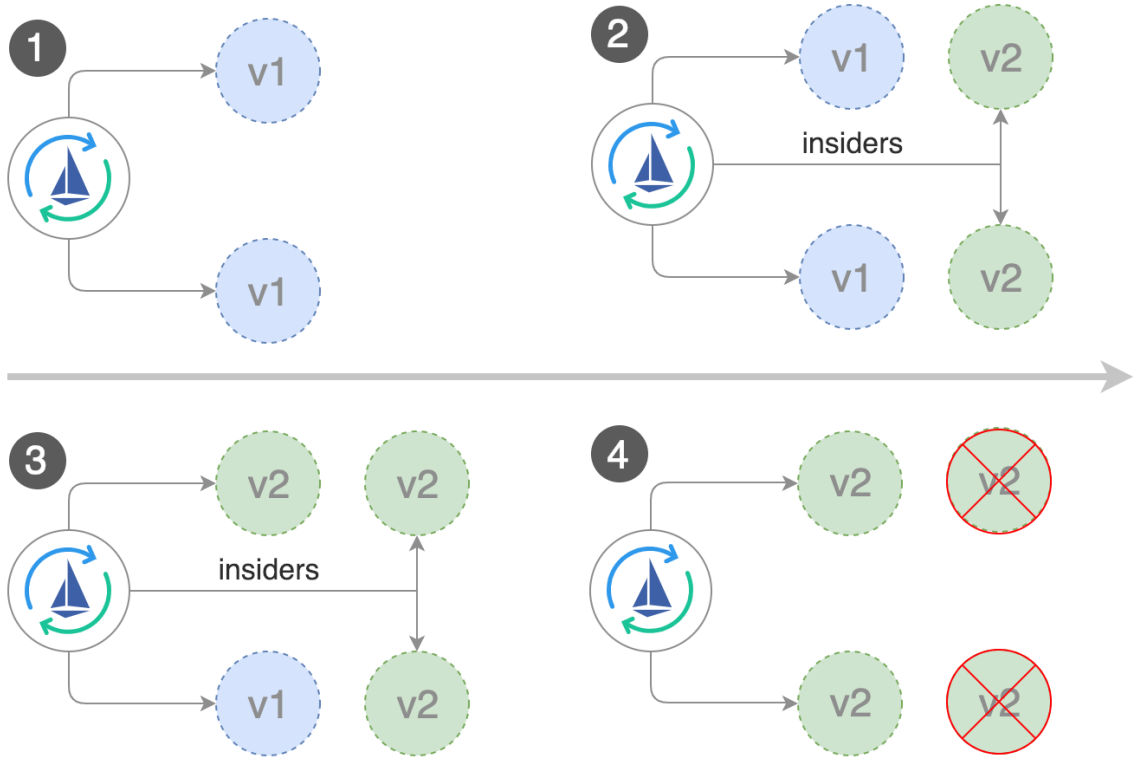
Canary (канареечные развертывания)

- Эта стратегия применяется, когда необходимо испытать некую новую функциональность, как правило, в бэкенде приложения.
- Суть подхода в том, чтобы создать два практически одинаковых сервера: один обслуживает почти всех пользователей, а другой, с новыми функциями, обслуживает лишь небольшую подгруппу пользователей, после чего результаты их работы сравниваются. Если все проходит без ошибок, новая версия постепенно выкатывается на всю инфраструктуру.



Dark (скрытые) или A/B-развертывания

- Скрытые развертывания имеют дело с фронтендом, а не с бэкендом, как канареечные.
- Вместо того, чтобы открыть доступ к новой функции всем пользователям, ее предлагают лишь ограниченной их части. Обычно эти пользователи не знают, что выступают тестерами-первопроходцами (отсюда и термин «скрытое развертывание»).

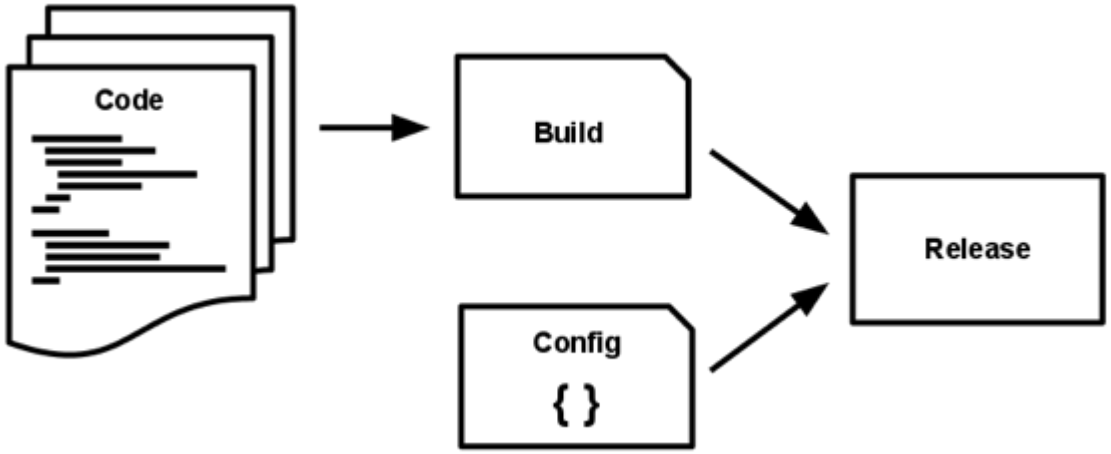


Сравнение стратегий развертывания

	Zero downtime	Real traffic testing	Targeted users
Recreate	нет	нет	нет
Rolling	да	нет	нет
Blue/Green	да	нет	нет
Canary	да	да	нет
A/B	да	да	да
Dark	да	да	нет

Сборка, релиз, выполнение

- Строго разделяйте стадии сборки и выполнения



The twelve-factor app

Развертывание без простоя: Проблемы

- Если в результате какой-то проблемы капсулы pod не запустятся, есть два варианта
  - Исправить сборку и повторить развертывание
  - Откатить развертывание

    Kubernetes хранит историю так называемых выкатываний (rollout).

    Каждый раз, когда вы обновляете развертывание, создается новое выкатывание.

    Благодаря этому вы можете легко откатить развертывание до предыдущей версии.
- Если ошибка выявлена уже после развертывания, то она не будет замечена Kubernetes
  - необходимо отделить *развертывание* (запуск сервиса в промышленной среде) от *выпуска* сервиса, в результате которого тот может начать обрабатывать пользовательский трафик

Отделение развертывания от выпуска

- *развертывание (Deployment)* — выполнение в промышленной среде
- *выпуск сервиса (Releasing a service)* — открытие доступа к нему конечным пользователям
- Шаги:
  1. Развертывание новой версии в промышленной среде без перенаправления к ней запросов конечных пользователей.
  2. Тестирование ее в реальных условиях.
  3. Выпуск сервиса для небольшого количества пользователей.
  4. Постепенный выпуск сервиса для все более широкой аудитории, пока он не станет обрабатывать весь промышленный трафик.
  5. Если на каком-либо этапе появится проблема, можно откатиться к старой версии.
     Если же вы уверены в том, что все работает как следует, старую версию можно удалить.
- Для отделения развертывания от выпуска сервиса удобнее всего использовать *сеть сервисов (service mesh)*

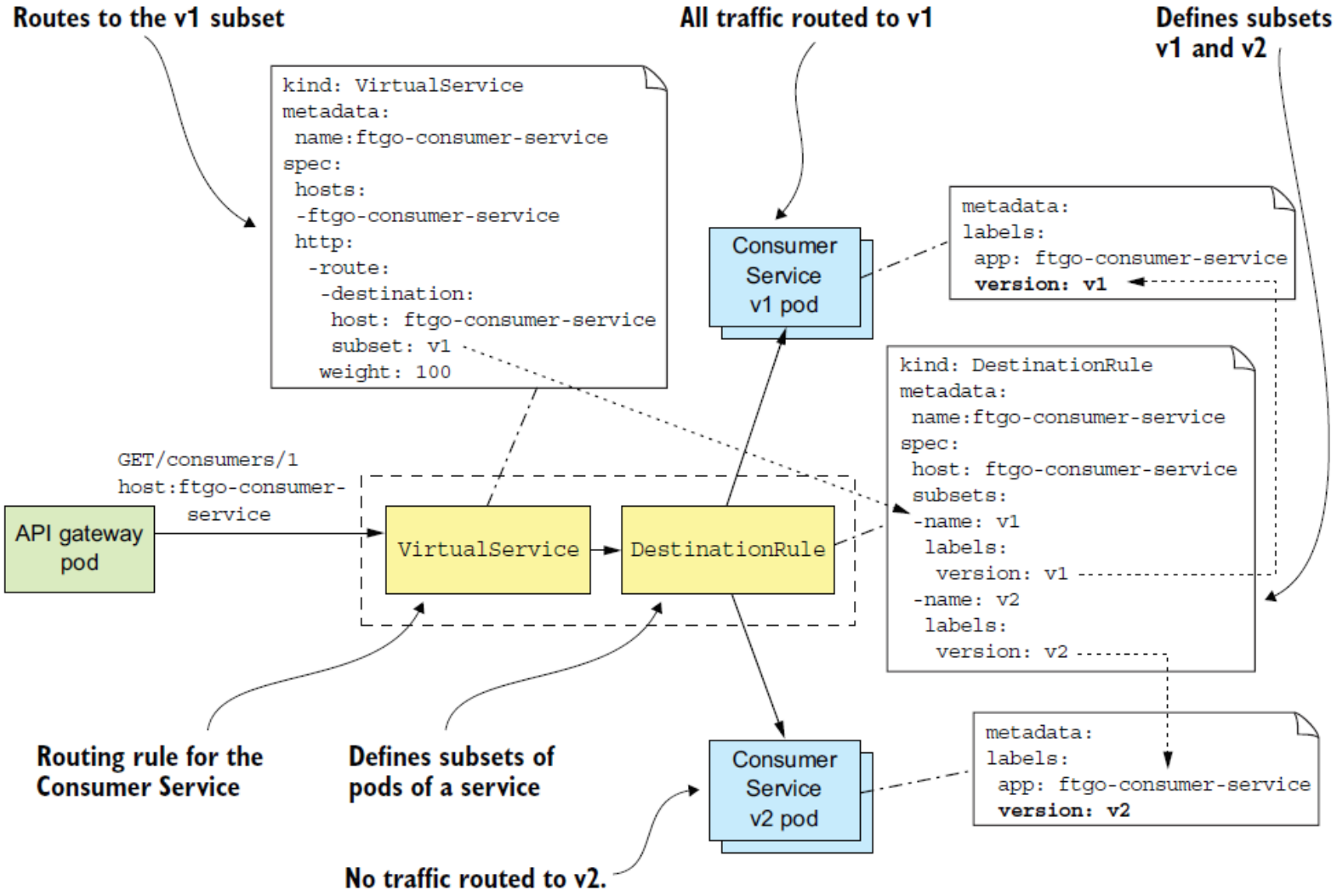
Развертывание сервиса с помощью Istio

- Для каждого сервиса приложения необходимо определить объекты Service и Deployment.
- Порт сервиса Kubernetes должен использовать соглашение об именовании, принятое в Istio и имеющее вид <протокол>[-<суффикс>].
  - В качестве протокола можно указать http, http2, grpc, mongo или redis.
  - Если имя не указано, Istio считает, что это порт TCP, и не применяет правила маршрутизации.
- Капсула (Pod) должна иметь метку *app*, которая идентифицирует сервис.

  Это требуется для поддержки распределенной трассировки в Istio.
- Чтобы иметь возможность запускать сразу несколько версий сервиса, название развертывания Kubernetes должно включать в себя версию
- У капсулы развертывания должна быть метка version, такая как version: vl.

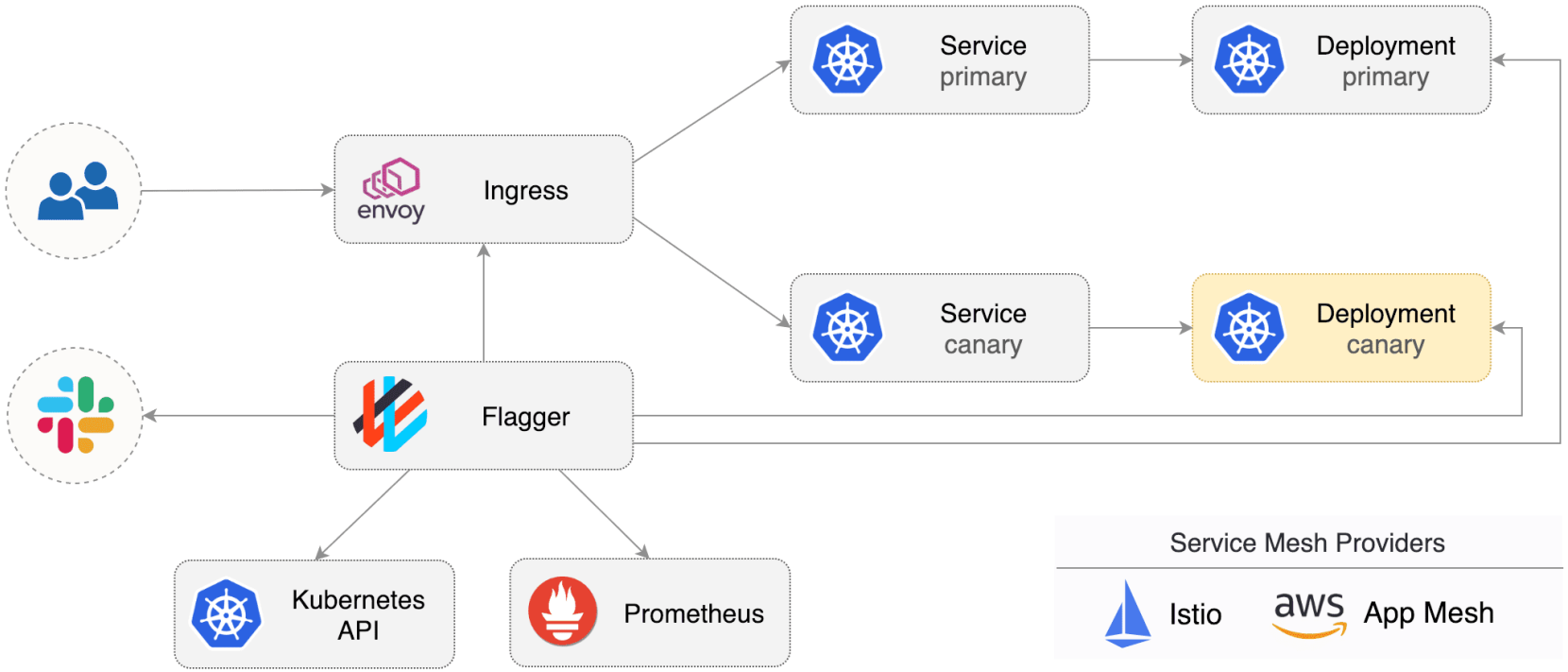
  Это позволяет Istio направлять трафик к конкретной версии сервиса.

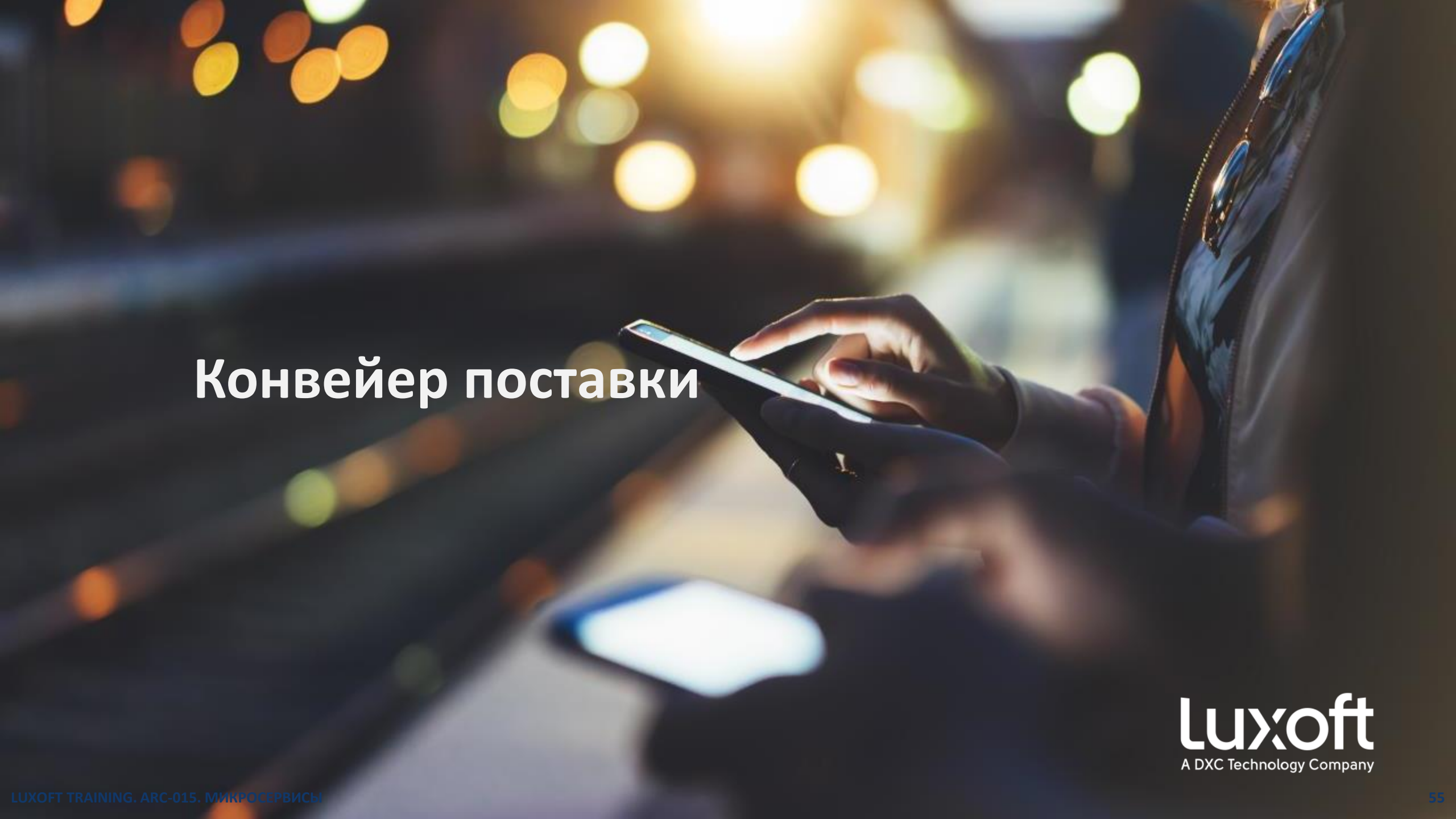
Istio: Правила маршрутизации



Канареечные развертывания с Weaveworks Flagger

Weaveworks Flagger позволяет легко и эффективно управлять канареечными выкатами.





Конвейер поставки

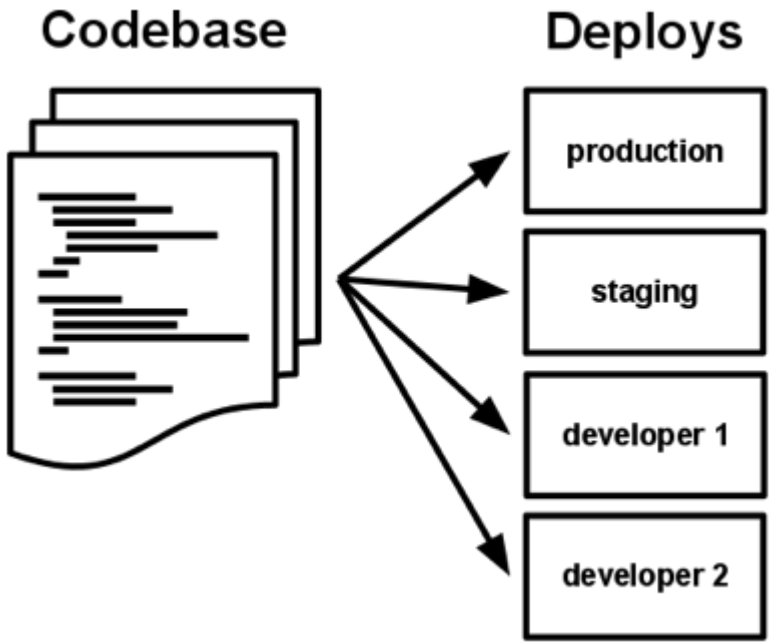
luxoft
A DXC Technology Company

Концепция DevOps

- Ответственность за развертывание приложений и сервисов частично ложится и на разработчиков.
- Варианты:
  - Администраторы предоставляют разработчикам консоль для развертывания проектов.
  - Код автоматически доставляется в промышленную среду после прохождения тестов.
- Проблемы развертывания в MSA
  - Развертывать приходится много
  - Развертывать приходится часто
    - Etsy развертывается 50 раз в день, а Amazon требует развертывания каждую секунду

Кодовая база в MSA

- **Кодовая база обязана быть единой для всех развёртываний, однако разные версии одной кодовой базы могут выполняться в каждом из развёртываний.**

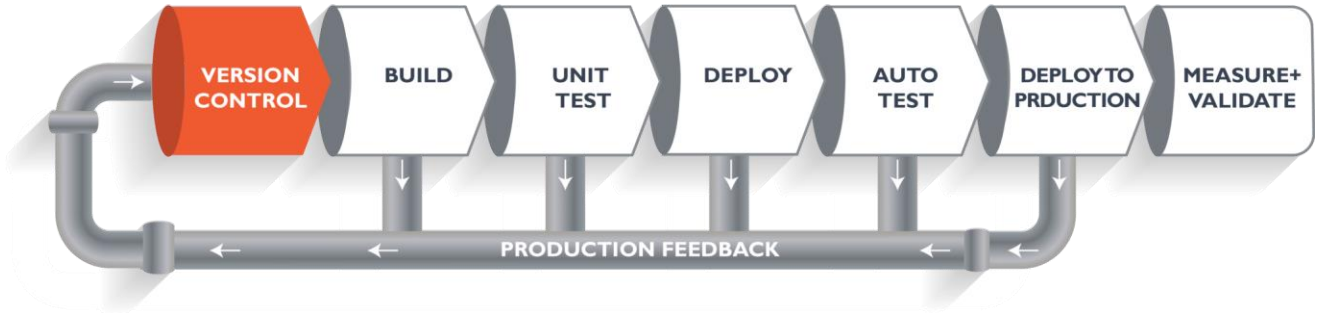


THE TWELVE-FACTOR APP

The twelve-factor app

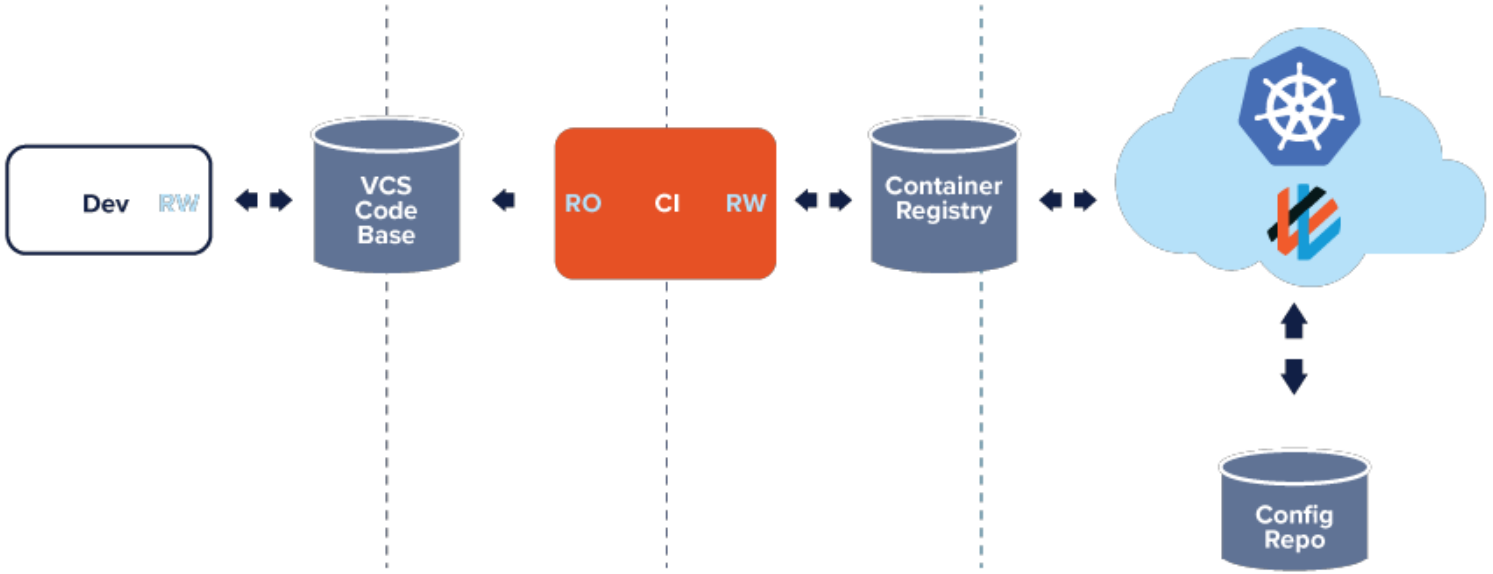
Паттерн «Конвейер поставки» (Deployment pipeline)

- Одна сборка на все окружения
- Конфигурация только из внешних источников (переменные среды, JNDI, база данных конфигураций)
- Общий репозиторий сборки
- Переход к следующему шагу только после удачного выполнения предыдущего шага
- Возможность отката сборки
- Изменения базы должны быть встроены в сборку (например использование Licuibase)



GitOps

- **Работает, используя Git как единый источник истины для декларативной настроек инфраструктуры и приложений.**
- **Любые операции по развертыванию или реконфигурированию среды проводятся через запрос в Git.**



Буду рад ответить на ваши вопросы