Введение в Kubernetes

План лекции

- Архитектура Kubernetes
- Основные понятия в Kubernetes
- Способы установки Kubernetes

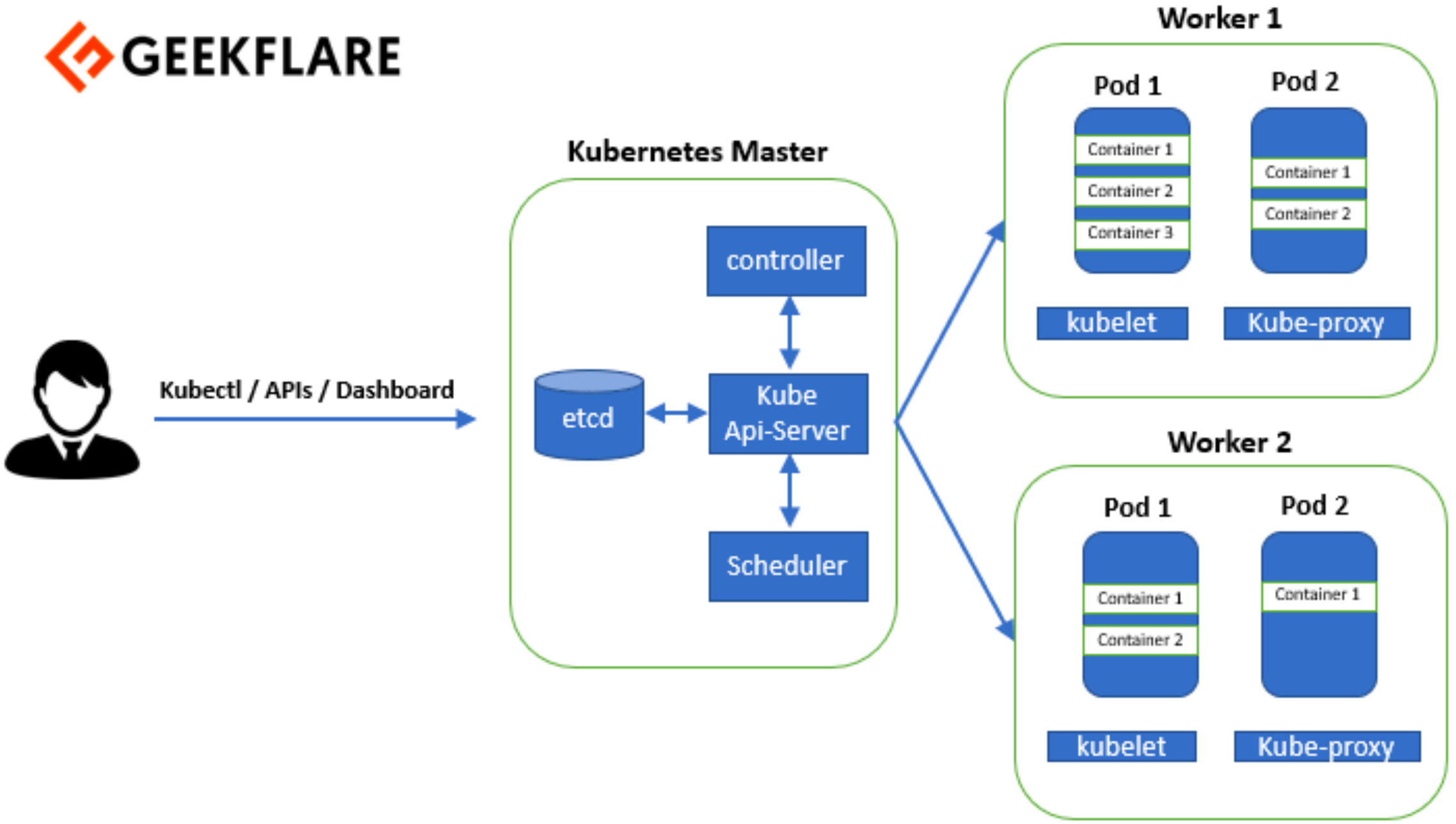
Архитектура Kubernetes

Kubernetes - это современная платформа для оркестрации контейнерных приложений.

Преимущества:

- Повторяемость конфигурации
- Простой скейлинг (горизонтальный и вертикальный)
- Автоматическое увеличение (или уменьшение) числа нод
- Автоматическое рекавери

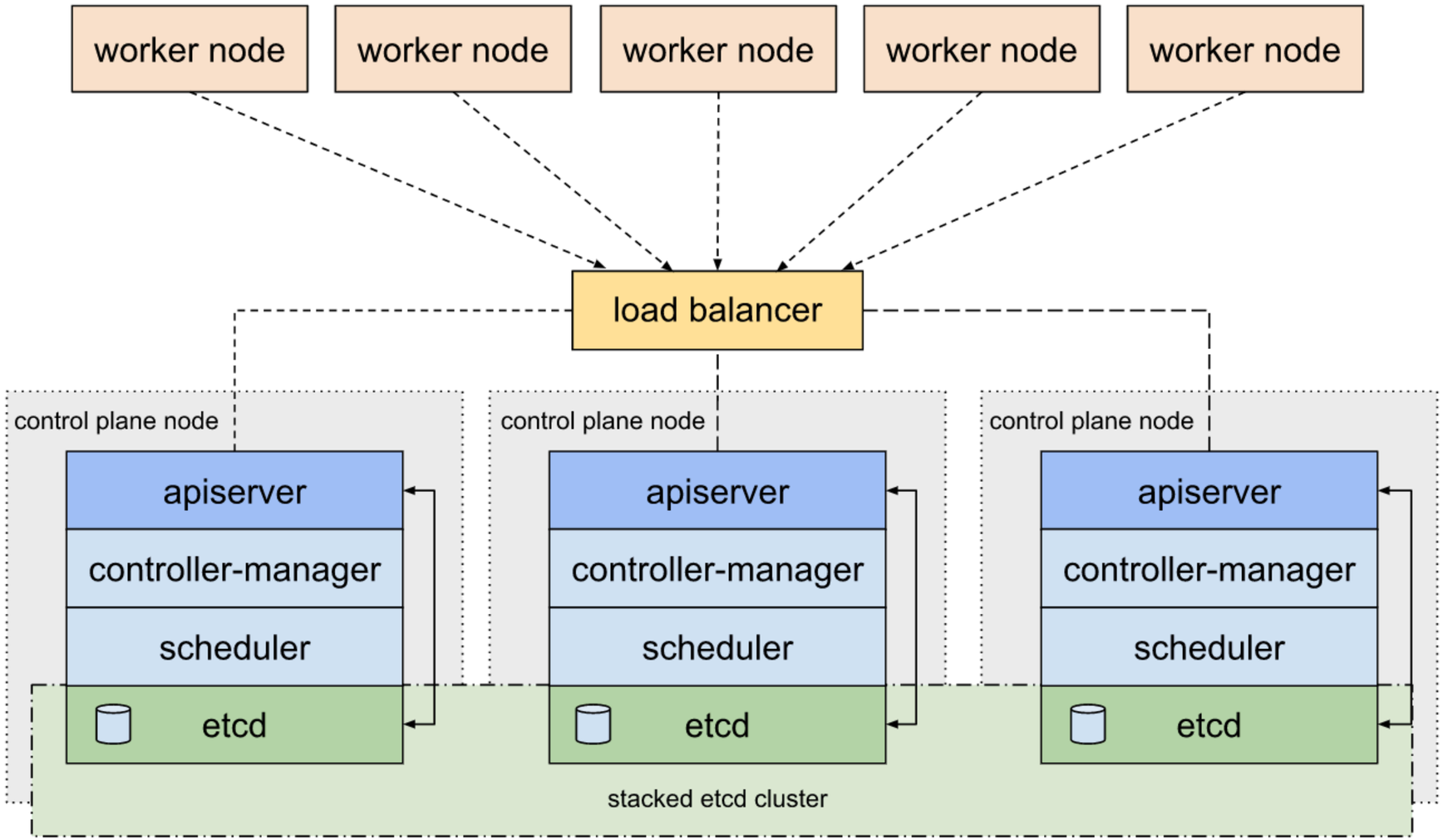
GEEKFLARE


Worker 1
Pod 1
Pod 2
Container 1
Container 2
Container 3
Container 1
Container 2
kubelet
Kube-proxy
Kubernetes Master
controller
etcd
Kube
Api-Server
Scheduler
Kubectl / APIs / Dashboard
Worker 2
Pod 1
Pod 2
Container 1
Container 2
Container 1
kubelet
Kube-proxy

Master ноды

- master - виртуалка, содержащая компоненты control-plane
- **api-server** - валидирует и конфигурирует данные для различных объектов
- **controller** - следит за состоянием кластера и руководит приведением текущего состояния кластера в желаемое
- **sheduler** - распределяет рабочую нагрузку с учетом текущей загруженности нод кластера и специальных правил
- etcd - это высоко-доступная key-value DB на Raft, которая используется для хранения всех данных в кубернетес

Популярная топология: 3 мастера, в котором по одному инстансу каждого компонента control-plane



Worker ноды

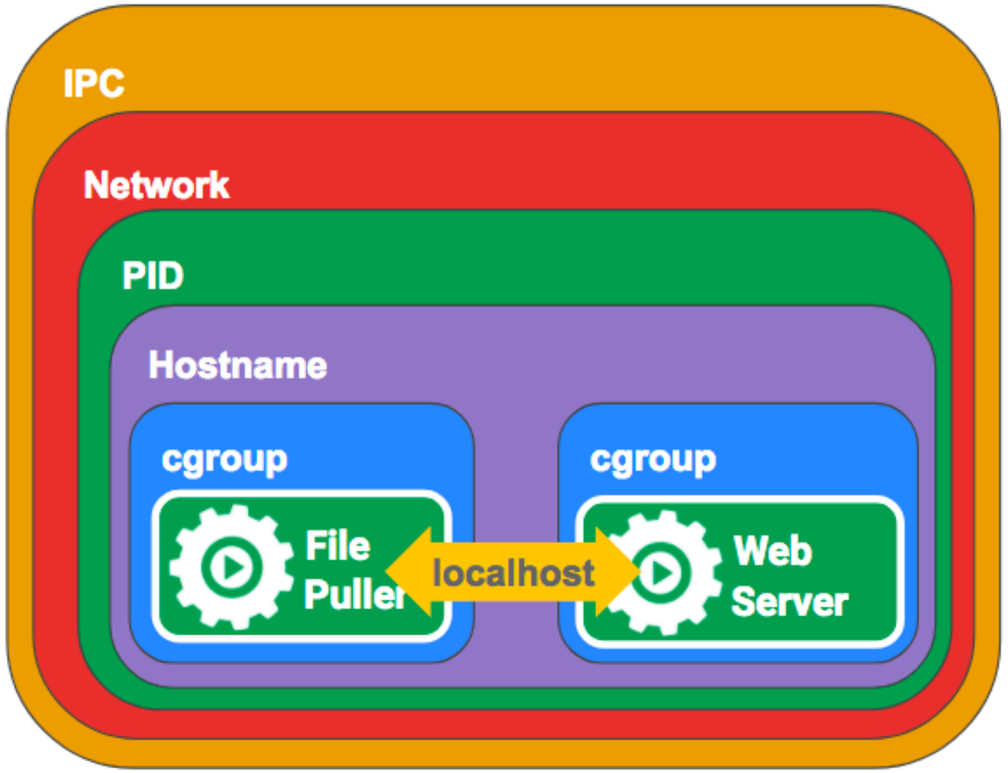
- **kubelet** - управляет жизненным циклом: подготовка среды для запуска, создание контейнера и его терминация
- **kube-proxy**, отвечающий за сетевое взаимодействие
- **container-runtime**: Docker, CRI-O, rkt и другие

Q&A time

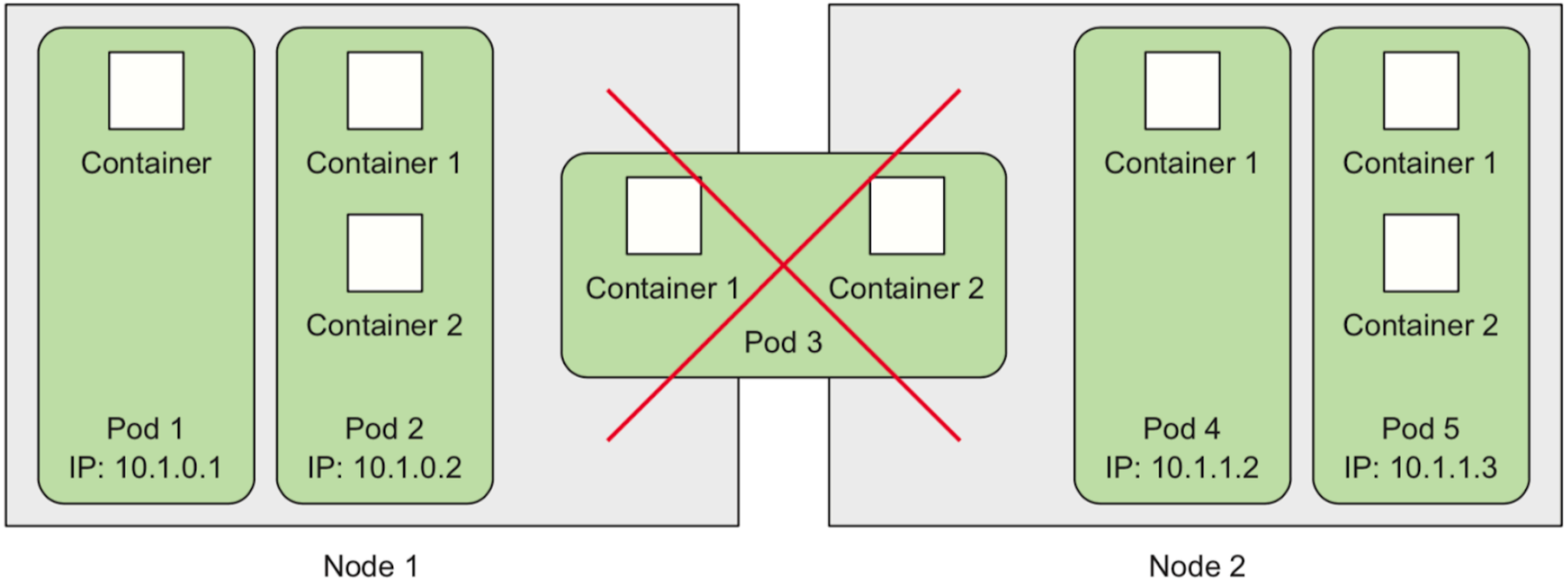
Объекты в Kubernetes

Pod

- это логически объединенная группа контейнеров (один или больше), которые имеют общую виртуальную среду (например общий сетевой немспейс).



Под - не делимая сущность, контейнеры шедулятся и масштабируются вместе.



Чтобы понять, как следует формировать поды с приложениями попробуйте ответить на несколько вопросов:

- Приложения должны масштабироваться вместе или по отдельности?
- Должны ли сервисы запускаться на одной ноде или могут быть разнесены на разные?
- Это связанные сервисы или независимые компоненты?

Пример не связанных сервисов: frontend & backend

Пример связанных сервисов: само приложение и контейнер для сбора метрик и экспоуза их для системы мониторинга

Описание объекта Pod на YaML

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
```

apiVersion - версия api Kubernetes

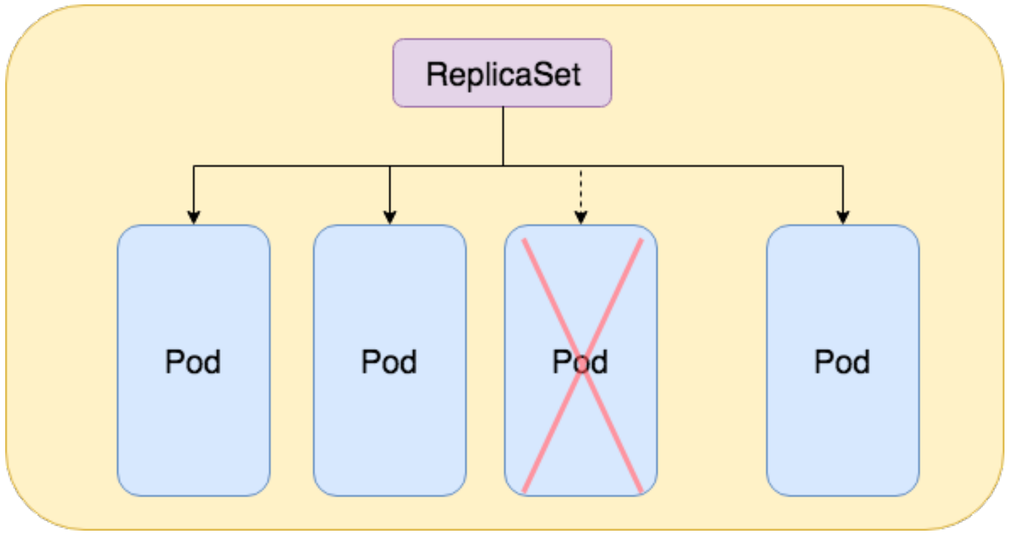
kind - тип объекта, который хотим создать

metadata - содержит имя, лейблы и аннотации (служебная информация)

spec - содержит описание создаваемого объекта

ReplicaSet

- это Kubernetes controller, который используется для контроля необходимого числа запущенных реплик приложения. Объект типа ReplicaSet использует лейблы, чтобы найти поды, которыми он будет управлять.



Описание объекта ReplicaSet на YaML

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
name: nginx-rs
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
```

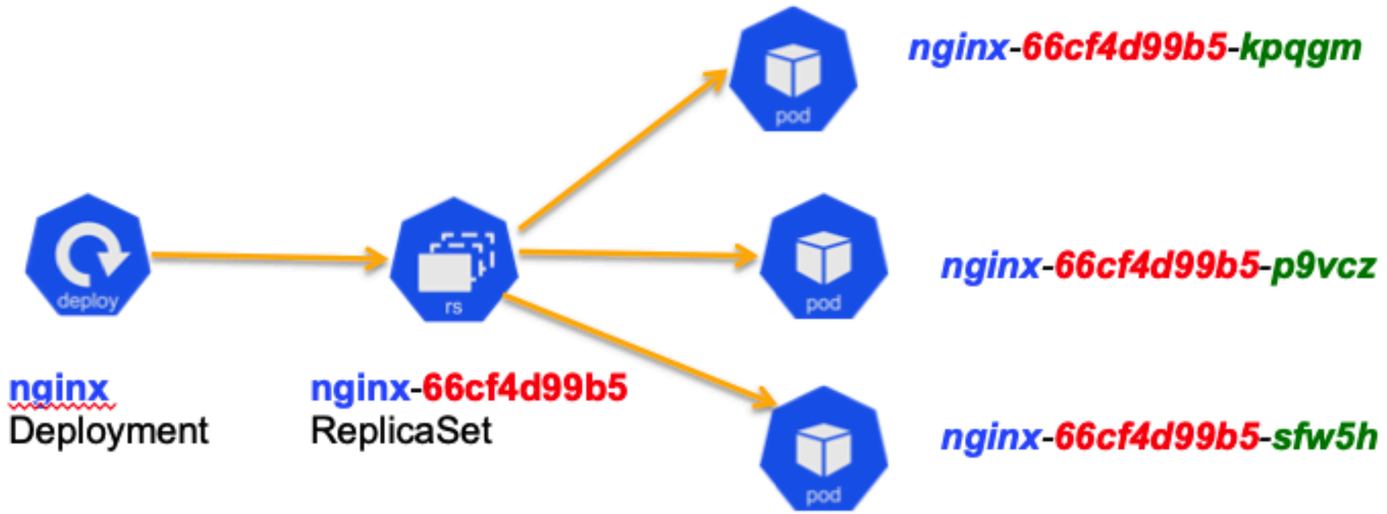
replicas - желаемое число реплик

selector - поды с какими лейблами отслеживать

template - темплейт для пода (тоже самое, что из YAML для пода)

Deployment

- тип объекта, который предоставляет декларативное обновление Pods и ReplicaSets.



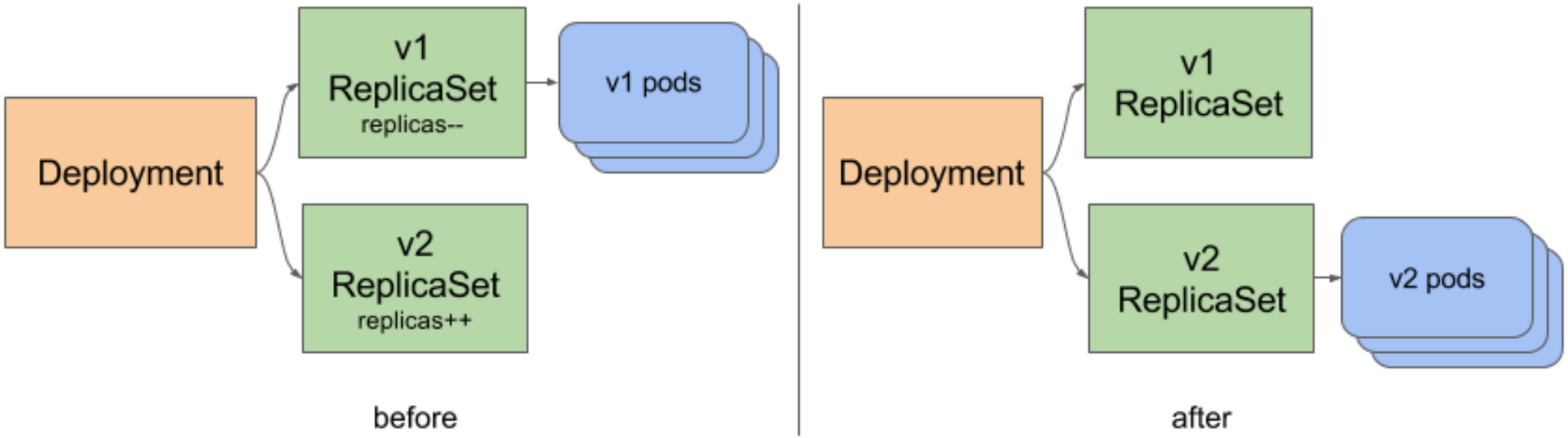
Описание объекта Deployment на YaML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
```

Все тоже самое, что и в YAML для ReplicaSet, но деплоймент, например, позволяет определять деплоймент стратегию и осуществлять роллбек к предыдущей версии.

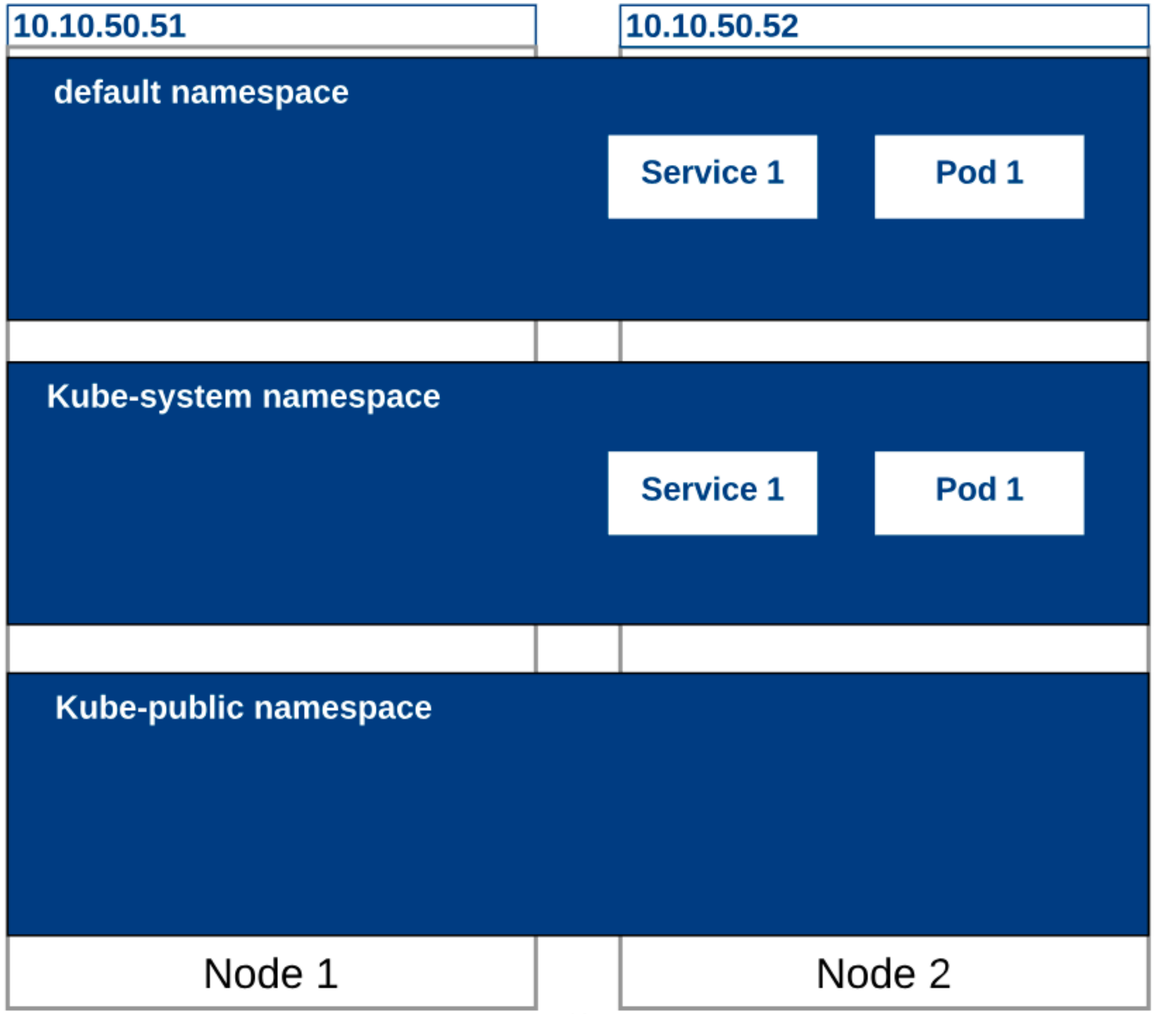
1. Создаем деплоймент yaml, применяем его
2. Деплоймент создает replicaSet
3. ReplicaSet сверяет количество подов по текущему селектору и создает новые, если desired != current

Demo time!


Deployment
v1
ReplicaSet
replicas--
v1 pods
v2
ReplicaSet
replicas++
before
Deployment
v1
ReplicaSet
v2
ReplicaSet
v2 pods
after

Namespace

- это виртуальный кластер внутри вашего физического



Namespace позволяет:

- Изолировать окружения и рабочую нагрузку
- Управлять правами доступа (ограничивать/разрешать доступ)
- Ограничивать потребляемые ресурсы per namespace

По умолчанию доступны:

- **Default**
- **Kube-system** - объекты, созданные kubernetes system
- **Kube-public** - доступен на чтение для всех пользователей в т.ч. неаутентифицированных

Важно помнить:

- При удалении немспейса удаляются все объекты, принадлежащие этому немспейсу.
- Переименовать немспейс нельзя.

Описание объекта Namespace на YaML

```
apiVersion: v1
kind: Namespace
metadata:
  name: custom-namespace
  labels:
    team: devops
    env: staging
```

Service

Существует несколько типов сервисов:

- **ClusterIP** - доступен внутри, тип по умолчанию
  - Из того же namespace: <service_name>: ***web***
  - Из другого namespace: <service_name>.<namespace>: ***web.nginx***
  - Полное имя для доступа: <service_name>.<namespace>.svc.cluster.local: ***web.default.svc.cluster.local***
- **LoadBalancer** - доступен снаружи через создание лоад балансера через облачного провайдера, где задеплоен кластер. NodePort / ClusterIP создаются автоматически для этого сервиса.
- **NodePort** - выставляет сервис через использование статичного порта на каждой ноде. <NodeIP>:<NodePort>
- **ExternalAddress** - тип сервиса позволяющий создать CNAME для какого-нибудь внешнего сервиса. Mongodb.production -> mongo1-c01-sdfg.

Описание объекта Service на YaML

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: webapp
    role: frontend
spec:
  # type is optional, if ClusterIP desired
  type: ClusterIP
  ports:
  - port: 443
    name: https
    targetPort: https
    selector:
      app: webapp
      role: frontend
```

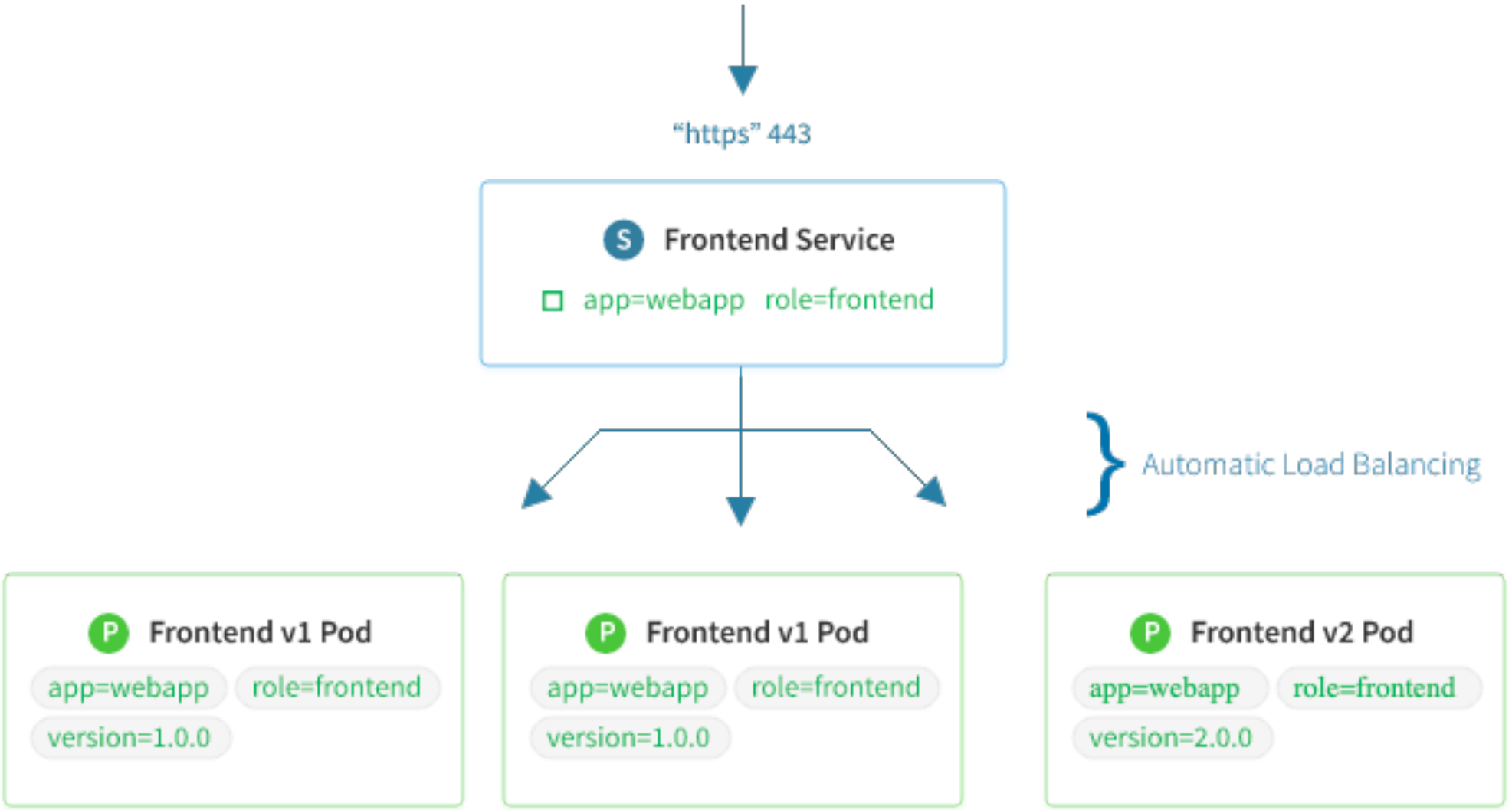
ports - содержит в себе массив портов, содержащих информацию о том:

- какой порт открыть на сервисе

- на какой порт в приложении перенаправлять трафик

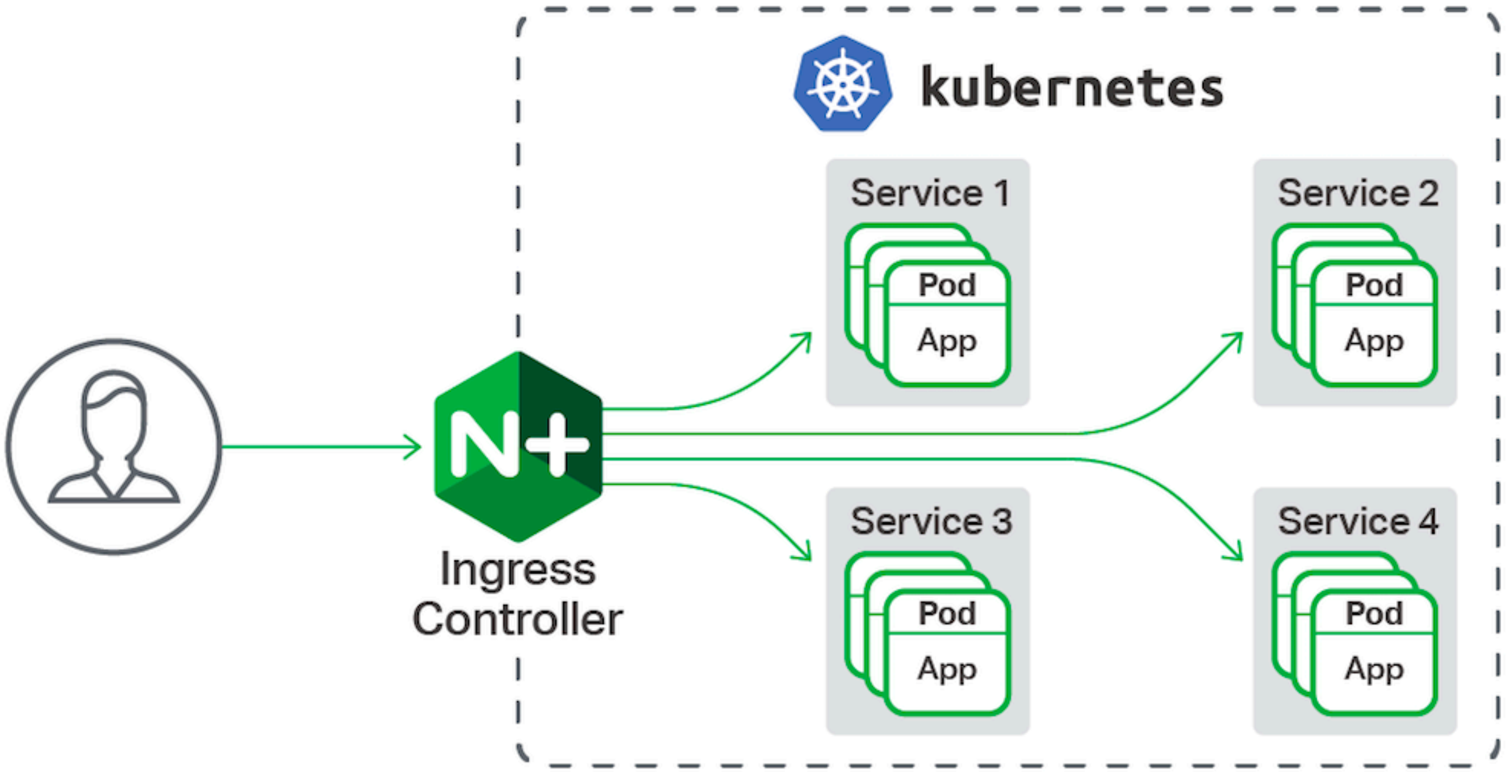
- имена source / target портов

- селектор, содержащий в себе лейблы подов, которые должны отвечать на запросы к этому сервису.


"https" 443
Frontend Service
app=webapp role=frontend
Automatic Load Balancing
Frontend v1 Pod
app=webapp
role=frontend
version=1.0.0
Frontend v1 Pod
app=webapp
role=frontend
version=1.0.0
Frontend v2 Pod
app=webapp
role=frontend
version=2.0.0

Ingress

- это тип объекта, который управляет доступом снаружи к сервисам внутри кластера.



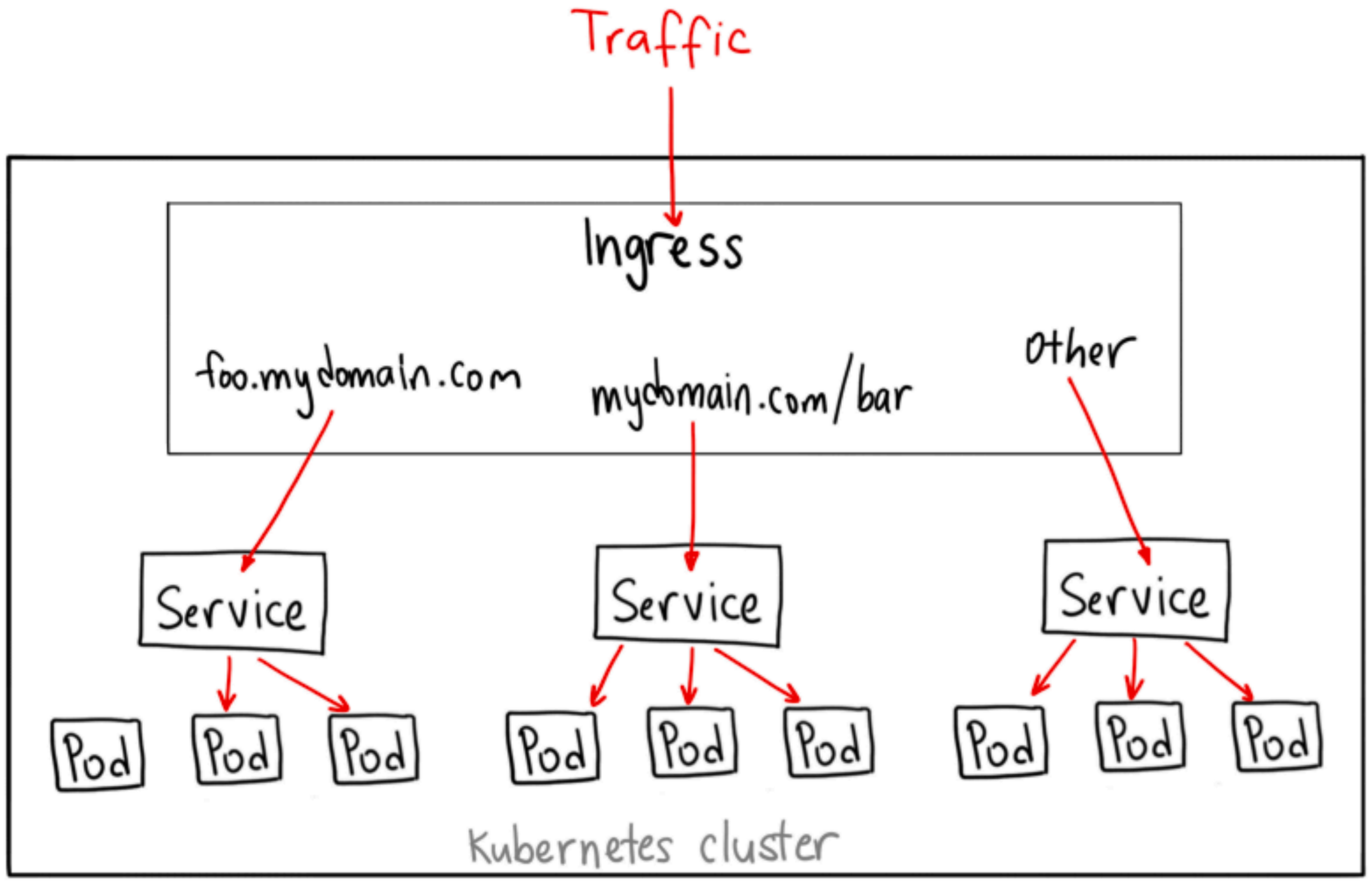
- От того, что вы задеплоите Ingress объект внешний роутинг работать не будет.
- Соответственно, за Ingress объектами должен кто-то следить
- … и это очередной контроллер: Ingress-Controller
- Их существует довольно много, но самые популярные:
  - Nginx ingress controller
  - Istio ingress controller
  - Traefik controller
  - HA proxy ingress controller

Описание объекта Ingress на YaML

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ui
spec:
rules:
  - http:
    paths:
    - path: /ui
      backend:
        serviceName: nginx
        servicePort: http
```

В paths: описываем пути, которые необходимо роутить и на какой backend (имя сервиса) нужно отправлять.

Demo time!


Traffic
Ingress
foo.mydomain.com
mydomain.com/bar
other
Service
Service
Service
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Kubernetes cluster

Достойны упоминания

- **Daemonsets** - тоже самое, что и деплоймент, только будет запущено на всех нодах. (Может использоваться для сбора логов, мониторинга воркер нод и т.д.)
- **StatefulSets** - почти тоже самое, что деплоймент, но имеет заранее известный список имен - nginx-0, nginx-1 and etc и другие особенности.
- **Configmap** - объект для хранения конфигурации: переменных, файлов.
- **Secrets** - тоже самое, что и Configmap, но закодированное в base64
- **Job/CronJob** - Job создает поды и убеждается, что они терминировались успешно, CronJob умеет запускать такие Job по расписанию. Можно использовать для создания бекапов, очистки чего-либо, подготовки репортов и т.д.

Способы установки Kubernetes

- Simple: minikube, kind, docker desktop in Mac / Windows
- Medium: cloud-managed installations - AWS EKS, GCP GKE, Azure AKS и др.
- Expert: training kubernetes-the-hardway, kubeadm, kops, kubespray

What’s next?

- Books: Kubernetes in Action, Kubernetes for Fullstack developers
- Learning by doing: Katacoda
- Online course: Introduction to Kubernetes
- Article with resources for learning K8S