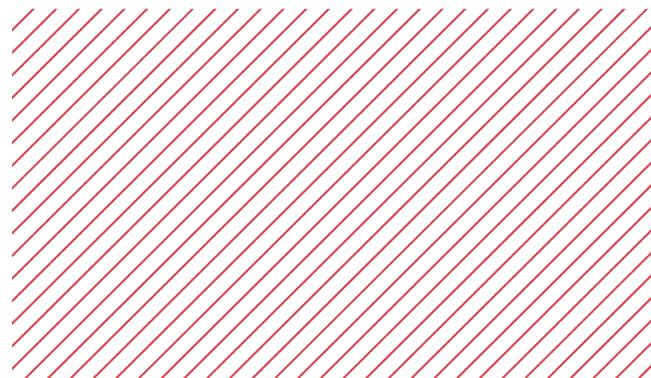


академия  
больших  
данных



# Kubernetes

Михаил Марюфич, MLE





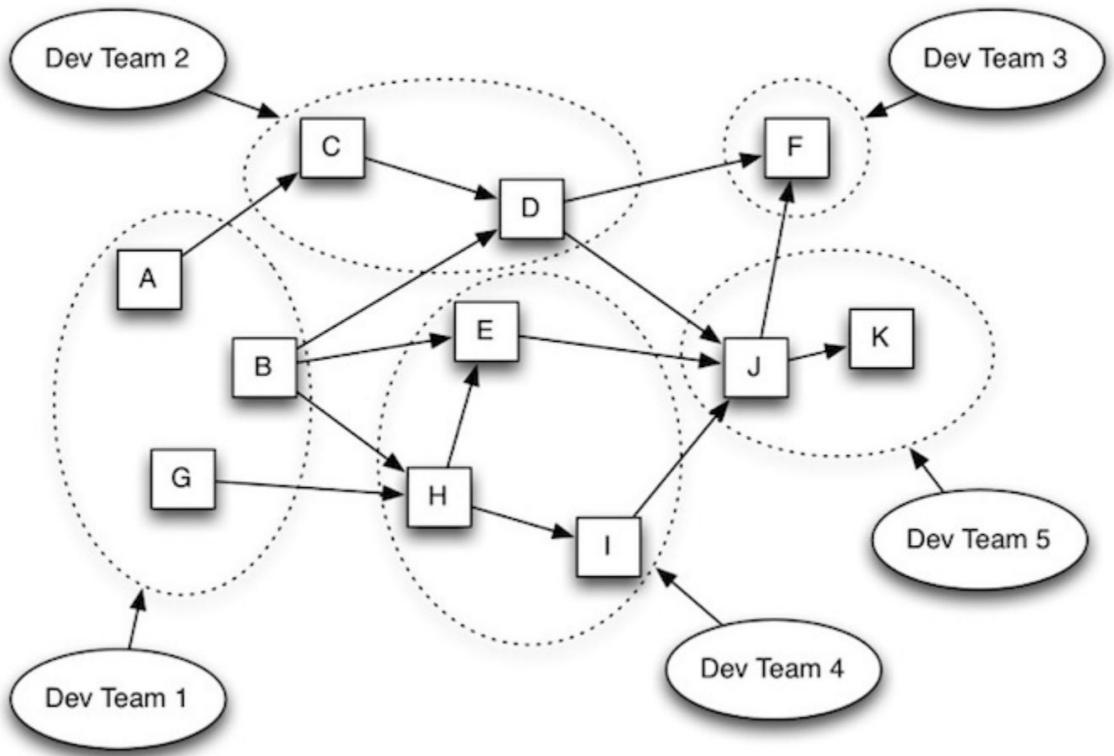
# Содержание занятия

---

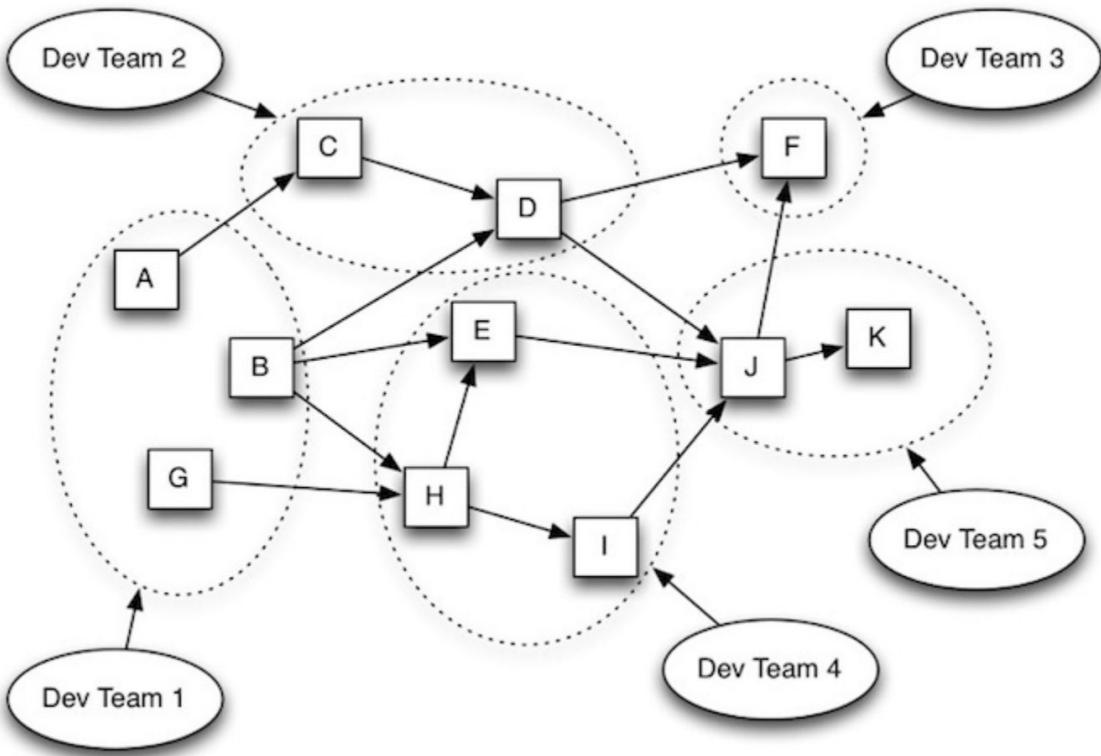
- Зачем нужна контейнерная оркестрация
- Что такое Kubernetes
- Основные сущности Kubernetes
- Запускаем ML модель в kubernetes(в процессе)
- Внутреннее устройство

Зачем нужна контейнерная  
оркестрация?

# Микросервисы



# Микросервисы



- каждое приложение это **docker container**
- каждому приложению нужны **ресурсы**
- они задеплоены на **разных машинах** и им нужно знать друг о друге



# Ресурсы

---

- CPU
- GPU
- MEMORY
- NETWORK
- etc

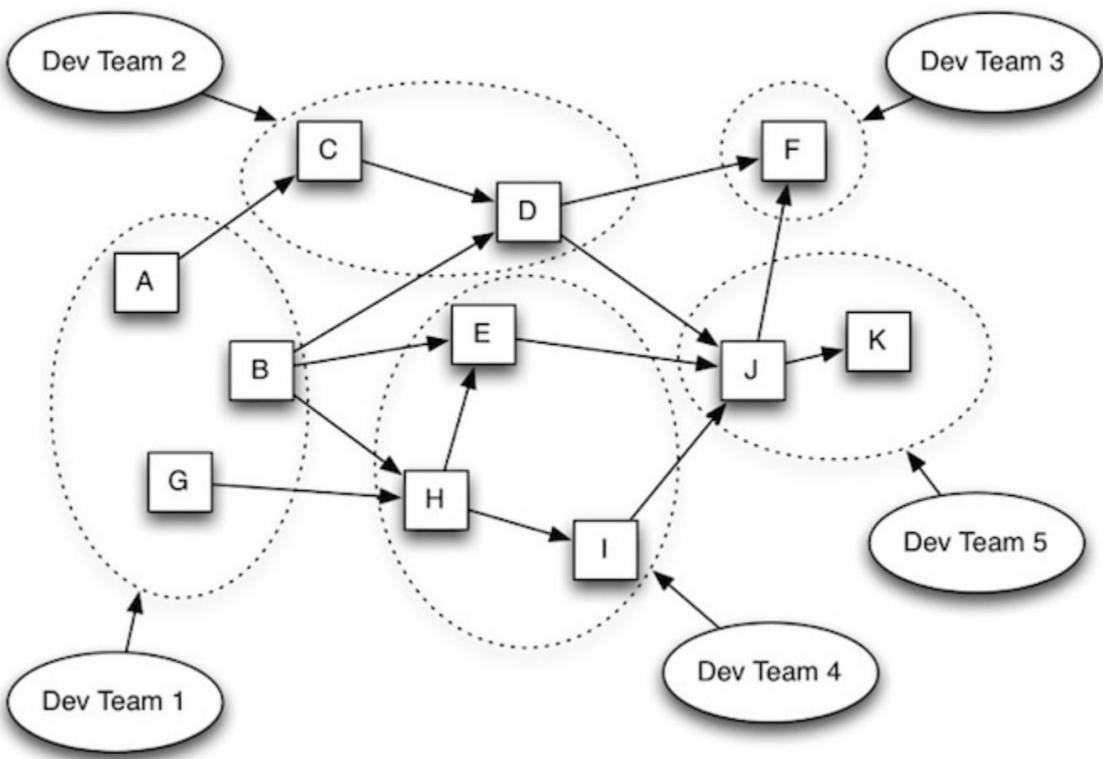


# Ресурсы

---

- CPU Каждая машина имеет разную конфигурацию
- GPU
- MEMORY Каждому приложению необходимо разное количество ресурсов
- NETWORK
- etc

# Микросервисы - проблемы



- Рассчитать потребление ресурсов сложно
- Ручная аллокация приложений становится проблемой

PETS/CATS

```
background-color: #F5F5F5;
text-shadow: 0px 1px 1px #777;
filter: dropshadow(color:#777);
color:#777;

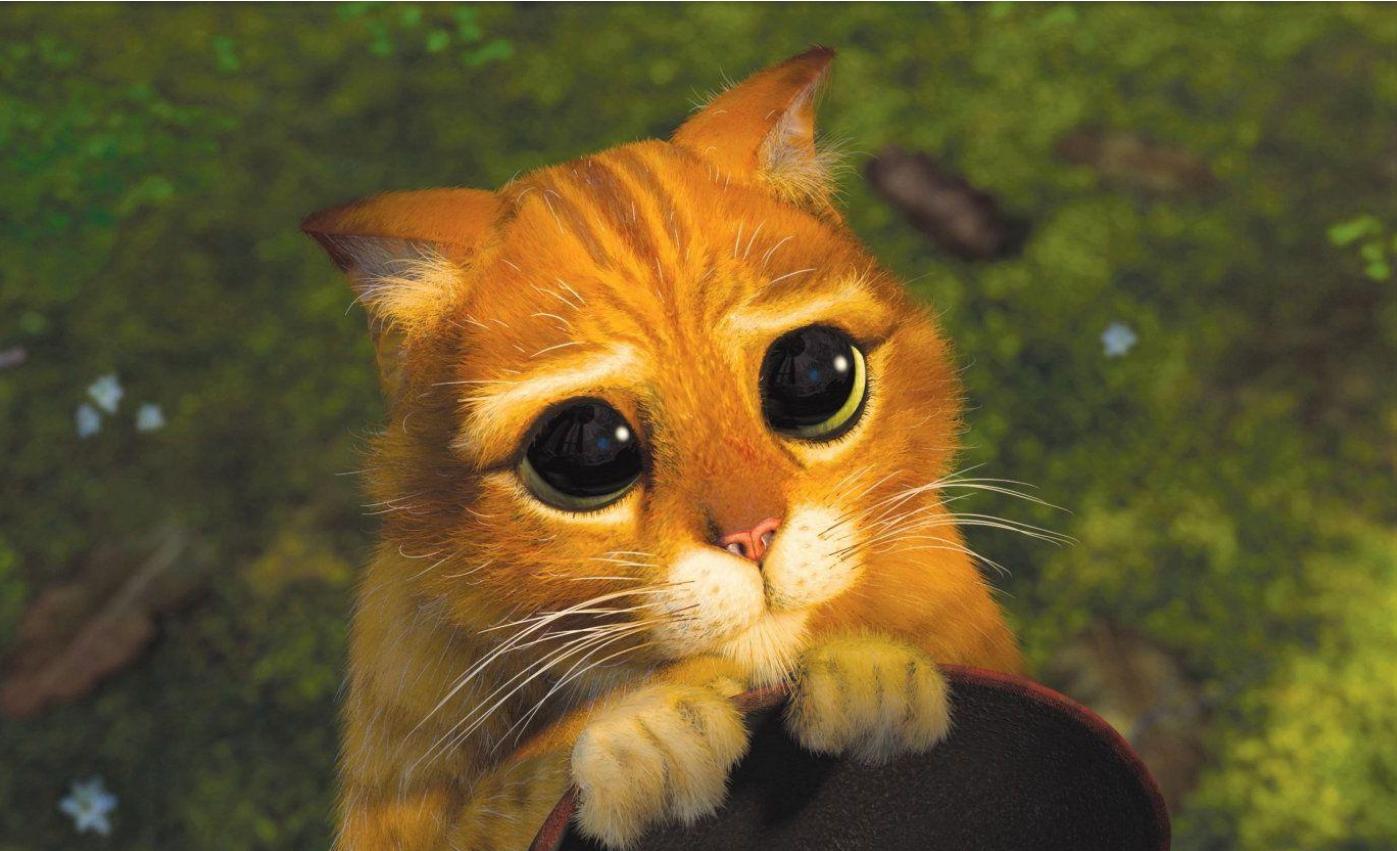
}
header #main-navigation ul li span:hover,
header #main-navigation ul li span:active {
  border: 1px solid #777;
  background-color: #F9F9F9;
  box-shadow: 0px 0px 1px #777;
  -webkit-box-shadow: 0px 0px 2px #777;
  -moz-box-shadow: 0px 0px 1px #777;
}

header #main-navigation ul li span.dashboard,
header #main-navigation ul li span.dashboard:hover,
header #main-navigation ul li span.dashboard:active {
  border: 1px solid #777;
  background-color: #F5F5F5;
  box-shadow: 0px 0px 1px #777;
  -webkit-box-shadow: 0px 0px 2px #777;
  -moz-box-shadow: 0px 0px 1px #777;
}
```

# PETS

---

- Уникален, неповторим
- Если болеет, то мы его лечим
- Если умирает, то скорбим
- Имеет “особое” имя  
(например, Тимоша)



# Стадо

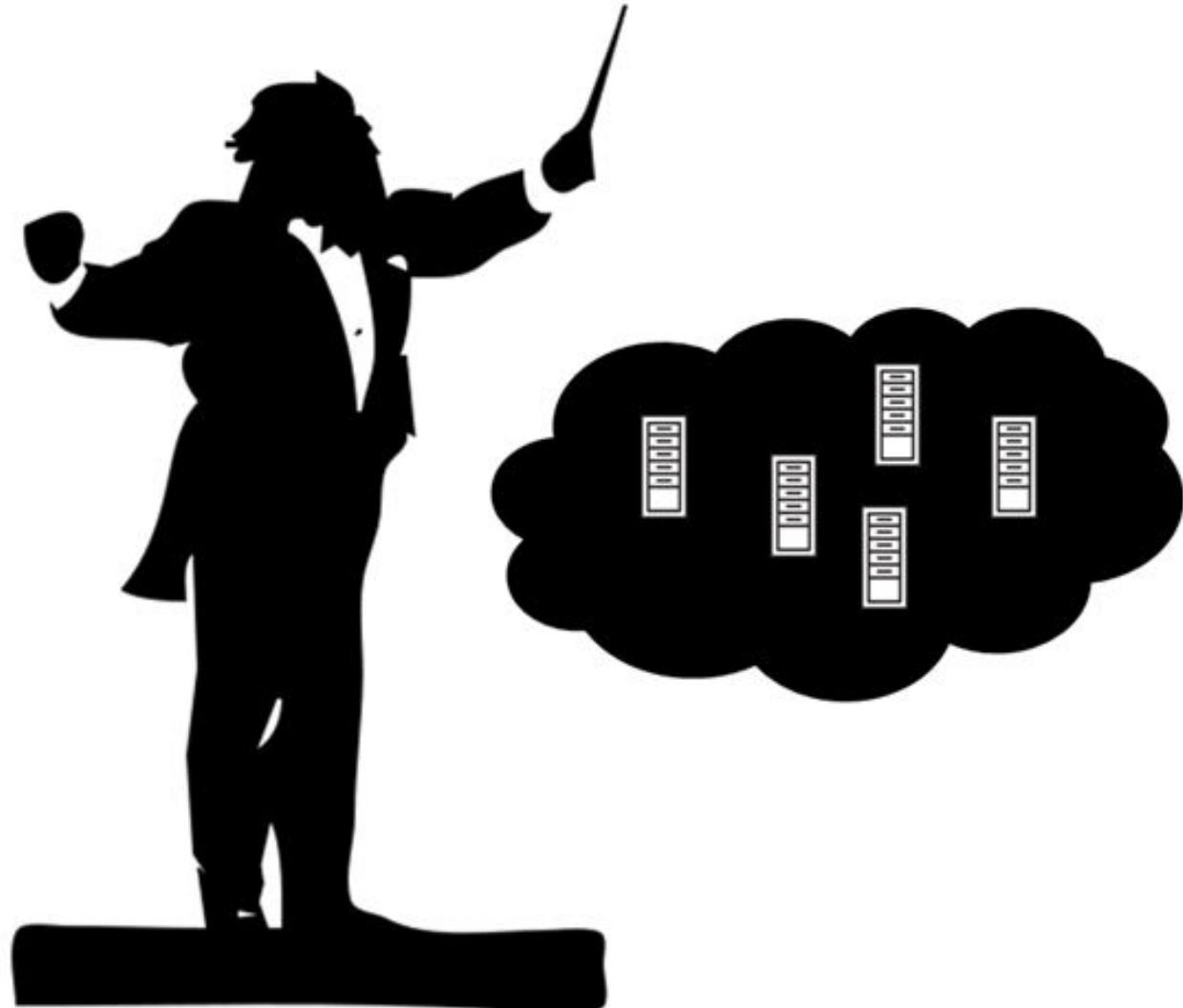
---

- Все одинаковые
- Если болеет, то убиваем и заменяем на такого же
- Вместо имен -- идентификаторы (например, e343t34)



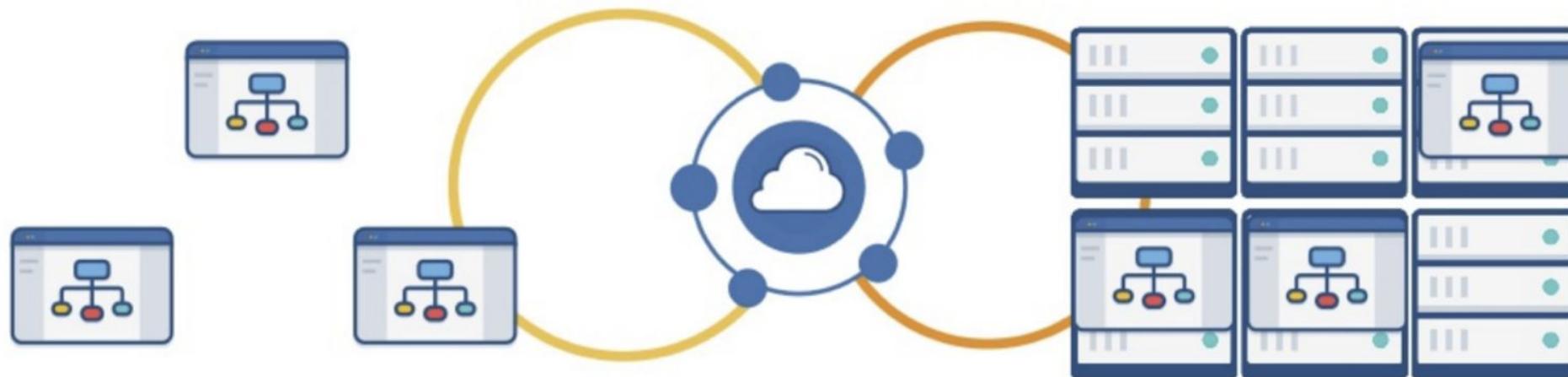
# Оркестрация

---



# Оркестрация

Orchestration  
Framework



Задачи

Ресурсы

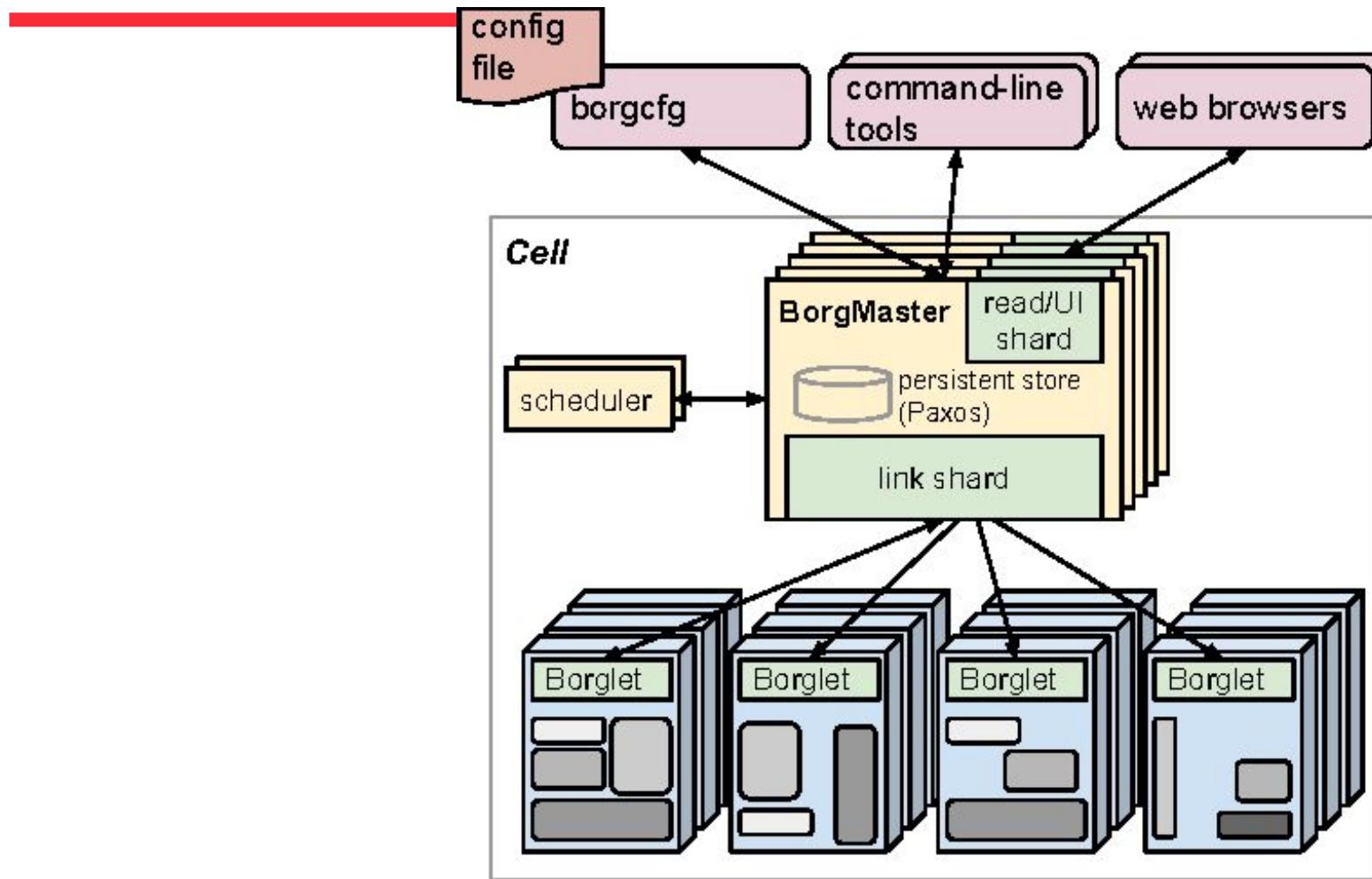


# Что делает оркестратор?

---

- Аллокация(кого-куда поставить)
- Реплицирование/масштабирование
- Проверка готовности сервиса
- Воскрешение
- Перепланирование (rescheduling)
- Управление сервисами
  - service discovery
  - load balancing

# BORG



<https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/>

# Популярные оркестраторы

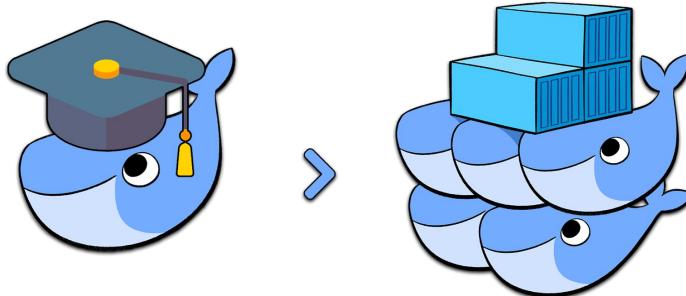
---



**kubernetes**



HashiCorp  
**Nomad**



*docker*

*docker swarm*



# Почему Kubernetes?

---

- Переосмысление проекта Borg от **Google** в контексте контейнеров
- >75 000 звезд на GitHub
- >99 000 коммитов
- Более 4 лет в Open Source
- Версия 1.0 была выпущена 21 июля 2015. Тогда же Kubernetes присоединился к **CNCF** и стал первым выпускником

# Kubernetes

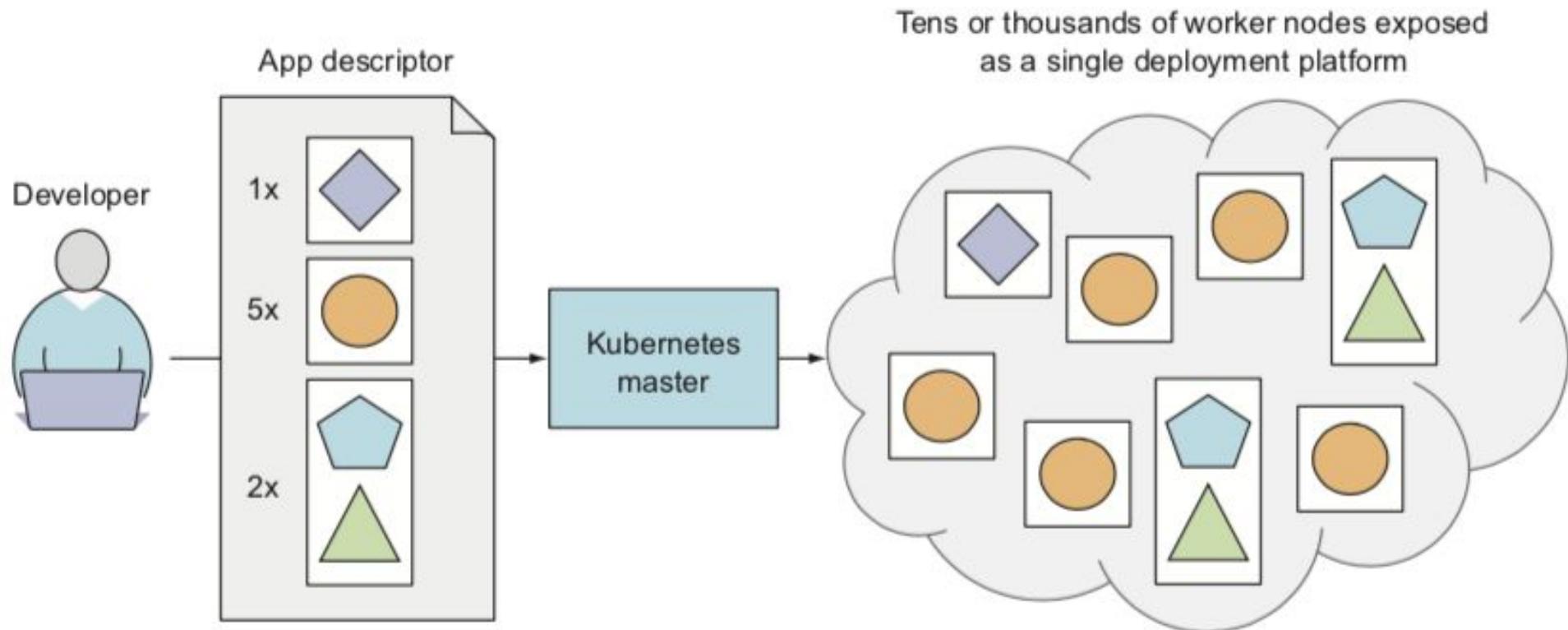


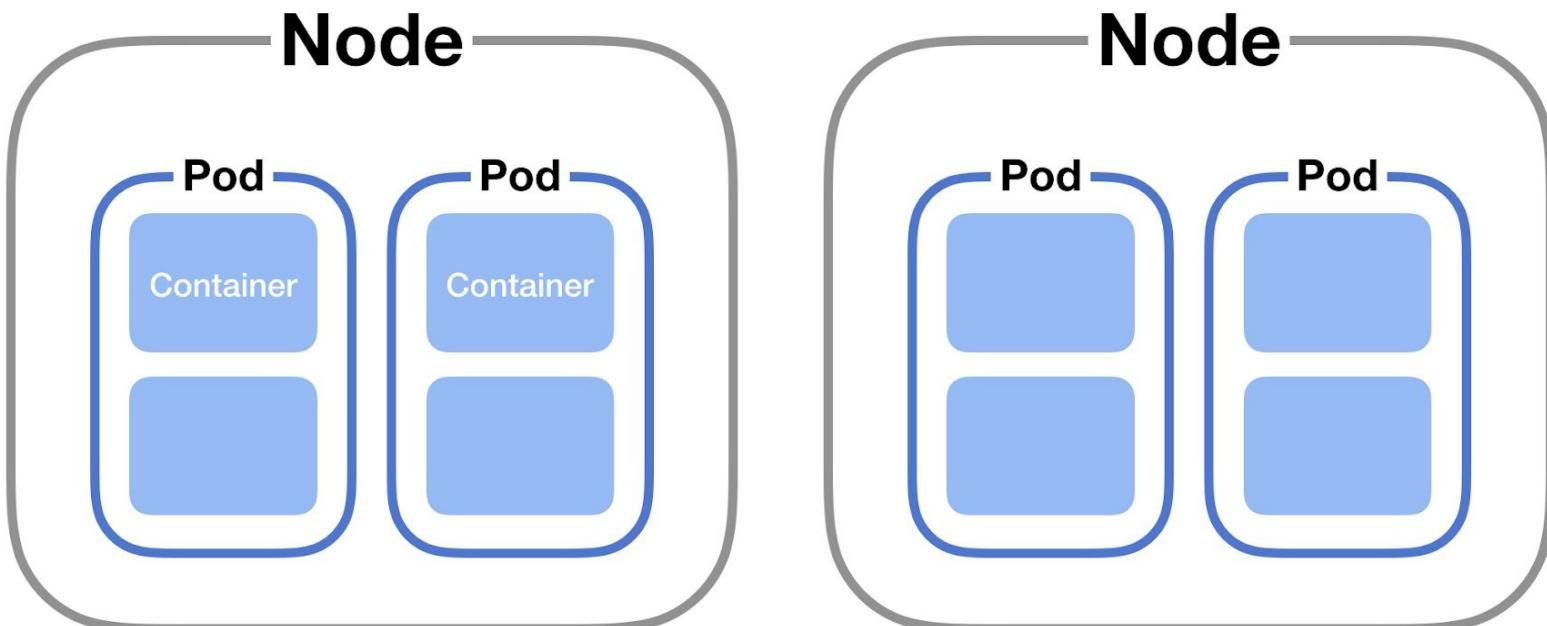
Figure 1.8 Kubernetes exposes the whole datacenter as a single deployment platform.

# Отвлекаемся на поднять кластер

POD

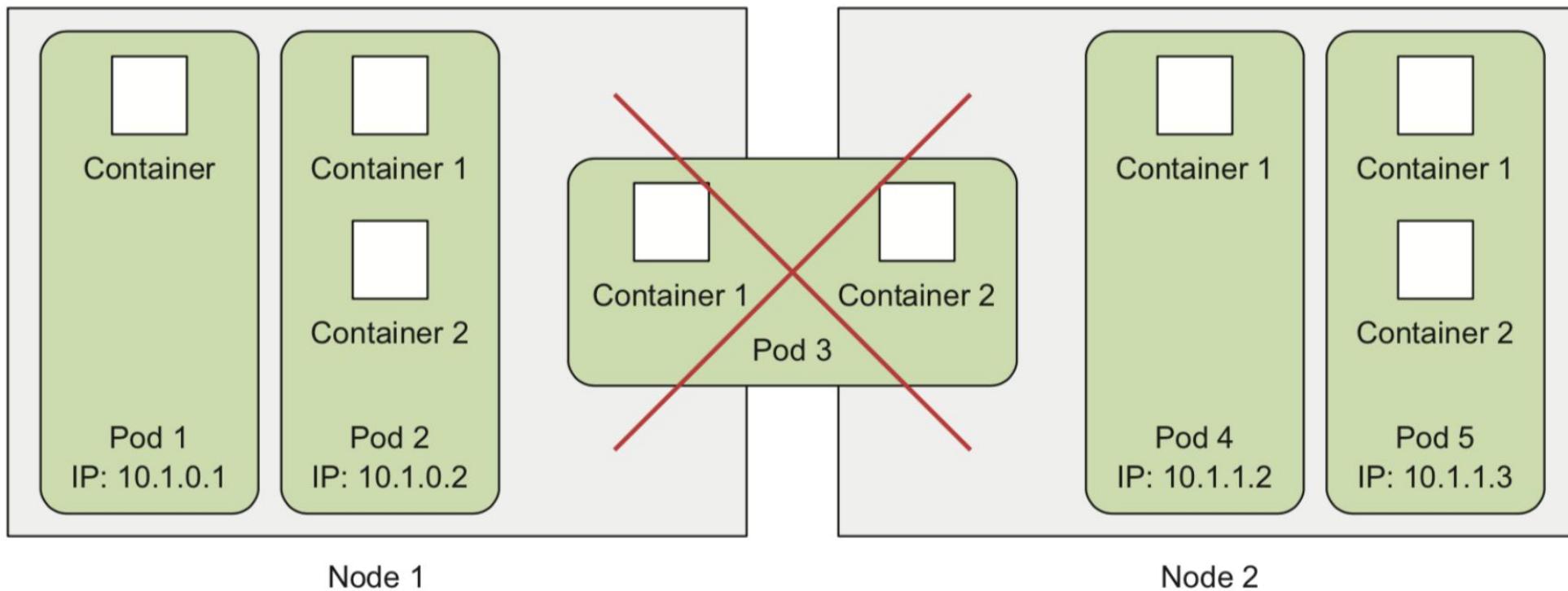
---

Cluster



# POD

- Группа контейнеров (один или несколько)
- Минимальная сущность, управляемая Kubernetes
- У всех контейнеров внутри одного POD общие Network, IPC, UTS, PID\* namespaces





# Манифест podа

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: test
spec:
  containers:
    - image: nginx:1.14
      name: nginx
      ports:
        - containerPort: 80
```

# Манифест podа

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: test
spec:
  containers:
    - image: nginx:1.14
      name: nginx
      ports:
        - containerPort: 80
```

**kind** - Тип объекта, который хотим создать

# apiVersion

---

Путь на используемую группу API для создания объекта

```
apiVersion: $GROUP_NAME/$VERSION
```

3 стадии развития:

**alpha** - стадия тестирования, функционал может быть удален из последующих версий

(v1alpha1)

**beta** - безопасно для использования, но функционал может меняться (v1beta1)

**stable** - стабильно (v1)

# METADATA

---

- Имя создаваемого объекта
- Метки (labels)
- Namespace
- Аннотации

```
metadata:  
  name: my-pod  
  labels:  
    app: test
```

# Спецификация

---

Что запускать и с какими параметрами

**containers:**

- **image:** nginx:1.14

**name:** nginx

**ports:**

- **containerPort:** 80



# Способы общаться с kubernetes

---

- kubectl (command line tool)
- web ui
- API



# Kubectl

---

- CLI утилита, распространяемая в виде бинарного файла
- Конфигурация для подключения к кластеру
- Объекты в кластере можно:
  - Создать (kubectl create)
  - Обновить (kubectl apply)
  - Получить (kubectl get)
  - Посмотреть (kubectl describe)
  - Удалить (kubectl delete)

# DEMO: работа с под

# Работа с ресурсами



# Реквесты

---

Ресурсы, необходимые для работы контейнера

Суммарное значение requests всех pod на хосте не может быть большим, чем количество физических ресурсов хоста

При определении на какой хост поместить тот или иной pod, будут учитывать значения requests

Если на хосте недостаточно физических ресурсов для запуска pod, то kube-scheduler гарантированно не выберет данный хост



# Лимиты

---

Пороговые значения ресурсов, которые может потреблять контейнер

При превышении данных значений контейнер будет:

- Перезапущен в случае превышения по RAM
- Ограничен в потреблении в случае превышения по CPU

Суммарное значение limits всех pod на хосте может быть большим, чем количество физических ресурсов хоста (overcommit)

# Реквесты/лимиты

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

<https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-resource-requests-and-limits>

# DEMO: работа с реквестами/лимитами

# Readiness/liveness пробы

---

Механизм в k8s, используемые для контроля готовности приложения к работе!

```
- image: mikhailmar/online_inference:v2
  name: fastapi-ml
  ports:
    - containerPort: 8000
  readinessProbe:
    httpGet:
      path: /healthz
      port: 8000
    initialDelaySeconds: 15
    periodSeconds: 3
```



# Типы проб

---

**ExecAction:** Command execution check, if the command's exit status is 0, it is considered a success.

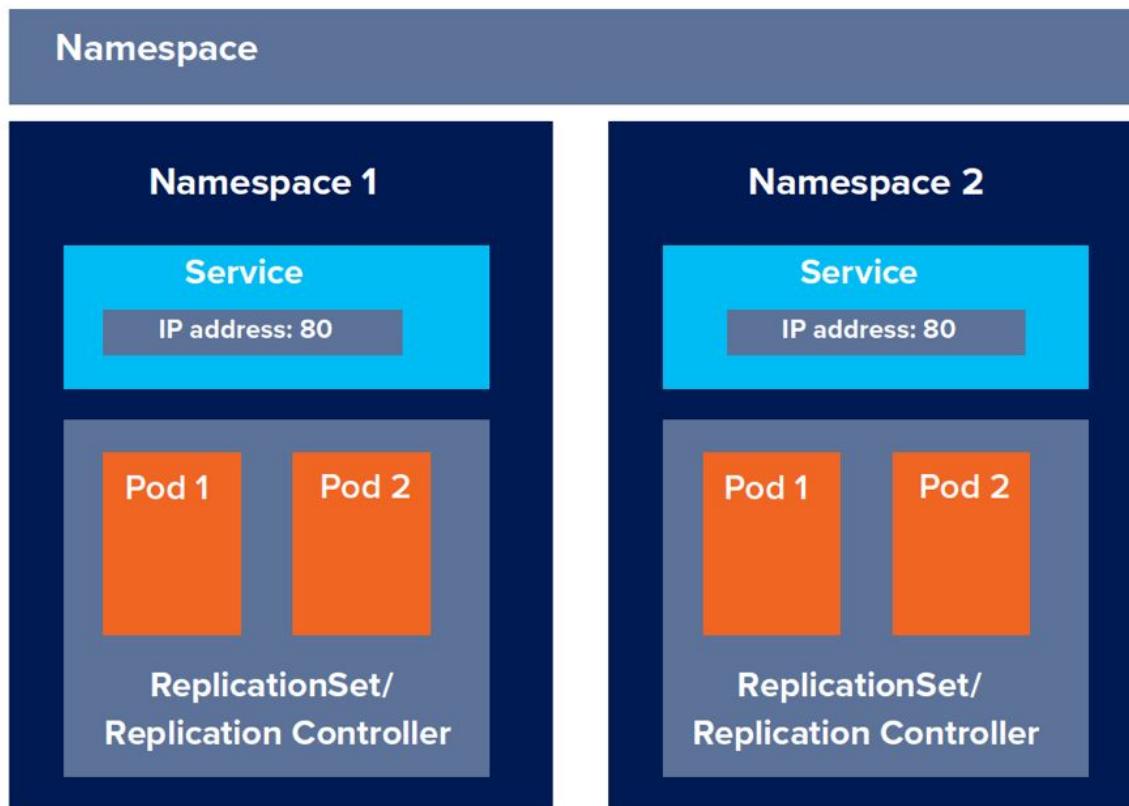
**TCPSocketAction:** TCP check to determine if the port is open, if open, it is considered a success.

**HTTPGetAction:** HTTP check to determine if the status code is equal to or above 200 and below 400.

# DEMO: работа с readinessProbe

# Namespaces

- Можно создать несколько виртуальных кластеров в рамках одного физического кластера
- Namespace - один виртуальный кластер (одно окружение)





## Применяются для

---

- Разграничения прав между командами
- Лимитирования ресурсов на проект с помощью квот (cpu, memory)



# Namespaces по умолчанию

---

- **default** - для объектов у которых явно не определена принадлежность к другому namespace
- **kube-system** - для системных объектов Kubernetes
- **kube-public** - для объектов к которым нужен доступ из любой точки кластера

# DEMO: работа с namespaces



# Конфигурирование приложений

---

Конфигурация приложения – это всё, что может меняться между развёртываниями (среда разработки, промежуточное и рабочее развёртывание). Это включает в себя:

- Идентификаторы подключения к ресурсам типа базы данных, кэш-памяти и другим сторонним службам
- Регистрационные данные для подключения к внешним сервисам, например, к Amazon S3 или Twitter

Нужно отделять конфигурацию от кода

# Переменные окружения

---

Стандартный способ конфигурировать приложение

```
FROM python:3.6
COPY requirements.txt ./requirements.txt
RUN pip install -r requirements.txt

COPY model.pkl /model.pkl
COPY app.py /app.py

WORKDIR .

ENV PATH_TO_MODEL="/model.pkl"

CMD ["uvicorn", "app:app", "--host", "0.0
```

```
postgres:
  image: postgres:12-alpine
  environment:
    - POSTGRES_USER=airflow
    - POSTGRES_PASSWORD=airflow
    - POSTGRES_DB=airflow
  ports:
    - "5432:5432"
  init:
  build:
    context: images/airflow-docker
```

# ENV в манифесте pod

---

```
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envvars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_FAREWELL
      value: "Such a sweet sorrow"
```

# DEMO: работа с ENV

# Конфиги

```
host: '0.0.0.0'  
port: 8000  
model_path: '../models/model.pkl'  
features:  
    - 'age'  
    - 'sex'  
    - 'cp'  
    - 'trestbps'  
    - 'chol'  
    - 'fbs'  
    - 'restecg'  
    - 'thalach'  
    - 'exang'  
    - 'oldpeak'  
    - 'slope'  
    - 'ca'  
    - 'thal'  
  
logging:  
    path: '../../../../../out/logs/inference/online_inference.log'  
    format: '%(asctime)s %(levelname)s %(message)s'  
    date_format: '%Y-%m-%d %H:%M:%S'  
    level: 20 # INFO  
    mode: 'a'  
    stdout: True
```

Очень удобны, но если добавлять их к коду приложения, то нарушается 12 factor.

Приходится пересобирать docker контейнер при изменении образа.

# ConfigMap

---

Стандартный способ хранения конфигурации в kubernetes

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-ml
  labels:
    app: fastapi
data:
  config.yaml: |
    val1: 1231
    kekek: 123
    custom.entry
  another.yaml: |
    your entry
```

# ConfigMap

---

ConfigMap можно примонтировать как volume

```
containers:
  - image: mikhailmar/fastapi_example:v1
    name: fastapi-ml
    ports:
      - containerPort: 8000
volumeMounts:
  - name: config
    mountPath: "/etc/config"
    readOnly: true
volumes:
  - name: config
    configMap:
      name: config-ml
```

# ConfigMap

---

Из ConfigMap можно импортировать в переменные окружения

```
containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    envFrom:
      - prefix: CONFIG_
        configMapRef:
          name: special-config
```



# DEMO: работа с ConfigMap

Pod — это много  
контейнеров

# Аналог Docker Compose?

```
version: "2"

services:
  appA:
    build: ./Dockerfile
    ports:
      - "80:80"
  appB:
    image: mongodb
    volumes:
      - ./data:<wherever>
  appC:
    build: ./Dockerfile
    instances: 2
    environment:
      - config: backend
```

# Аналог Docker Compose?

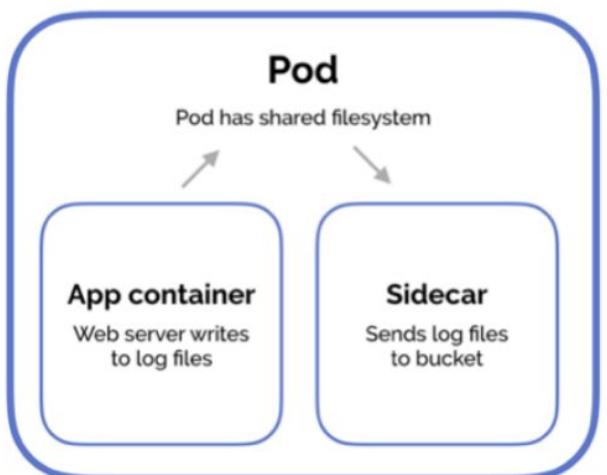
```
version: "2"

services:
  appA:
    build: ./Dockerfile
    ports:
      - "80:80"
  appB:
    image: mongodb
    volumes:
      - ./data:<wherever>
  appC:
    build: ./Dockerfile
    instances: 2
    environment:
      - config: backend
```

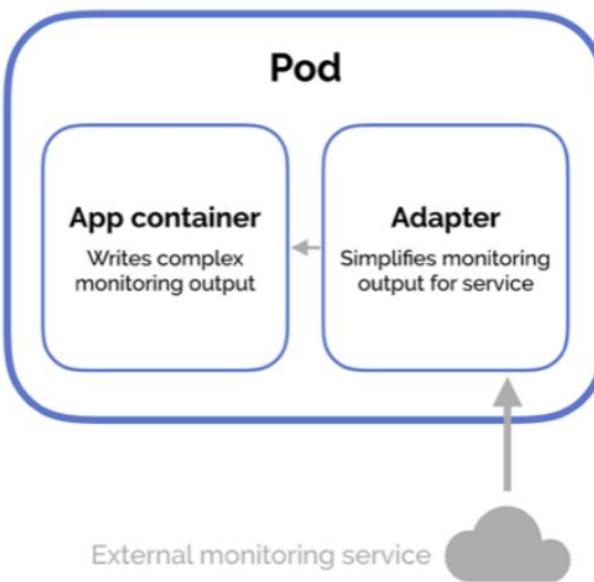
- **Docker compose** придумали для того, чтобы создавать себе на локальном компьютере (или в окружении для тестов) **эмуляцию** продакшен среды
- Не то, как приложения будут задеплоены в реальной жизни

# Паттерны использования POD

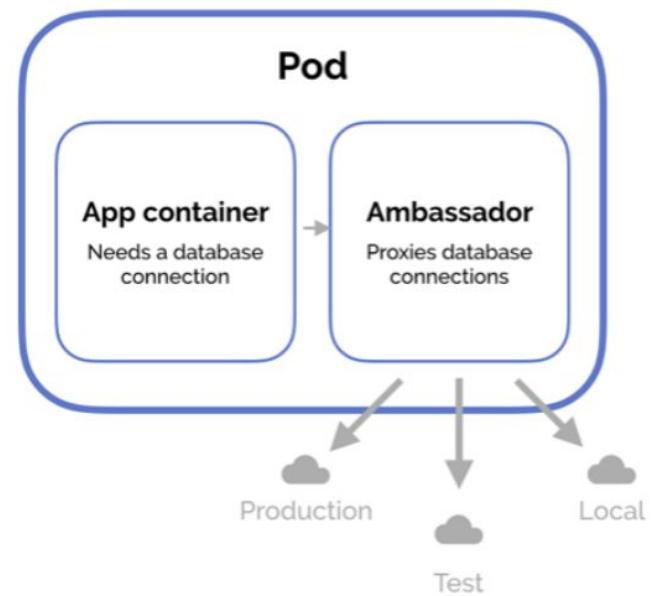
**Sidecar**



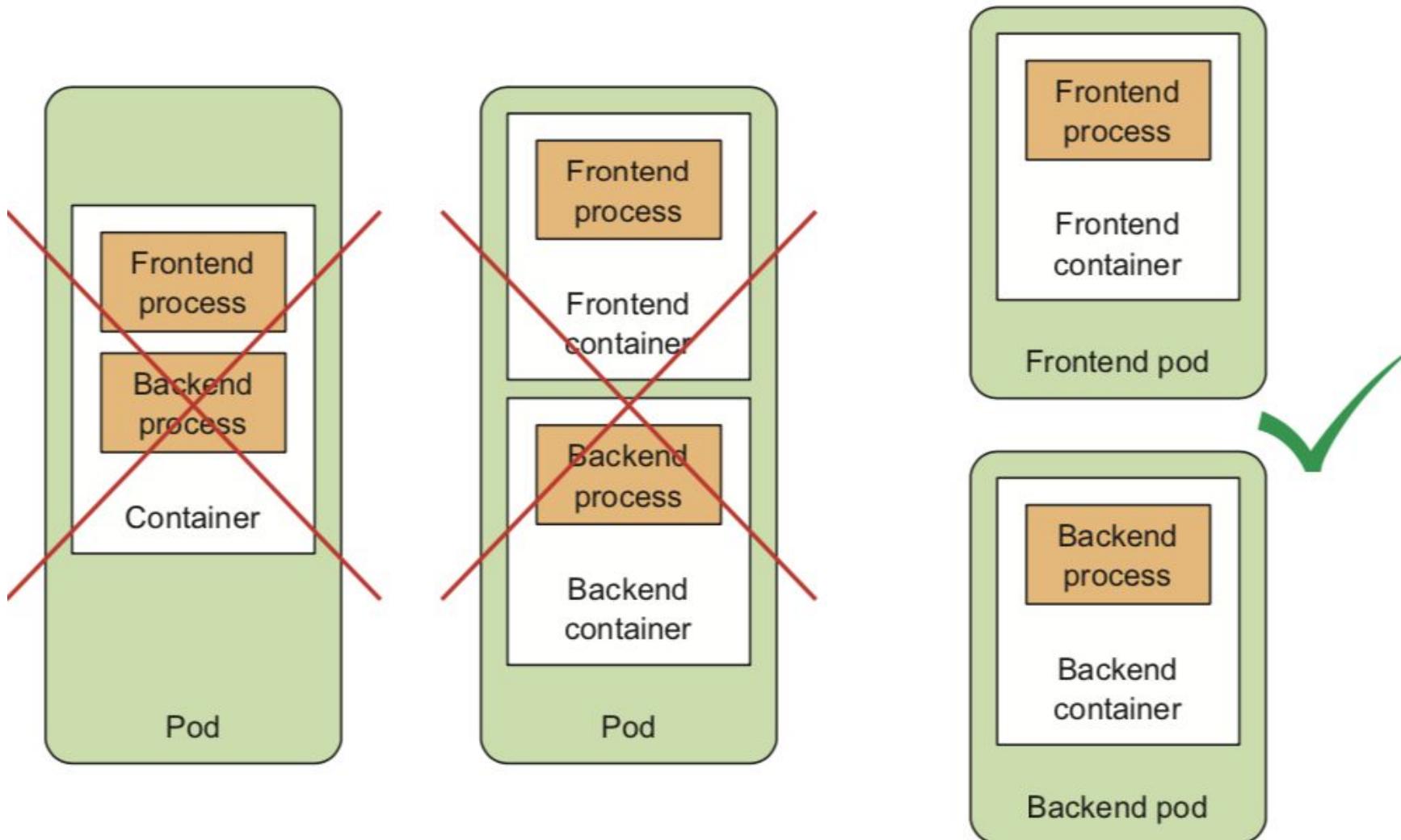
**Adapter**



**Ambassador**

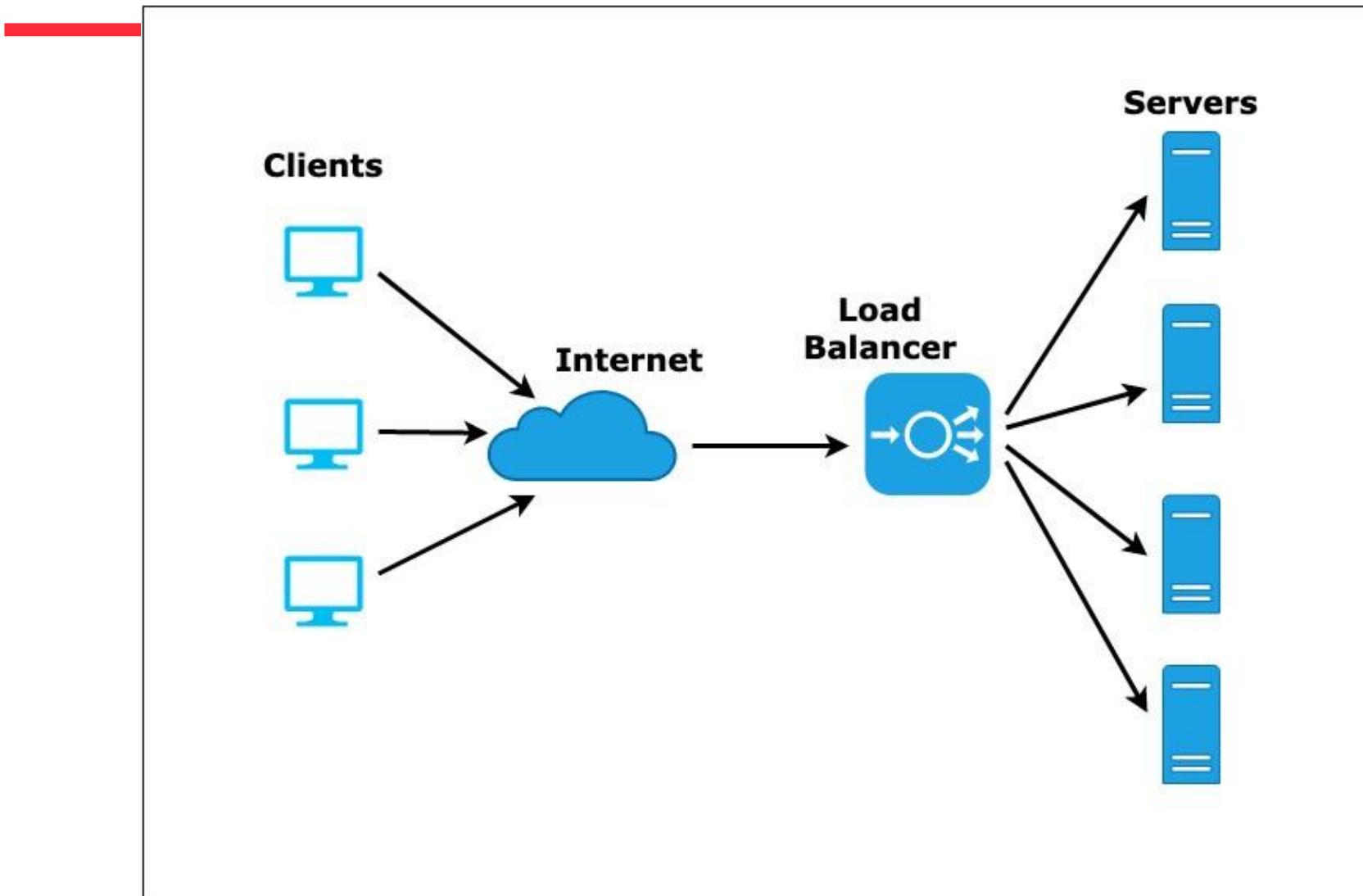


# Не надо так -- надо вот так!



# Масштабирование

# Масштабирование



# ReplicaSet

---

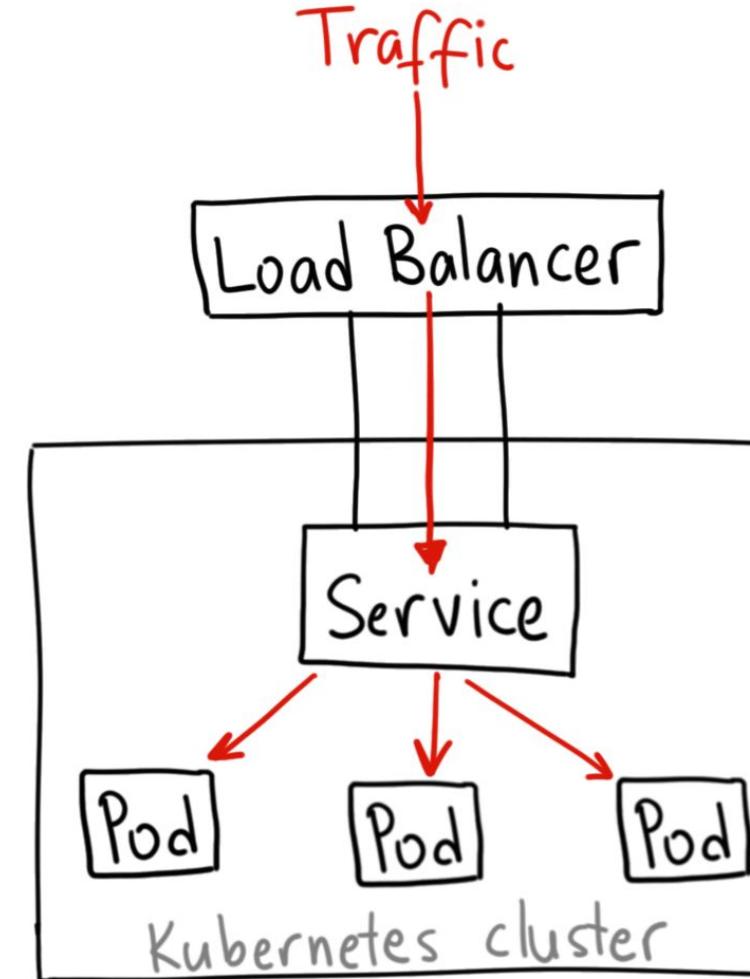
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-pod
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: my-pod
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx:1.14
          name: nginx
      ports:
        - containerPort: 80
```

# DEMO: работа с replicaset

# Service

---

Класс сущностей в кубере, предоставляющий возможность обращаться к подам не указывая конкретного названия.



# Service

```
apiVersion: v1
kind: Service
metadata:
  name: ml-service
spec:
  selector:
    app: fastapi-ml
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 8000
```

# Обновление

---

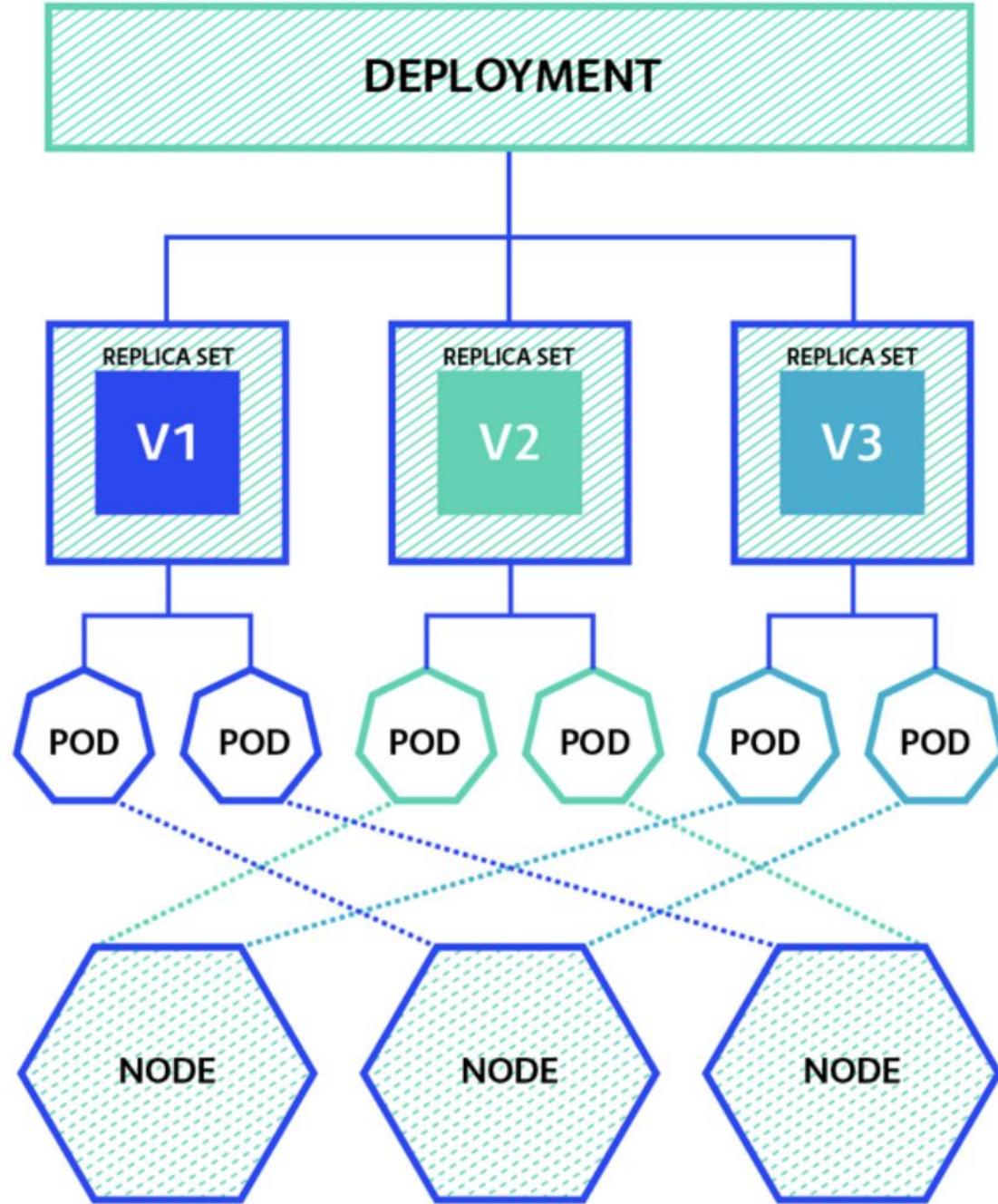


V1.1 -> V1.2

# Deployment

---

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-pod
  labels:
    app: nginx
spec:
  replicas: 4
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  selector:
    matchLabels:
      app: nginx
  template: <2 keys>
```



# DEMO: работа с deployment

# Job

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

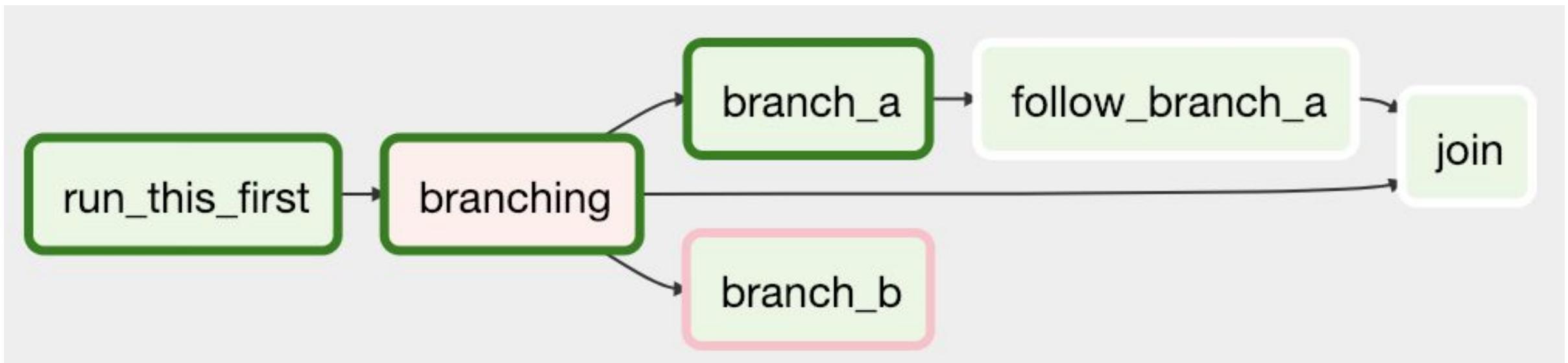
# CronJob

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox
              imagePullPolicy: IfNotPresent
              command:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster
  restartPolicy: OnFailure
```

# DEMO: запустим Job

# He DAG

---



Dag в k8s



**Kubeflow**

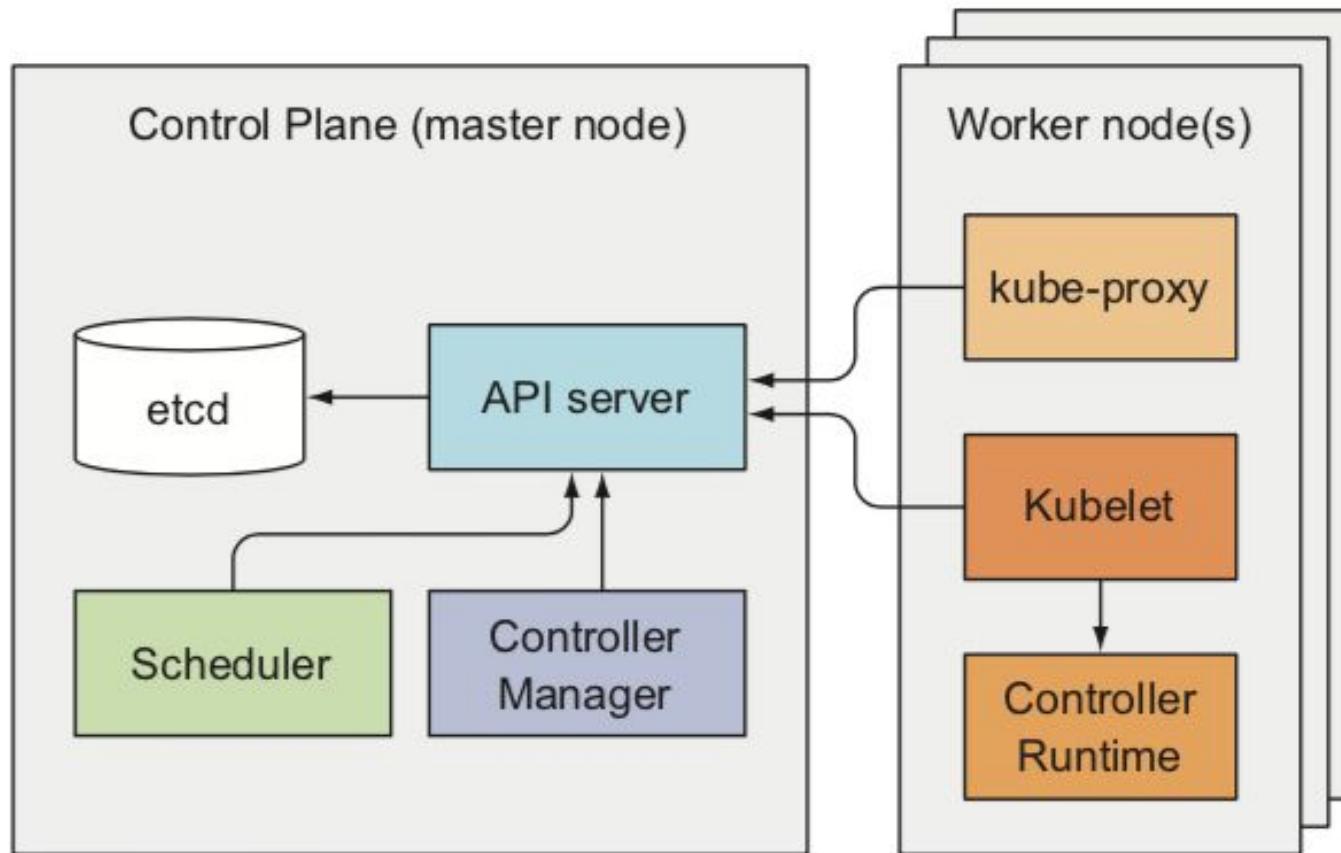


**argo**

# Внутренности kubernetes

# Архитектура kubernetes

- Control Plane ( в продакшен развертках их может быть несколько)
- Worker (их много by design)



# ETCD

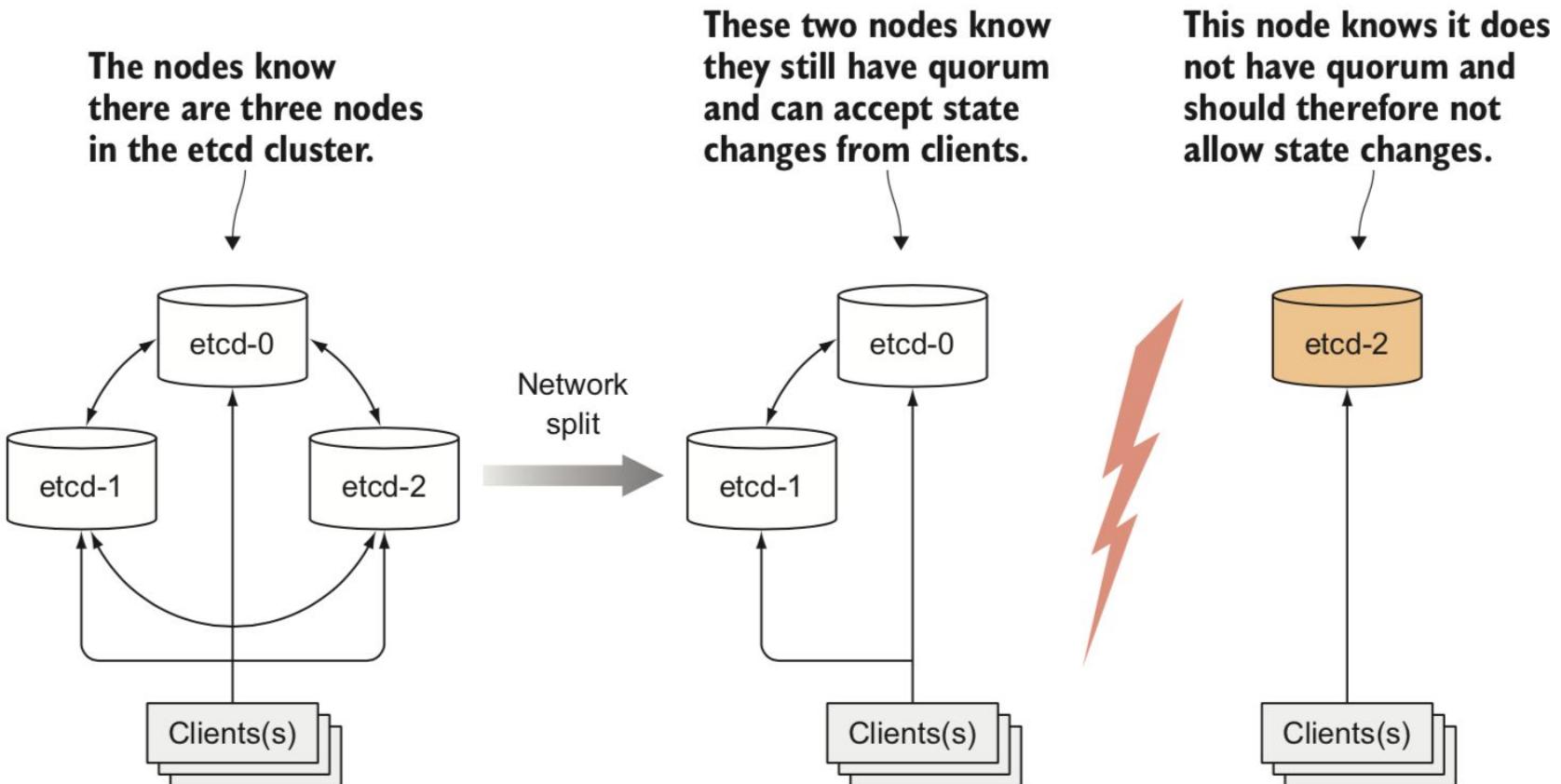
---

- Распределенное key-value хранилище
- Опирается на имплементацию **Raft Consensus Algorithm**
- Хранит состояние кластера



# ETCD

Допускается выход из строя  $(N-1)/2$  участников





# kube-apiserver

---

- API-шлюз - точка входа для всех взаимодействий компонент Kubernetes
- Предоставляет REST-сервис (работа над состояниями объектов)
- Проверяет и конфигурирует API-объекты (pods, services, ...)
- Сохраняет состояние в etcd (только он общается с etcd)



# kube-controller-manager

---

Отслеживает через API изменение состояний объектов

Набор различных контроллеров:

- ReplicaSet controller
- Endpoints controller
- Namespace controller



# kube-scheduler

---

- Учет имеющихся ресурсов
- Учет требований к ресурсам
- Учет ограничений по ресурсам (affinity, anti-affinity)
- Решение где запустить контейнеры (на основе ограничений и требований)



# kubelet

---

- Общается с API-Server
- Разворачивает Pods согласно PodSpec (описание Pod в формате YAML или JSON)

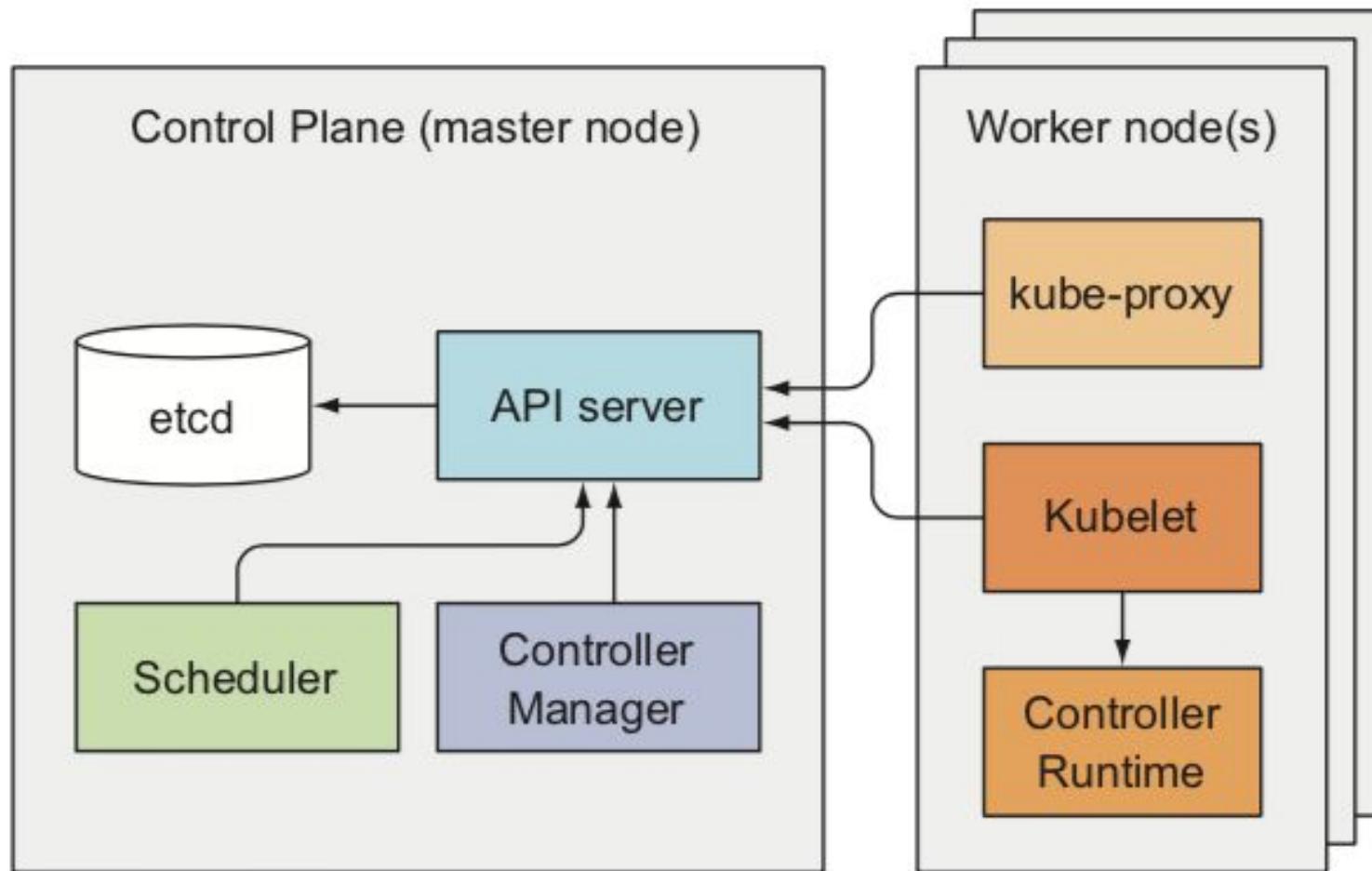
# kube-proxy

---

- Проксирует TCP и UDP (не HTTP)
- Используется для работы с сервисами
- Обеспечивает внутреннюю балансировку

# Архитектура kubernetes

kubectl apply -f my-deployment.y





# Как развернуть kubernetes?

---

- cloud providers
- baremetal (kubeadm, kubespray)
- локальные инсталляции
  - minikube
  - kind



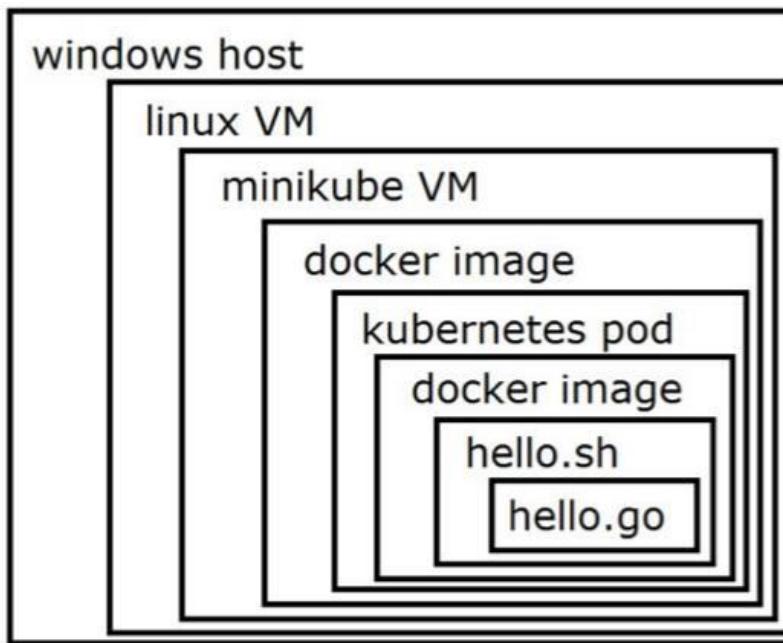
Табличка «Сарказм»  
@glorphyndale

...

---

Помоги Даше-разработчице  
понять, на каком уровне протекает  
абстракция

[Translate Tweet](#)



# Kubernetes in action

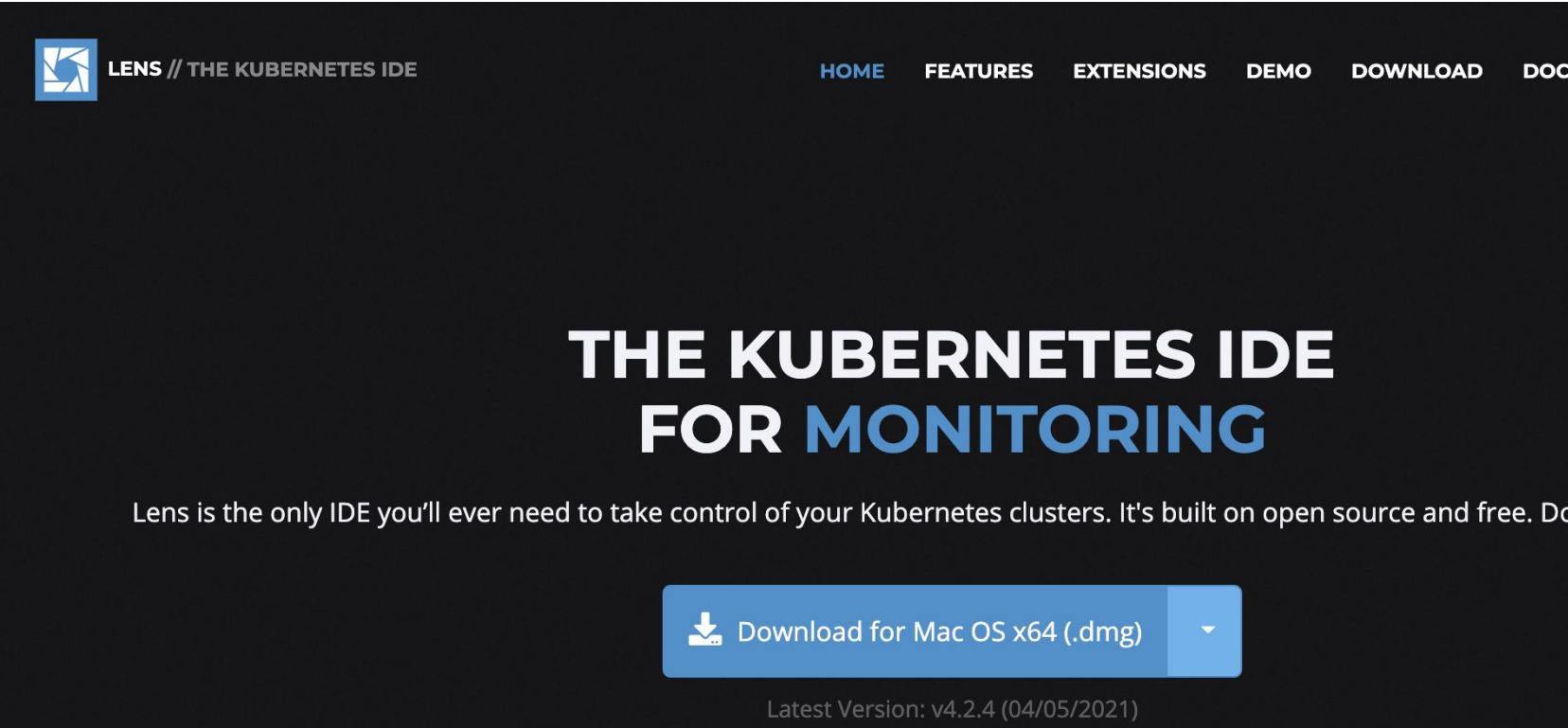
---



## Kubernetes IN ACTION

Marko Lukša

# Lens



The screenshot shows the official website for Lens, a Kubernetes IDE. The header features a red diagonal bar on the left and a red horizontal bar below it. The main navigation bar includes links for HOME, FEATURES, EXTENSIONS, DEMO, DOWNLOAD, and DOC. The title "LENS // THE KUBERNETES IDE" is displayed next to a blue square icon. The main heading "THE KUBERNETES IDE FOR MONITORING" is prominently displayed in white and blue text. Below the heading, a paragraph describes Lens as the only IDE for Kubernetes monitoring. A large blue "Download" button is centered, with a sub-link for "Download for Mac OS x64 (.dmg)". At the bottom, it indicates the "Latest Version: v4.2.4 (04/05/2021)".

LENS // THE KUBERNETES IDE

HOME FEATURES EXTENSIONS DEMO DOWNLOAD DOC

# THE KUBERNETES IDE FOR MONITORING

Lens is the only IDE you'll ever need to take control of your Kubernetes clusters. It's built on open source and free. Download now.

[Download for Mac OS x64 \(.dmg\)](#)

Latest Version: v4.2.4 (04/05/2021)

<https://k8slens.dev/>



# Нужно ли разработчику знать kubernetes?

---

- 1) Разработчик должен иметь возможность релизить свои приложения
- 2) Никто кроме разработчика не поймет, почему оно не запускается, нужно дебажить
- 3) Никто не знает сколько ресурсов оно потребляет — нужна возможность прописать

Правильное использование k8s дает разработчику все эти возможности



# Содержание занятия

---

- Зачем нужна контейнерная оркестрация
- Что такое Kubernetes
- Основные сущности Kubernetes
- Запускаем ML модель в kubernetes(в процессе)
- Внутреннее устройство

<https://forms.gle/wD66dFJUFxN8mjYe9>

академия  
больших  
данных



# Kubernetes

Михаил Марюфич, MLE

