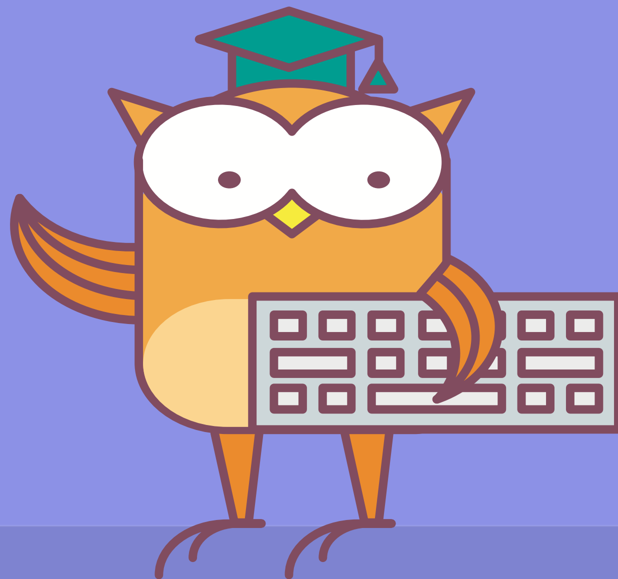
ОНЛАЙН-ОБРАЗОВАНИЕ

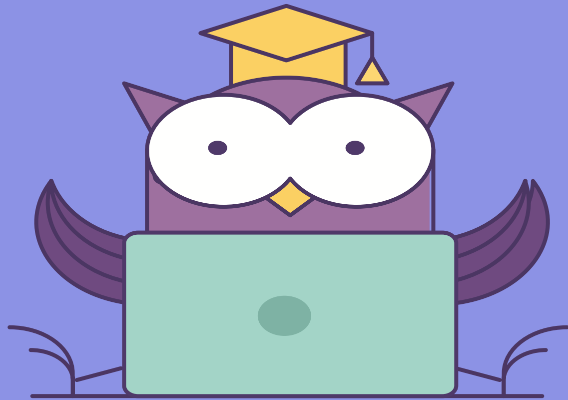
OTUS

Kubernetes



Дроздецкий Владимир

Меня хорошо слышно && видно?

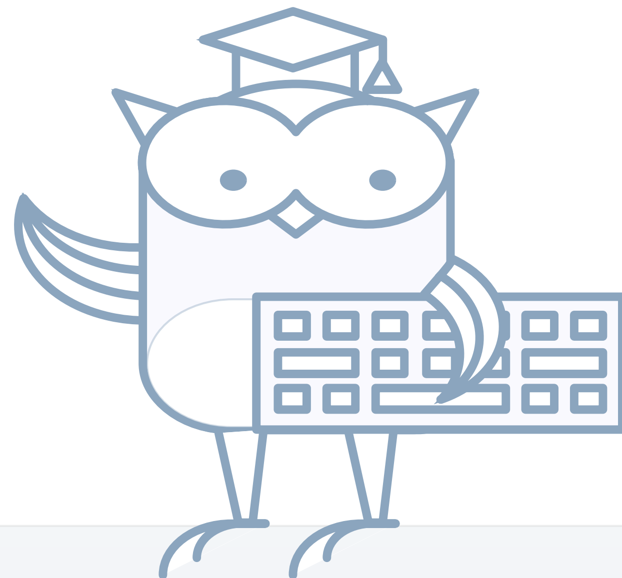


Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

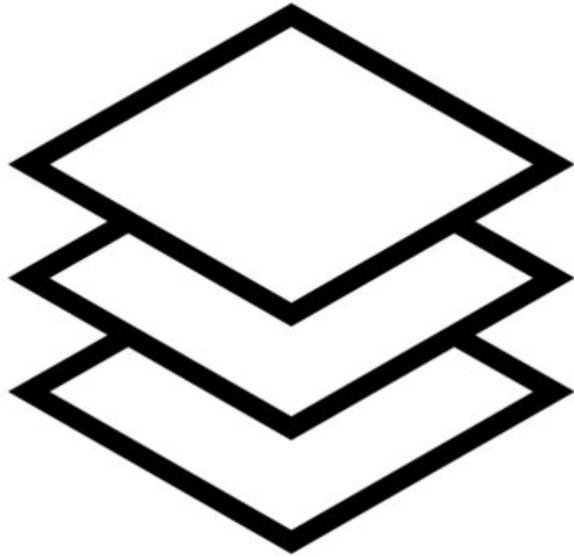
OTUS

1. Оркестрация
2. Виды оркестраторов
3. Kubernetes



Дроздецкий Владимир

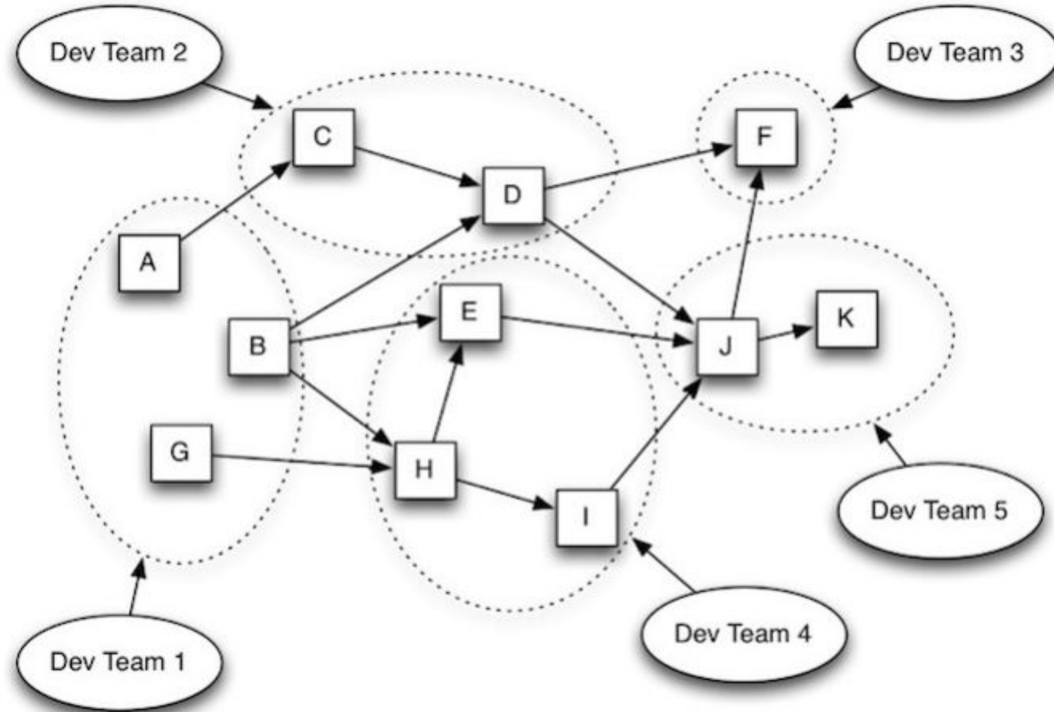
OTUS

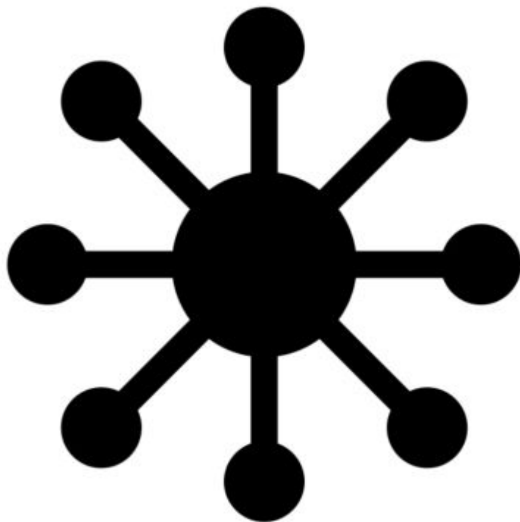


Устройство проекта:

- Несколько сервисов/приложений
- Просто деплоить и управлять конфигурацией
- Инженеры принимают решение куда деплоить
- Инженеры хранят знания о том, сколько ресурсов нужно

Kubernetes





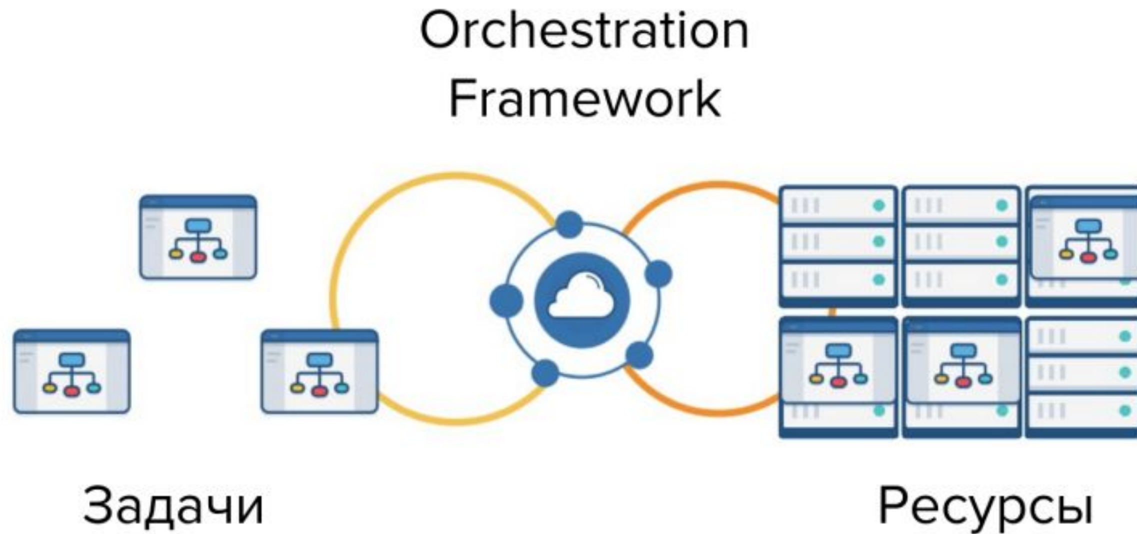
Поддержка

- Рассчитать потребление ресурсов сложно
- Ручная аллокация приложений становится проблемой
- Деплой становится рутиной

Kubernetes

OTUS

- Управление кластером хостов
- Планирование и распределение задач
- Автоматизация



Когда?

- Есть больше 1-го приложения
- Гибкое управление большим количеством ресурсов
- Нужен контроль над состоянием

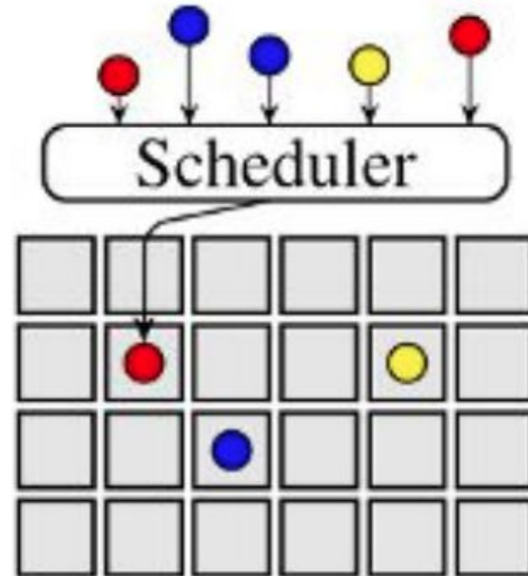
Чем занимается планировщик?

- Выделение (Аллокация) ресурсов для запуска задачи
- Дать все необходимое для запуска задачи
- Контроль ресурсов
- Контроль состояния задач
- Реакции на изменения состояния задач

OTUS

Монолитное планирование (`kube-scheduler`, `Borg`)

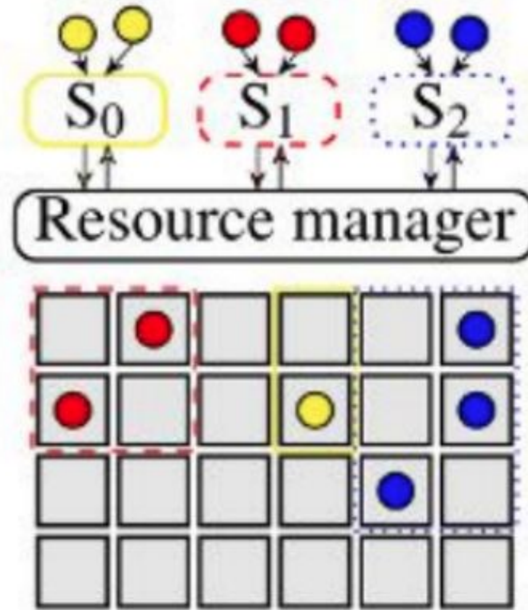
- Планировщик сам выполняет оценку доступных ресурсов
- Технически простое решение
- Могут возникать задержки планирования (*head-of-line blocking*), особенно для batch-заданий



OTUS

Двухуровневое планирование (Mesos + Marathon)

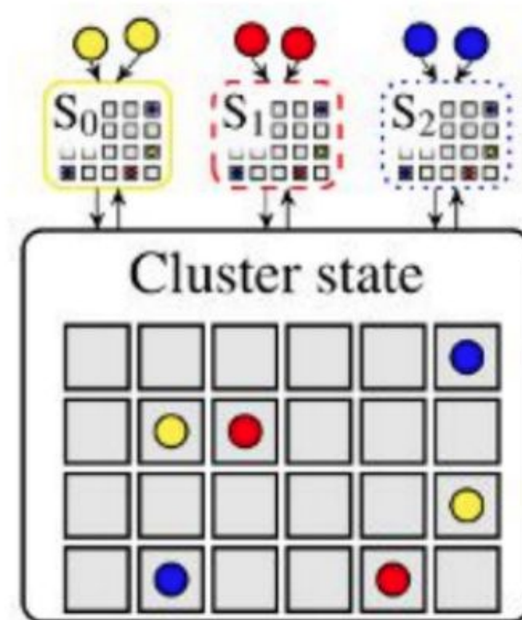
- Существуют отдельные сущности: планировщик ресурсов и планировщик задач
- Легко добавлять новые типы планировщиков
- Ресурсы кластера могут быть недоиспользованы



OTUS

Shared-state архитектура (Nomad, Google Omega)

- Каждый планировщик независимо выполняет выделение ресурсов, используя общее хранилище состояния кластера
- После выделения ресурсов он коммитит транзакцию задачу в общий стейт
- Есть шанс множественного Split Brain и затраты на разрешение конфликтов



Kubernetes

OTUS

- Управление кластером
- Планирование и распределение задач
- Автоматизация
- Управление сервисами для контейнеров
  - Service Discovery
  - Load balancing, networking
  - Storages
  - ...

- Память
- CPU
- GPU
- Хранение данных
  - локальные хранилища
  - удаленные хранилища
- Сетевые ресурсы
  - Порты
  - Управление IP адресами
  - Управление туннелями и overlay-сетями

- Расположение (bin packing, affinity rules)
- Реплицирование/масштабирование
- Проверки готовности/жизнеспособности
- Воскрешение
- Перепланирование (rescheduling)
- Cron/batch jobs
- Запуск демонов

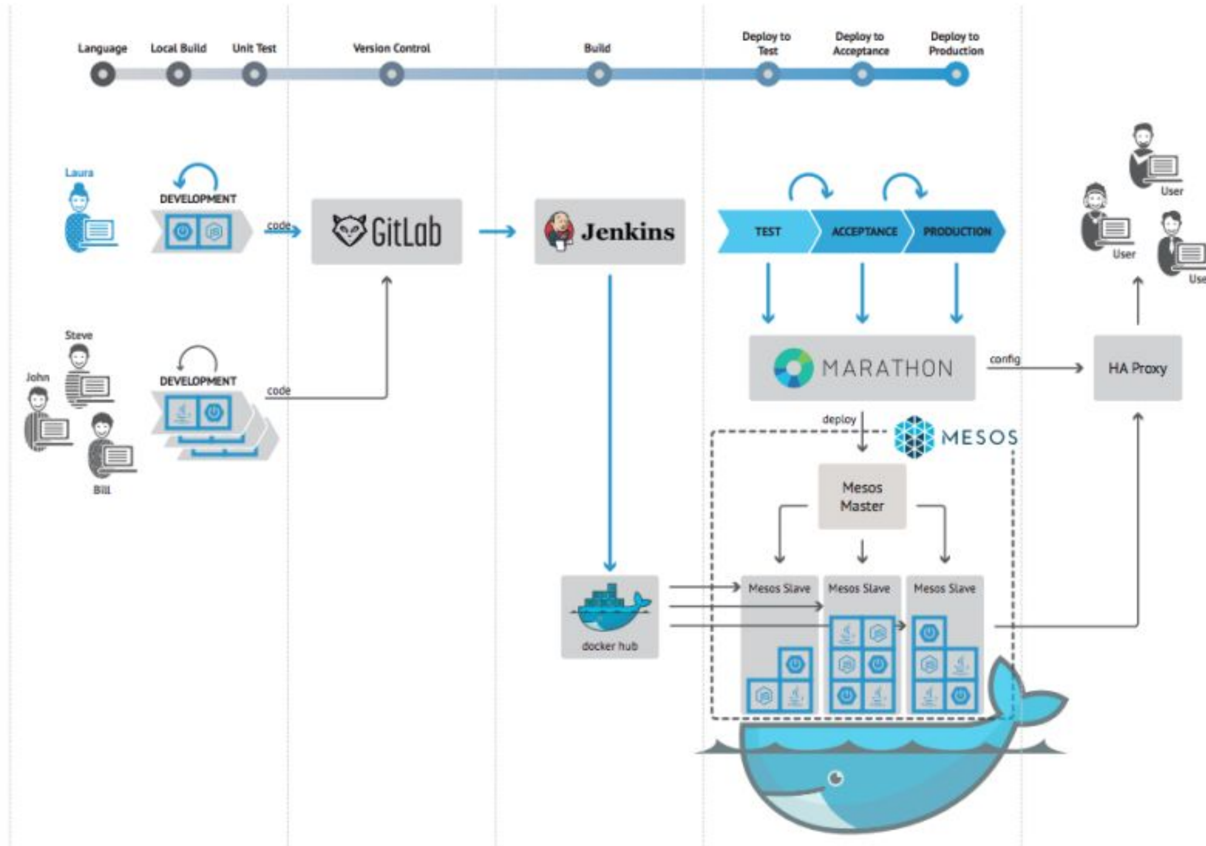
Kubernetes

OTUS

- Метки
- Группы
- Зависимости
- Балансировка нагрузки
- Virtual IP
- DNS и регистрация в Service Discovery
- Управление секретами
- Хранение и шаблонизация конфигурационных файлов

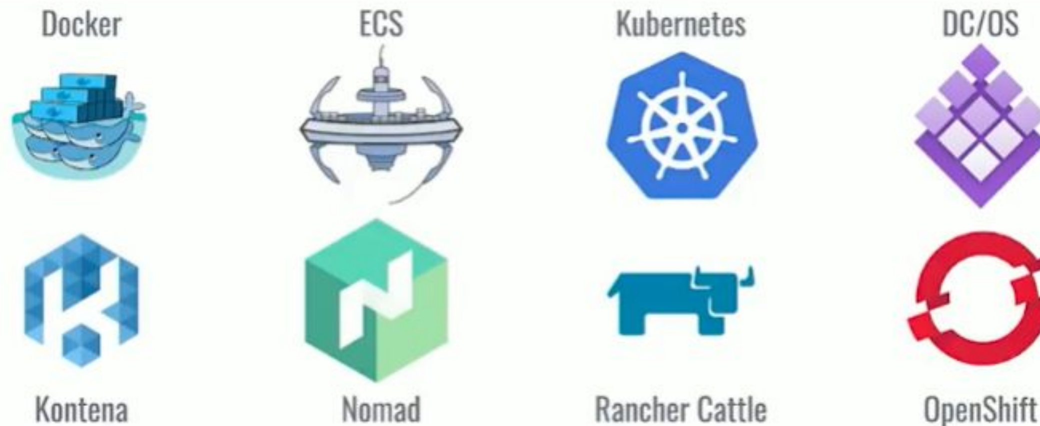
Kubernetes

OTUS



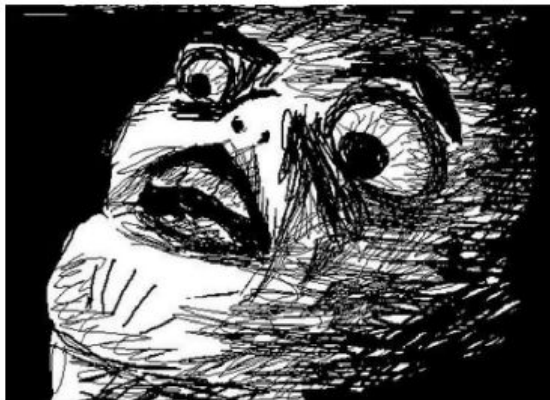
Kubernetes

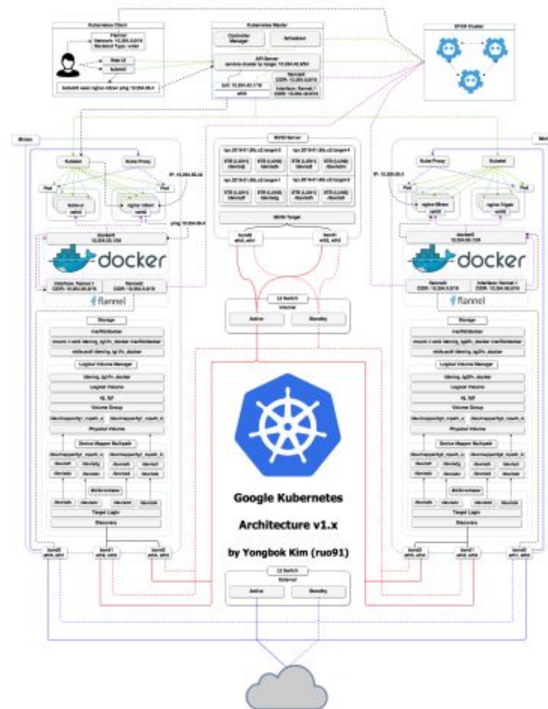
OTUS



У большинства под капотом - Kubernetes. И почти все копируют архитектуру Google Borg/Omega.

Что видит человек впервые?



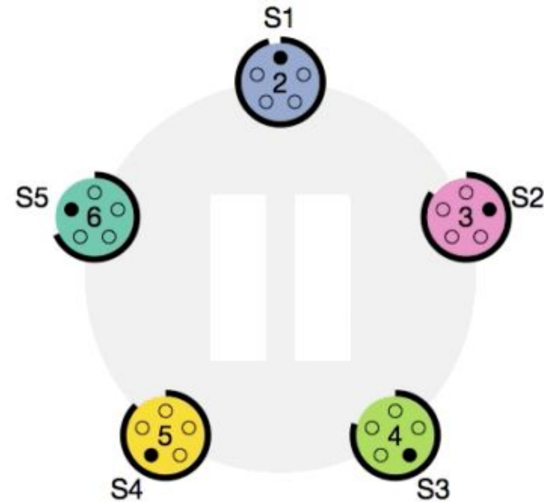


Kubernetes

OTUS

- Используют общее распределенное консистентное хранилище, обычно на основе алгоритмов Raft/Paxos
- Спецификация задания описывает целевое состояние (декларативна)
- Основной процесс кластера - *reconciliation loop*, цикл обработки событий в кластере (состояние рабочих нод, изменение спецификации заданий) и запуска планировщика ресурсов
- Планировщик ресурсов является отдельной сущностью, используется для генерации плана изменений

- Алгоритм сходимости и консистентной репликации лога изменений
- Можно потерять (N-1)/2 участников ("меньшую половину")
- Нечетное число участников лучше
- Документ об алгоритме согласования доступен по ссылке



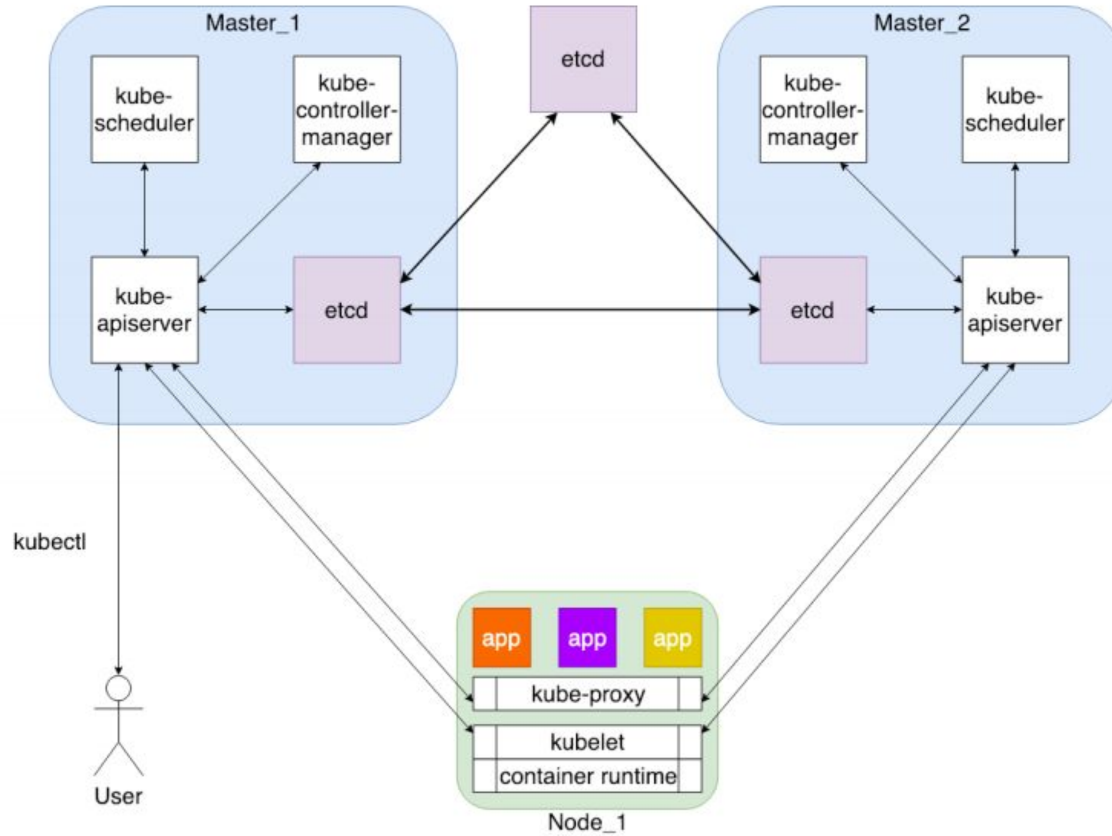
https://raft.github.io/

OTUS

- k8s
- Переосмысление проекта **Borg** от Google в контексте контейнеров
- **~57 000** звезд на GitHub
- **~82 000** 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

OTUS



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

OTUS

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

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)

- 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**)

Конфигурация kubectl - это **контекст** (context)

Контекст - комбинация:

- **cluster** - API сервера
- **user** - пользователь для подключения к кластеру
- **namespace** - область видимости (не обязательно, по умолчанию default)

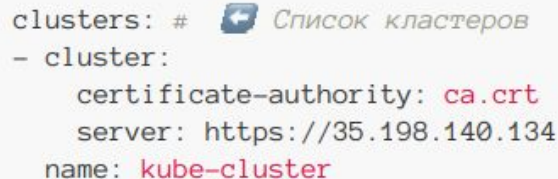
Информацию о контекстах kubectl сохраняет в файле ~/.kube/config

Kubernetes

OTUS

Кластер (cluster) - это:

- **server** - адрес kubernetes API-сервера
- **certificate-authority** - корневой сертификат (которым подписан SSL-сертификат API-сервера)
- **name** - имя для идентификации

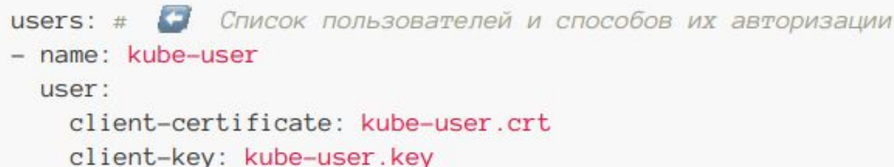


```
clusters: # Список кластеров
- cluster:
    certificate-authority: ca.crt
    server: https://35.198.140.134
  name: kube-cluster
```

OTUS

Пользователь (**user**) - это:

- Данные для аутентификации:
  - username + password (Basic Auth)
  - client key + client certificate
  - token
  - auth-provider config (например GCP)
- name (имя) для идентификации в конфиге



```
users: # ↩ Список пользователей и способов их авторизации
- name: kube-user
  user:
    client-certificate: kube-user.crt
    client-key: kube-user.key
```

Контекст (**context**) - это:

- **cluster** - имя кластера из списка clusters
- **user** - имя пользователя из списка users
- **namespace** - область видимости по-умолчанию (не обязательно)
- **name** - имя контекста для идентификации

```yaml
contexts:   # Список контекстов
- context:
    cluster: kube-cluster
    user: kube-user
  name: kube-context
```

Kubernetes

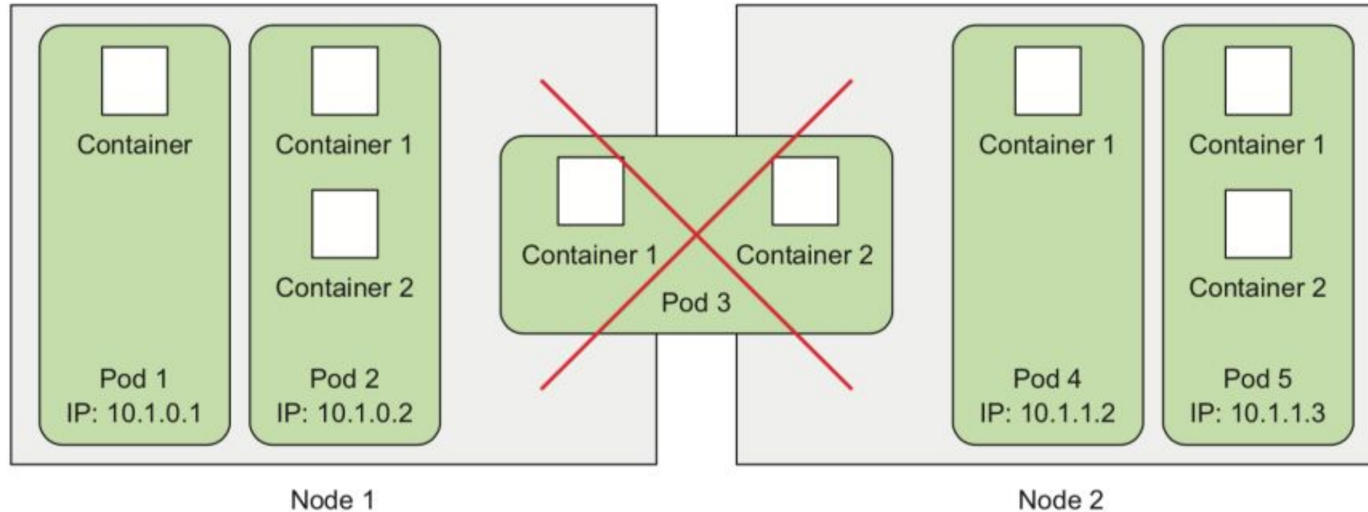
OTUS

```
apiVersion: v1
clusters:                      # ⮐ Список кластеров
- cluster:
    certificate-authority: ca.crt
    server: https://35.198.140.134
  name: kube-cluster
contexts:                      # ⮐ Список контекстов
- context:
    cluster: kube-cluster
    user: kube-user
  name: kube-context
current-context: kube-context # ⮐ Текущий контекст
kind: Config
preferences: {}
users:                         # ⮐ Список пользователей и способов их авторизации
- name: kube-user
  user:
    client-certificate: user.crt
    client-key: user.key
```

OTUS

- Группа контейнеров (один или несколько)
- Минимальная сущность, управляемая Kubernetes

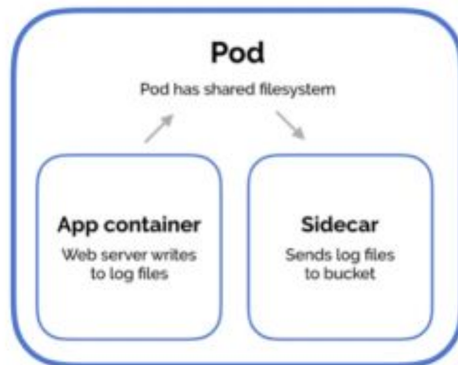
У всех контейнеров общие Network, IPC, UTS, PID* namespaces

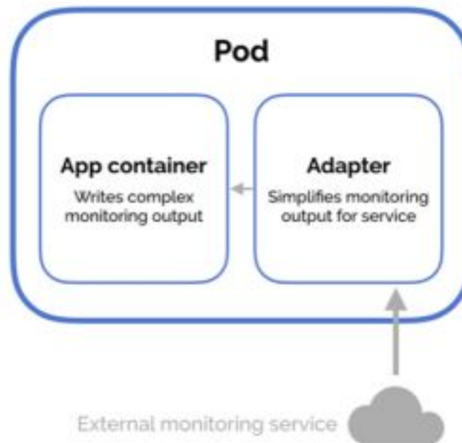


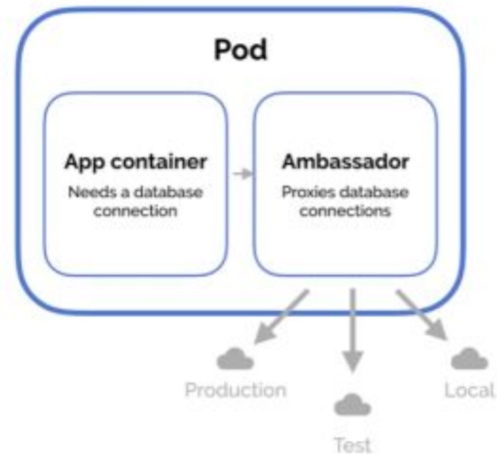
Контейнеры внутри одного Pod или разные Pod?

- Сервисы должны масштабироваться совместно или по отдельности?
- Должны ли сервисы быть запущены вместе или могут быть разнесены на разные хосты?
- Это связанные сервисы или независимые компоненты?

OTUS







YAML

- 1 файл может описывать много сущностей
- Обязательные поля:
  - apiVersion
  - kind
  - metadata
- Спецификация объекта

OTUS

prometheus-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: prometheus
spec:
  containers:
  - name: prometheus
    image: prom/prometheus:v2.10.0
```

OTUS

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)

kind

Тип объекта, который хотим создать

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: prometheus
spec:
  containers:
  - name: prometheus
    image: prom/prometheus:v2.10.0
```

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: monitoring
```

OTUS

apiVersion

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: prometheus
spec:
  containers:
  - name: prometheus
    image: prom/prometheus:v2.10.0
```

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob
spec:
-- omitted --
```

metadata

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

```
apiVersion: v1
kind: Pod
metadata:
  name: prometheus
  labels:
    app: prometheus
  namespace: monitoring
spec:
-- omitted --
```

Ограничения по ресурсам (CPU/Memory) для контейнеров

- **Requests** - сколько ресурсов нужно контейнеру для работы
- **Limits** - пороговые значения ресурсов, при превышении которых контейнер будет перезапущен*

Указание requests и limits для контейнеров - рекомендуемая практика работы с Kubernetes

Kubernetes

OTUS

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.15.8
    resources:
      requests:
        memory: "64Mi"
        cpu: "100m"
      limits:
        memory: "128Mi"
        cpu: "200m"
```

Спасибо за внимание!