

# Введение в Kubernetes День 2

Летняя школа ВШЭ 2024.





# Содержание

01 Kubernetes API

02 Pod vs контейнер

03 Основные  
объекты  
Kubernetes

04 Контроллеры





01

# Kubernetes API

# Что такое Kubernetes API?

Основной интерфейс  
для управления  
кластером Kubernetes.

Все взаимодействие с кластером  
происходит через API - создание,  
изменение и удаление объектов.

Каждый объект в Kubernetes  
представлен как ресурс API.



# Характеристики k8s API

Kubernetes API Server  
запущен на master-node

Ответственен за механизм  
авторизации и  
аутентификации

Stateless (каждый запрос от  
клиента к серверу  
обрабатывается независимо  
от предыдущих запросов)

Реализует RESTful API



# Основные понятия Kubernetes API

## Ресурсы

Это основные объекты, которые вы создаете и управляете в Kubernetes. Примеры ресурсов - Pods, Namespaces, Services.

## Типы ресурсов

Типы ресурсов определяют, какие ресурсы доступны в Kubernetes. Например, Pod — это тип ресурса, который представляет контейнерное приложение.

## Объект

Объект — это конкретный экземпляр типа ресурса. Когда вы создаете конкретный экземпляр ресурса (например, определенный Pod), Это называется объектом.



# Объекты Kubernetes API

Kubernetes объект — это "запись намерения" (record of intent).

После создания объекта система Kubernetes будет постоянно поддерживать его существование, обеспечивая желаемое состояние вашего кластера. Для работы с объектами Kubernetes — создания, изменения или удаления — необходимо использовать Kubernetes API

<https://kubernetes.io/docs/concepts/overview/working-with-objects/#understanding-kubernetes-objects>





# Объекты Kubernetes API

Объекты Kubernetes — это постоянные сущности в системе Kubernetes, представляющие состояние кластера, они описывают:

- Какие контейнерные приложения запущены и на каких узлах
- Ресурсы, доступные этим приложениям
- Политику поведения приложений, включая рестарты, обновления и отказоустойчивость.

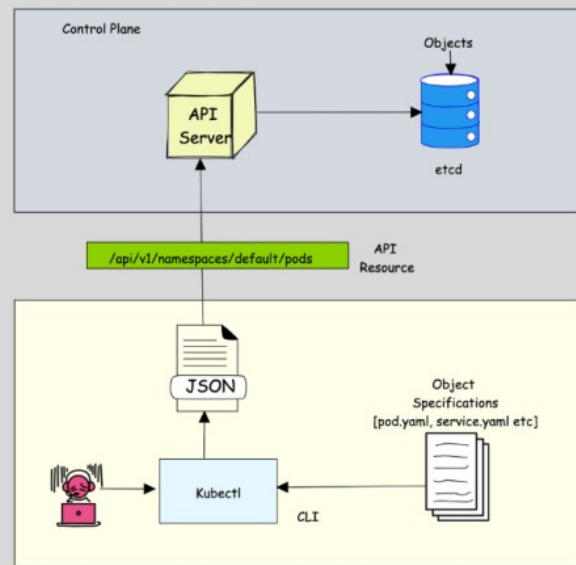
<https://kubernetes.io/docs/concepts/overview/working-with-objects/#understanding-kubernetes-objects>



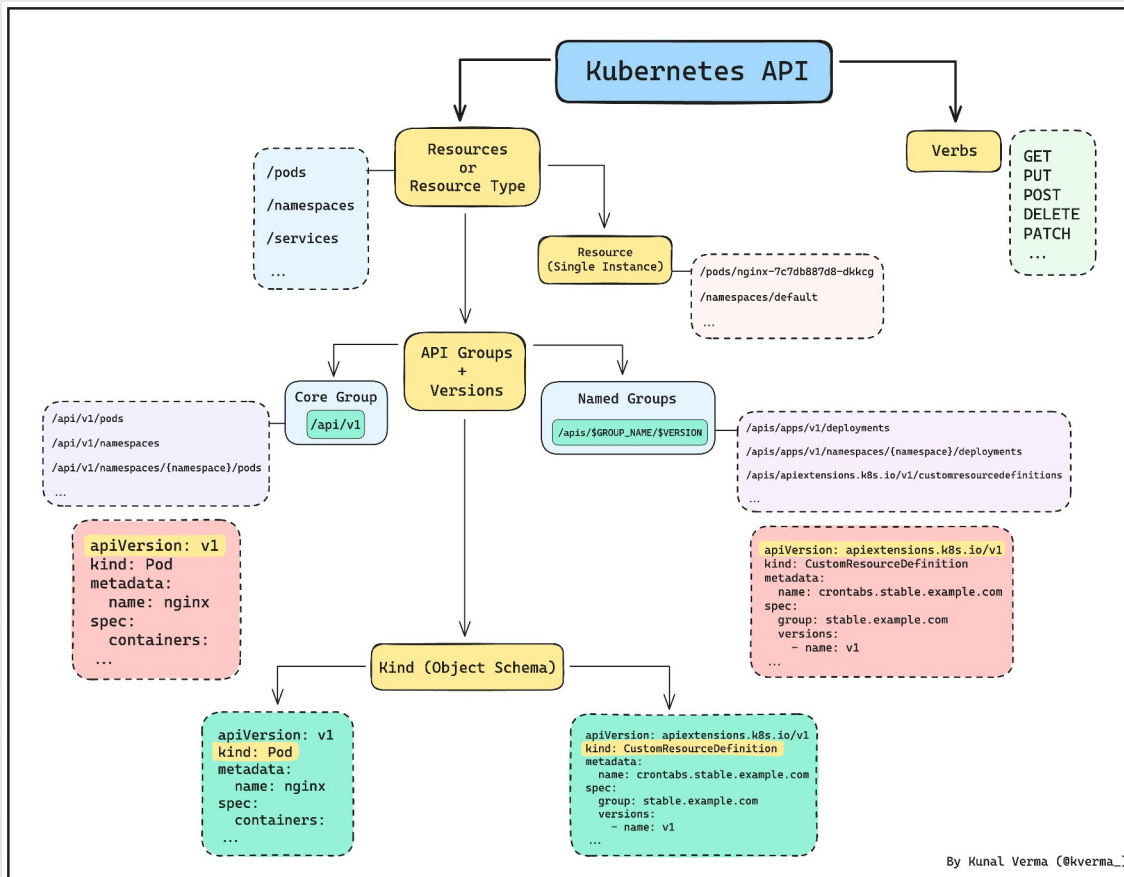


# Действия сервера API при вызове

1. Проверяет разрешение вызова (используя RBAC).
2. При необходимости изменяет данные вызова через mutating webhooks.
3. Валидирует данные вызова (с помощью внутренней проверки и валидирующих веб-хуков).
4. Сохраняет данные и возвращает ответ.
5. Уведомляет подписчиков об изменении объекта.

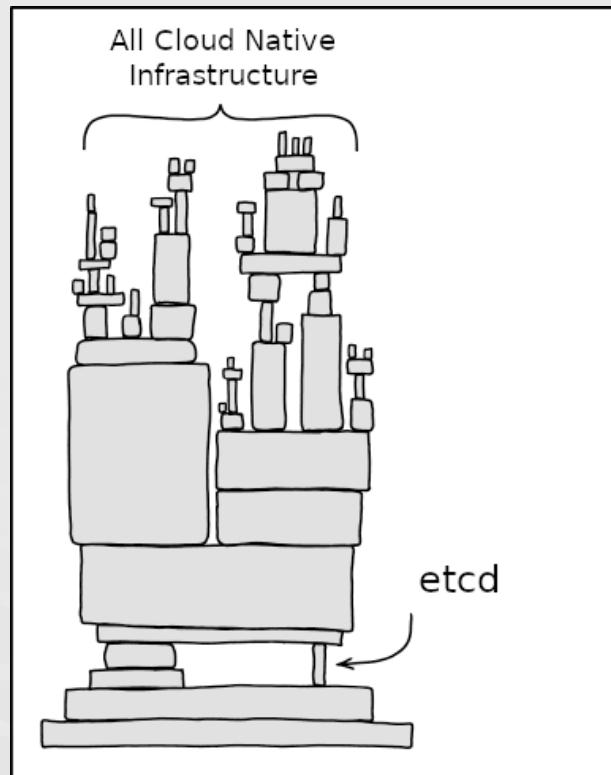


# Kubernetes API



# Где хранятся объекты API?

- Etcd - строго согласованное распределённое хранилище ключ-значение
- Часть control plane, размещается на управляющих узлах кластера
- Работает в нескольких экземплярах для отказоустойчивости.
- Хранит информацию о все объектах k8s.
- Взаимодействовать с etcd напрямую может только kube-api





02

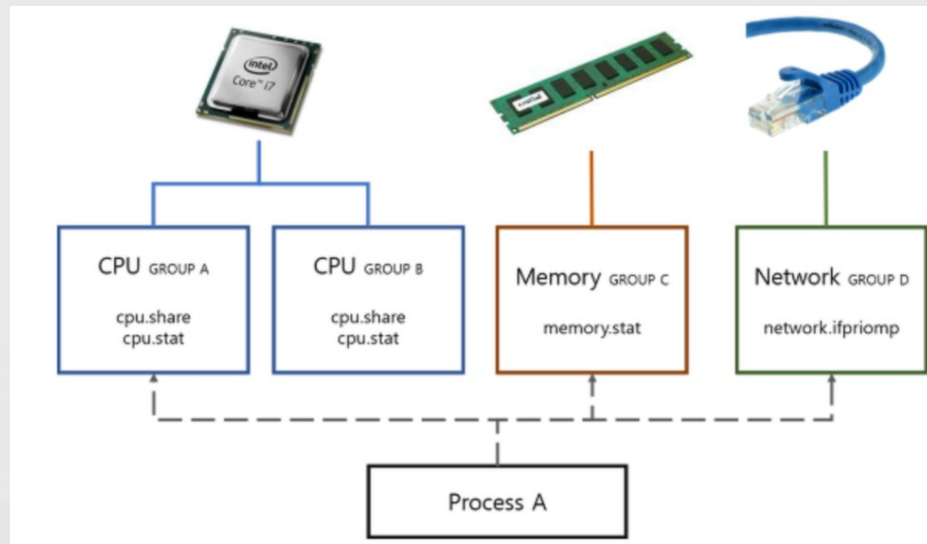
# Pod vs контейнер

# Пространства имен Linux

<b>Mount (mnt)</b>	Изолирует таблицы монтирования, позволяя контейнеру иметь свою файловую систему.
<b>UTS</b>	Изолирует имя хоста и доменное имя, позволяя контейнеру иметь свой <code>hostname</code> и <code>domainname</code> .
<b>IPC</b>	Изолирует межпроцессное взаимодействие, позволяя процессам внутри контейнера общаться только друг с другом через системные механизмы IPC.
<b>PID</b>	Изолирует идентификаторы процессов, позволяя контейнеру видеть только свои процессы.
<b>Network (net)</b>	Изолирует сетевые интерфейсы, IP-протоколы и номера портов, предоставляя контейнеру свою сетевую стеки и устройства.
<b>Cgroup</b>	Изолирует видимость файловой системы <code>cgroup</code> , позволяя контейнеру видеть только свои контрольные группы.
<b>User</b>	Изолирует идентификаторы пользователей, позволяя процессам внутри контейнера иметь свои UID и GID, которые могут отличаться от реальных идентификаторов в хост-системе.

# Cgroups

cgroups (control groups) — это механизм Linux, который позволяет управлять и ограничивать ресурсы (процессорное время, память, диск, сеть) для групп процессов. cgroups предоставляют возможность контролировать, сколько ресурсов может использовать каждый процесс или группа процессов





Исследуем контейнеры – демо.



# Исследуем контейнеры

```
docker run --name foo --rm -d --memory='512MB' --cpus='0.5' nginx:alpine
```

```
eu@serv1:~$ sudo lsns | grep nginx
```

```
4026532691 mnt      3  2206 root
4026532692 uts      3  2206 root
4026532693 ipc      3  2206 root
4026532694 pid      3  2206 root
4026532695 net      3  2206 root
4026532751 cgroup   3  2206 root
```

```
nginx: master process nginx -g daemon off;
nginx: master process nginx -g daemon off;
nginx: master process nginx -g daemon off;
nginx: master process nginx -g daemon off;
nginx: master process nginx -g daemon off;
nginx: master process nginx -g daemon off;
```

```
eu@serv1:~$ sudo systemd-cgls --no-pager
```

```
CGroup /:
```

```
-.slice
```

```
└─docker-8ba5323bf4037356e959f7d93422ac0b04ff7099d850c1af636490d1755c8cd9.scope
```

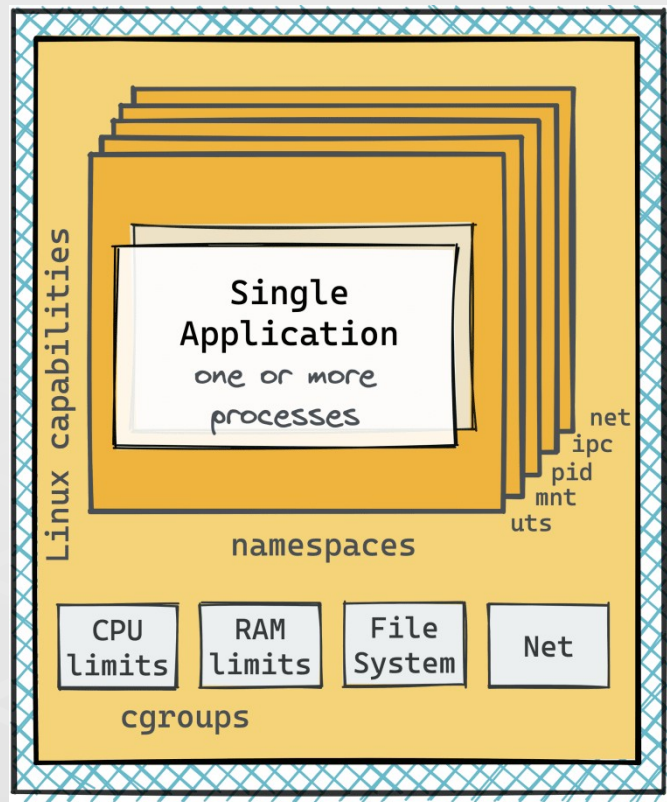
```
├─2206 nginx: master process nginx -g daemon off;
├─2251 nginx: worker process
└─2252 nginx: worker process
```



# Схема контейнера

После создания контейнер получает:

- mnt
- ipc
- ipc
- Pid
- Net
- cgroup



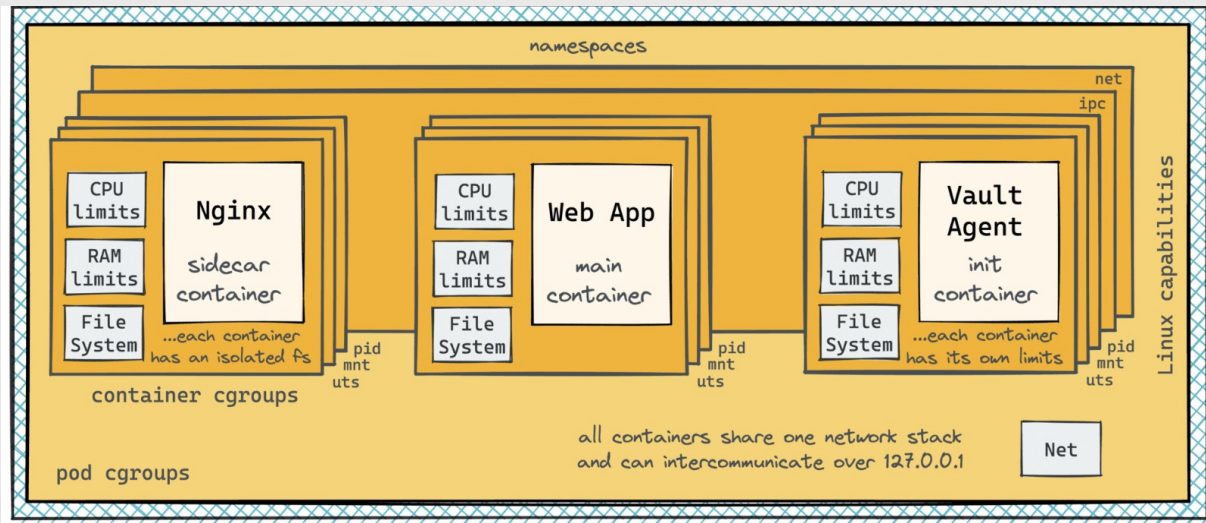


Исследуем поды – демо.



# Схема пода

- При создании пода сначала запускается pause контейнер, для которого создаются неймспейсы: net, mnt, uts, ipc, pid
- Затем контейнеры запускаются внутри пода и получают только 3 неймспейса: mnt, pid, cgroup
- Оставшиеся 3 неймспейса (net, uts, ipc) контейнеры делят между собой.





# Исследуем поды

Поэтому контейнеры в поде могут:

- Общаться по сети
- Общаться по IPC
- Иметь общий домен и hostname



# Сравнение

## Контейнер

Легковесная единица выполнения приложений

Имеет собственную изолированную файловую систему, сеть и процессы

Изолированная среда для выполнения одного или нескольких процессов

Запускается в рамках одного неймспейса

## Pod

Основная сущность Kubernetes, содержащая один или несколько контейнеров

Разделяет общие неймспейсы (сеть, IPC, UTS) между контейнерами

Контейнеры в Pod'е работают как единое целое, имея общий IP-адрес и ресурсы

Поддерживает несколько контейнеров, которые могут взаимодействовать друг с другом



Вопросы.





03

# Основные объекты Kubernetes

# Описываем Pod

apiVersion: Версия API Kubernetes, здесь v1.

kind: Тип создаваемого объекта, в данном случае Pod.

metadata:

name: Уникальное имя Pod'a в namespace.

labels: Метки для группировки и фильтрации объектов.

spec:

containers:

- name: Имя контейнера внутри Pod'a.

image: Образ контейнера, здесь nginx:latest.

ports:

- containerPort: Порт, доступный внутри контейнера (здесь 80).

```
apiVersion: v1
kind: Pod
metadata:
  name: name_of_the_pod
  labels:
    app: my-app
spec:
  containers:
    - name: nginx-container
      image: nginx:latest
      ports:
        - containerPort: 80
```



В реальности одиночные  
размещения объектов `rod`  
используются довольно редко!  
`Rod` – являются «кирпичиками»  
для многих других объектов API.

# Почему нельзя использовать просто pod?

Pod — лишь базовый строительный блок

Pod не может масштабироваться в ответ на увеличенную нагрузку.

Любые изменения требуют ручного обновления или перезапуска Pod'ов.

Если Pod падает, он не восстанавливается автоматически.

# Для чего нужен Lifecycle Management?

## Автоматизация процессов

Объекты такие как ReplicaSet или Deployment, автоматизируют создание, масштабирование и обновление Pod'ов.

## Обеспечение доступности

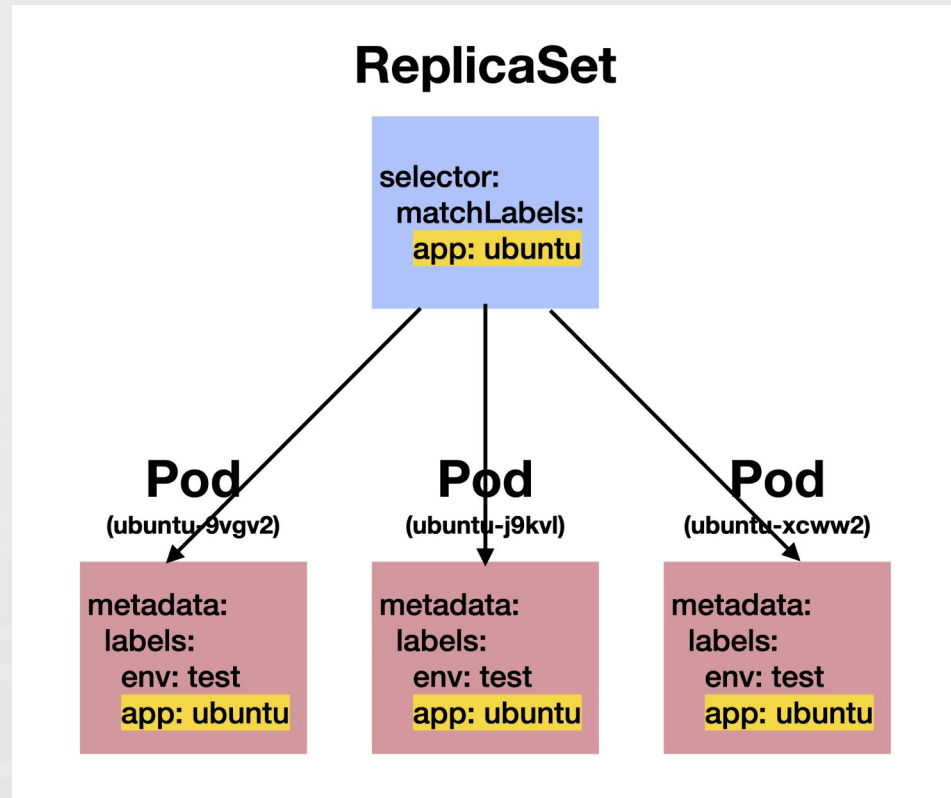
Deployment управляет обновлениями и откатами, поддерживая работоспособность приложения

## Масштабируемость и отказоустойчивость

Управление жизненным циклом Pod'ов позволяет Kubernetes поддерживать стабильность приложений даже при изменении нагрузки.

# ReplicaSet

Гарантирует доступность  
определенного числа идентичных  
Pod'ов, что позволяет обеспечить  
отказоустойчивость и  
масштабируемость приложений:



# Основные функции ReplicaSet

## Поддержание количества реплик

ReplicaSet следит за тем, чтобы всегда было запущено заданное количество Pod'ов. Например, если заданы 5 реплик, и один из Pod'ов выходит из строя, ReplicaSet создаст новый Pod, чтобы снова достичь требуемого количества

## Обеспечение отказоустойчивости

Если узел в кластере выходит из строя, и Pod, запущенный на этом узле, становится недоступен, ReplicaSet автоматически восстановит этот Pod на другом доступном узле.

## Масштабируемость

При увеличении нагрузки на приложение можно легко увеличить количество реплик, изменив спецификацию ReplicaSet. ReplicaSet создаст дополнительные Pod'ы для обработки увеличенной нагрузки.

# ReplicaSet

Number of pods (replicas)

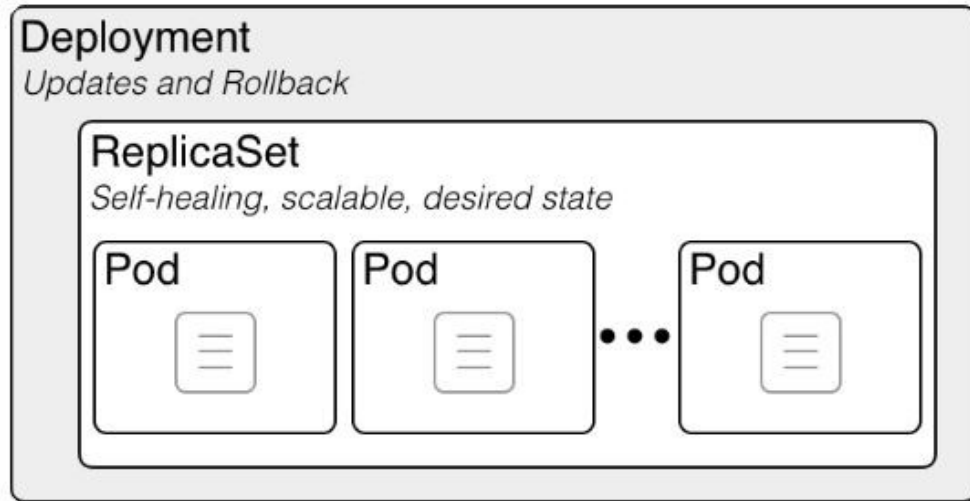
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: hello
  labels:
    app: hello
spec:
  replicas: 5
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
      - name: hello-container
        image: busybox
        command: [ ... ]
```

Which pods to watch?

Pod template to use

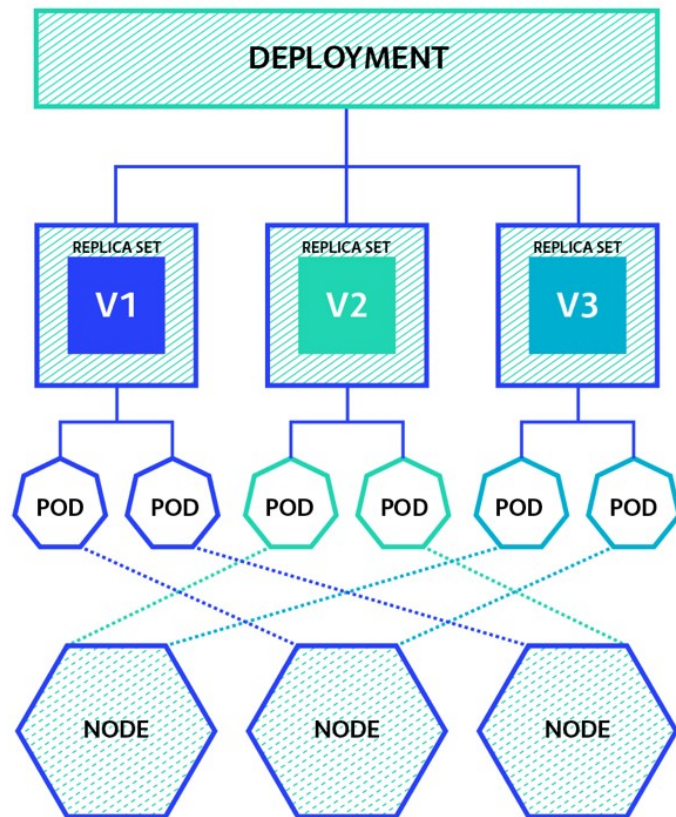
# Deployment

- Deployment — это основной объект управления состоянием приложений в Kubernetes.
- Обеспечивает управление жизненным циклом приложений - развертывание, обновление, откат, и масштабирование
- Поддерживает высокую доступность приложений за счет автоматического контроля за состоянием Pod'ов.



# Как Deployment расширяет ReplicaSet?

- Управление версиями - Deployment ведет историю изменений и позволяет возвращаться к предыдущим состояниям.
- Поддержка обновлений - в отличие от ReplicaSet, Deployment может управлять сложными процессами обновления.
- Откаты - автоматический возврат к предыдущей версии в случае неудачного обновления.





# Deployment

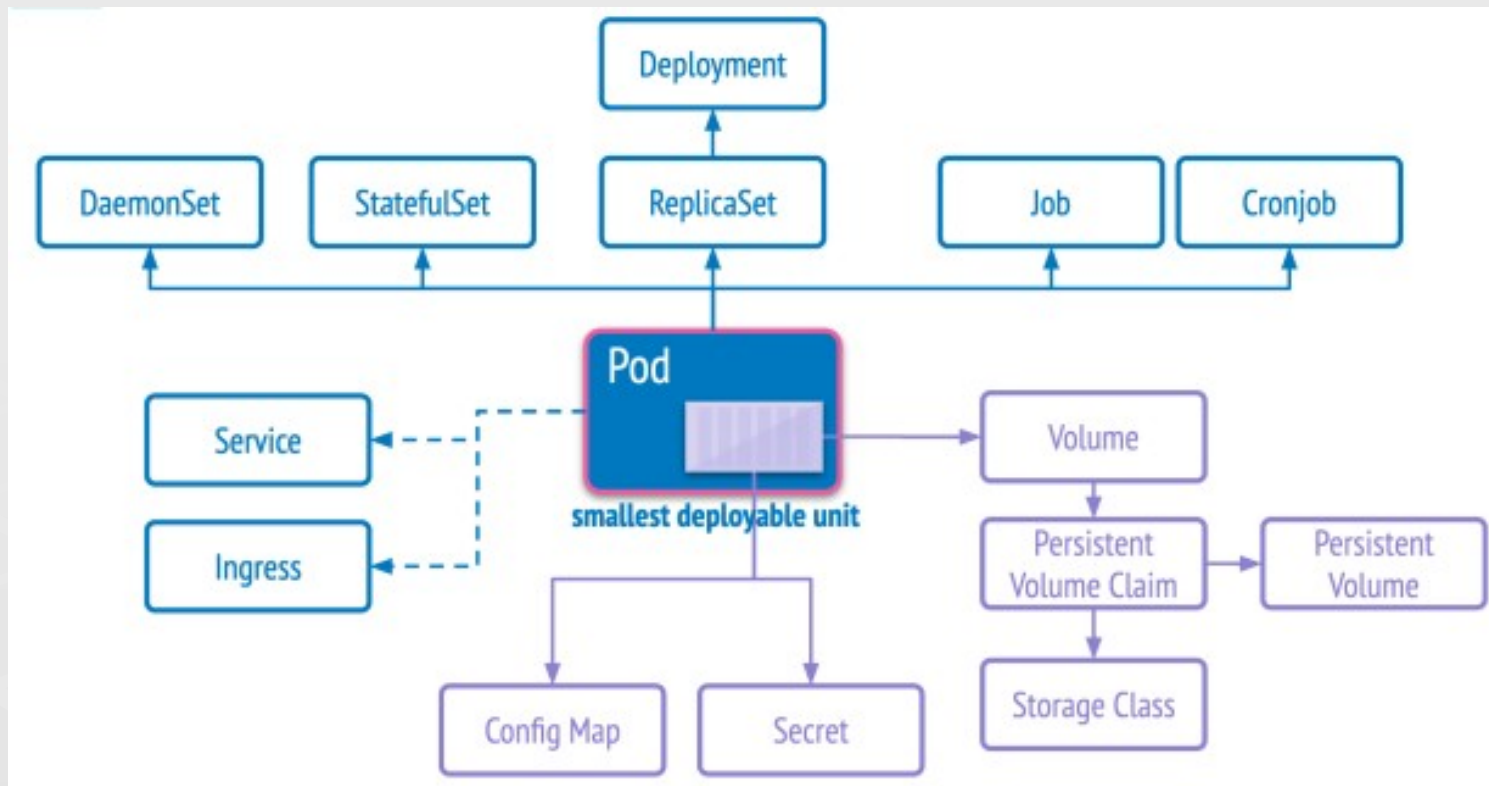
```
apiVersion: apps/v1 # Версия API для создания Deployment.
kind: Deployment # Определяет тип объекта, который создается.
metadata:
  name: nginx-deployment # Имя Deployment.
  labels:
    app: nginx # Метка ресурса.
spec:
  replicas: 3 # Количество Pod'ов, которые нужно запустить.
  selector:
    matchLabels:
      app: nginx # Указывает, какие Pod'ы должен отслеживать Deployment.
  template: # Шаблон для создания Pod'ов.
    metadata:
      labels:
        app: nginx # Метка для Pod'ов, создаваемых этим Deployment.
    spec:
      containers: # Описание контейнеров в Pod'ах.
        - name: nginx # Имя контейнера.
          image: nginx:latest # Образ контейнера (последняя версия Nginx).
          ports:
            - containerPort: 80 # Порт, открытый в контейнере для доступа извне.
```

# Как происходит обновление deployment?

1. При изменении конфигурации (например, образа контейнера), Deployment создает новый ReplicaSet.
2. Старые Pod'ы заменяются новыми поэтапно, минимизируя простой (Rolling Update).
3. Deployment временно управляет двумя ReplicaSet (старым и новым) для обеспечения плавного перехода.

Демонстрация связи Deployment -  
> ReplicaSet -> Pod

# Взаимосвязи объектов с Pod

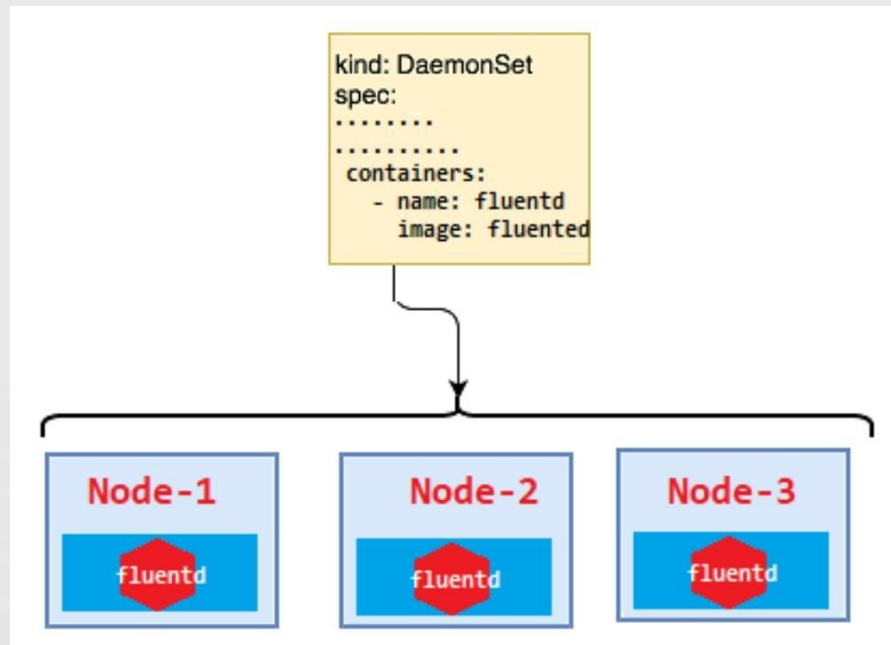


# DaemonSet

- DaemonSet гарантирует, что определённый Pod запущен на каждой доступной ноды в Kubernetes кластере.
- При добавлении новой ноды в кластер, DaemonSet автоматически запускает Pod на этой ноды.
- При удалении ноды, соответствующий Pod автоматически удаляется.

Типичное использование:

- Запуск демона сбора журналов на каждом узле
- Запуск демона мониторинга узла на каждом узле



<https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>



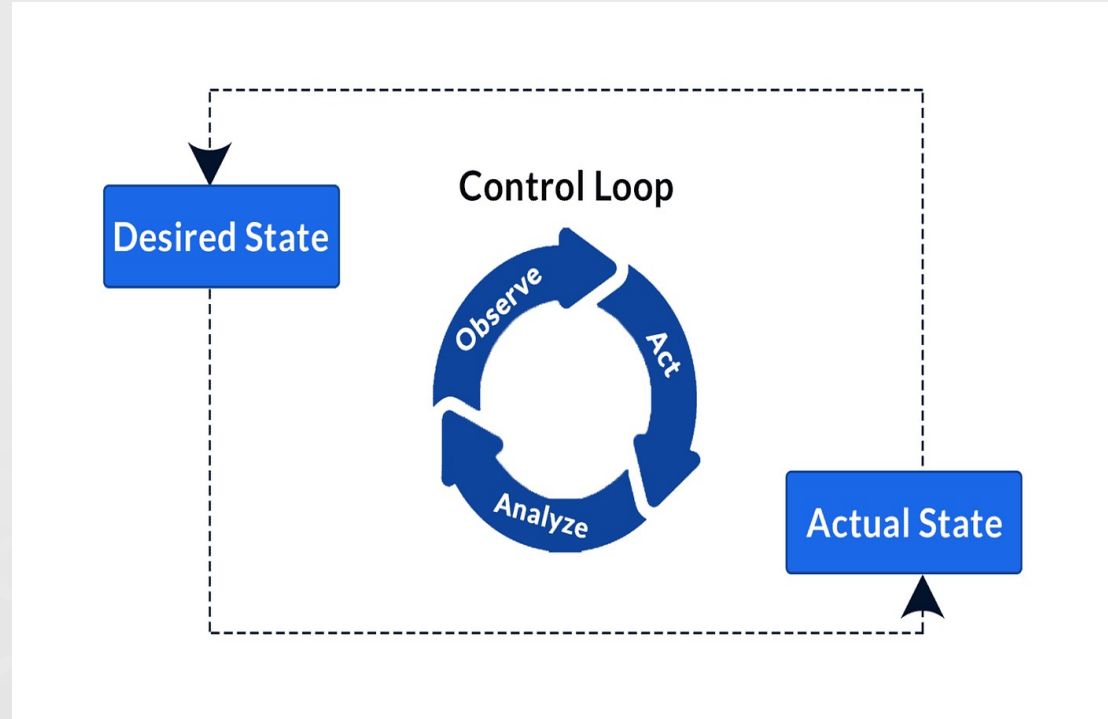
04

# Контроллеры

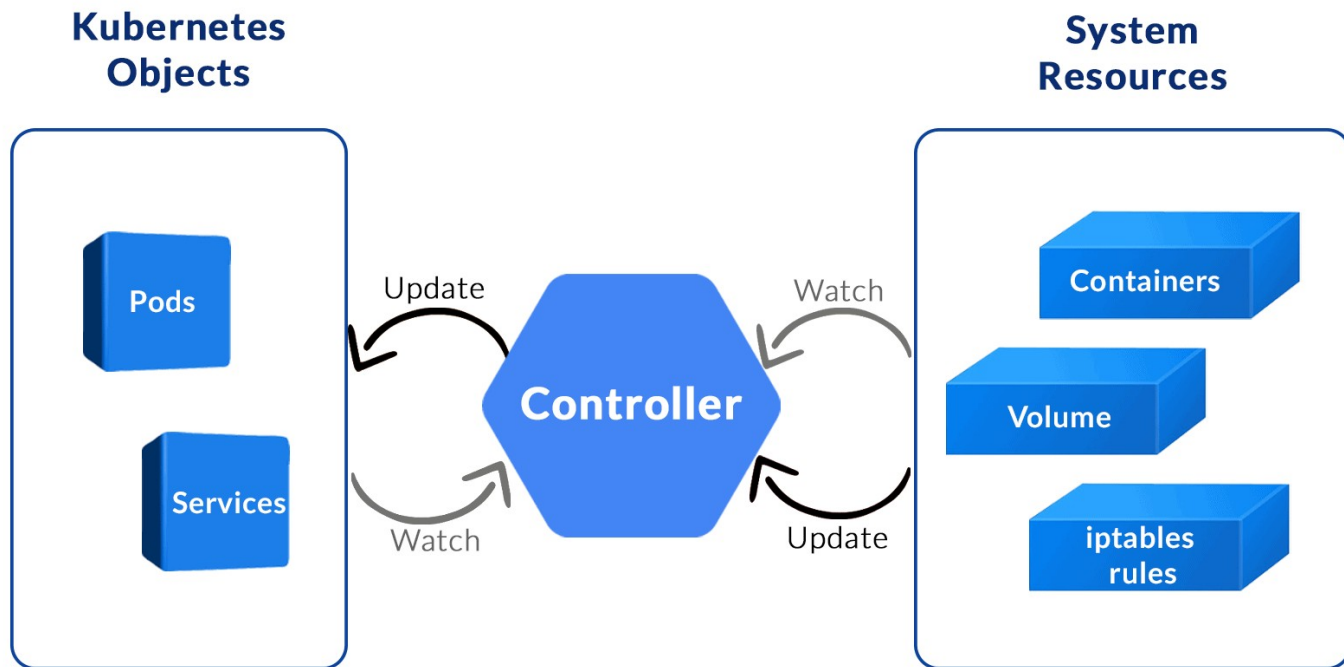
# Что такое контроллеры?

Контроллеры — ключевые компоненты в Kubernetes, которые обеспечивают соответствие желаемого состояния кластера текущему состоянию.

Контроллеры отслеживают объекты Kubernetes и предпринимают действия для устранения любых расхождений между желаемым и текущим состоянием.

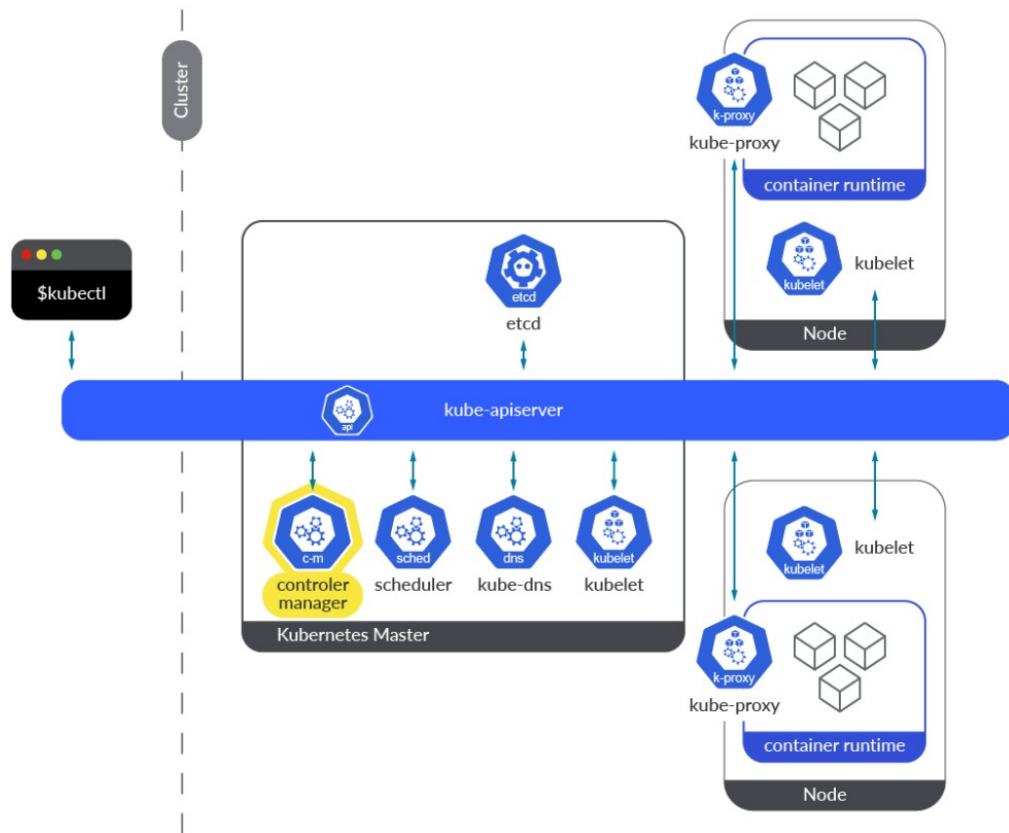


# Циклы управления





# Controllers в контексте кластера



# У ресурсов есть контроллеры



# Компоненты контроллеров

## Механизм Watch

- Механизм, который позволяет контроллерам "наблюдать" за изменениями в Kubernetes объектах в реальном времени
- API Server отправляет уведомления (события) контроллерам при изменении состояния отслеживаемых объектов.

## Shared Informer

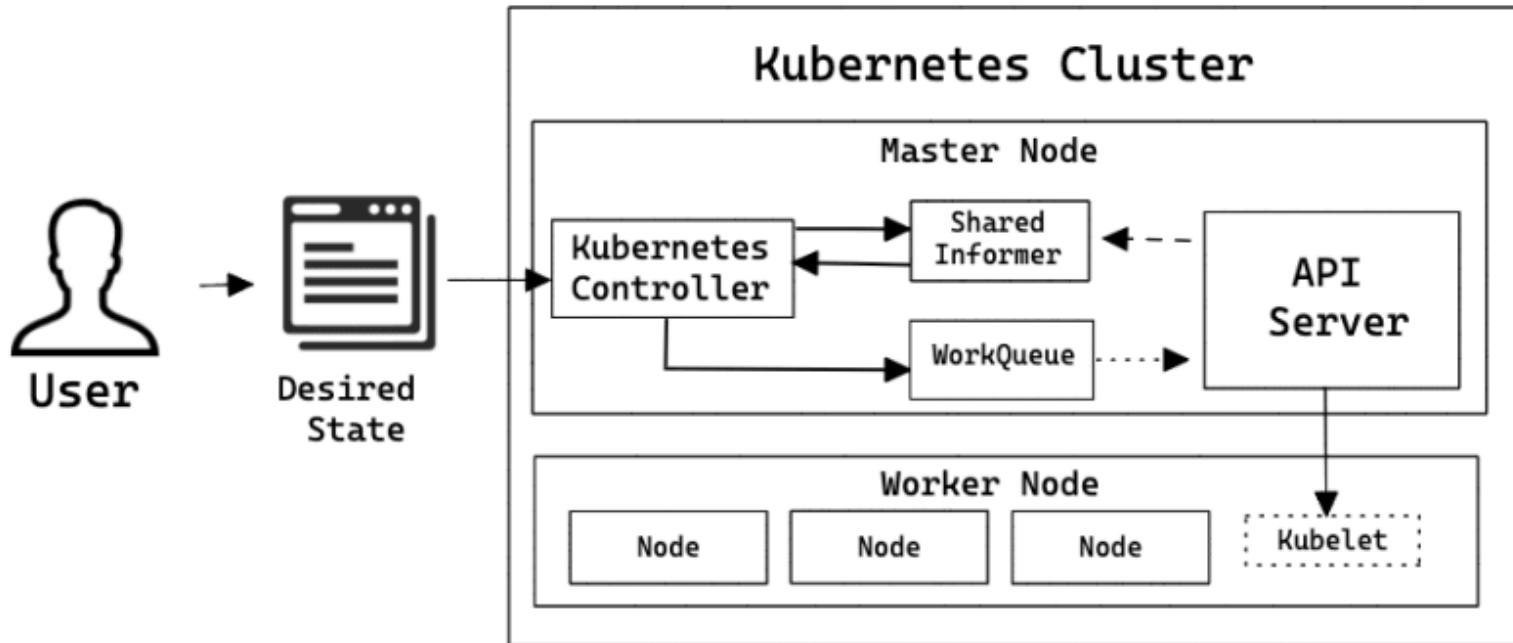
- Компонент, который кеширует данные о Kubernetes объектах и делится ими с несколькими контроллерами.
- Подписывается на изменения через механизм Watch, хранит данные в локальном кеше и уведомляет контроллеры об изменениях.

## Work Queue

- Очередь, в которую помещаются задачи на выполнение, когда обнаруживается рассогласование состояния.
- Контроллеры извлекают задачи из очереди и выполняют необходимые действия для достижения желаемого состояния.

# Как controllers работают?

## Working of a Kubernetes Controller



# Алгоритм работы контроллера полностью

## 1. Инициализация контроллера:

- a) Контроллер запускается как часть kube-controller-manager.
- b) Определяет тип объектов для отслеживания (например, Pod, Deployment, Job).

## 2. Настройка Shared Informer:

- a) Подписывается на изменения в состоянии объектов через API Server.
- b) Запрашивает начальное состояние и настраивает **watch** для мониторинга в реальном времени.

## 3. Обновление Work Queue:

- a) При изменении состояния объекта API Server уведомляет Shared Informer.
- b) Shared Informer добавляет задачи в Work Queue для обработки контроллером.

## 4. Обработка задач:

- a) Контроллер извлекает задачи из Work Queue, сравнивает текущее состояние объекта с желаемым.
- b) При необходимости инициирует изменения через API Server.

## 5. Принятие мер:

- a) Если текущее состояние отличается от желаемого, контроллер инициирует изменения.
- b) API Server обновляет etcd, а узлы (kubelet) приводят состояние в соответствие.

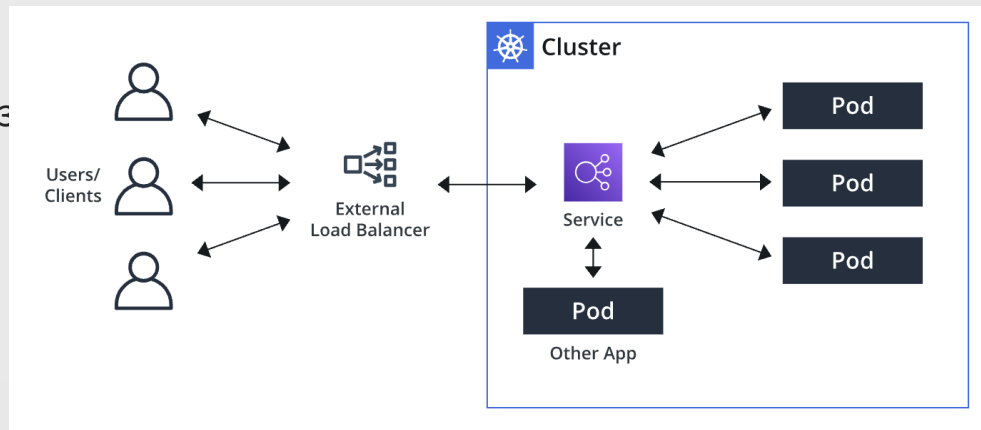
## 6. Постоянный цикл:

- a) Контроллер непрерывно отслеживает и корректирует состояние объектов в бесконечном цикле.

А на сегодня все!

# Service

Service — это объект, который обеспечивает доступ к набору Pod'ов через сеть (dns или ip).



<https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/>

# Типы service

## ClusterIP

Дефолтный тип сервиса, который открывает доступ к Pod'ам только внутри кластера через внутренний IP-адрес. Используется для внутренней коммуникации между сервисами.

## NodePort

Создает внешний load balancer (если поддерживается облачным провайдером) и направляет трафик на сервис внутри кластера. Подходит для продакшн-приложений, которым нужен доступ из интернета

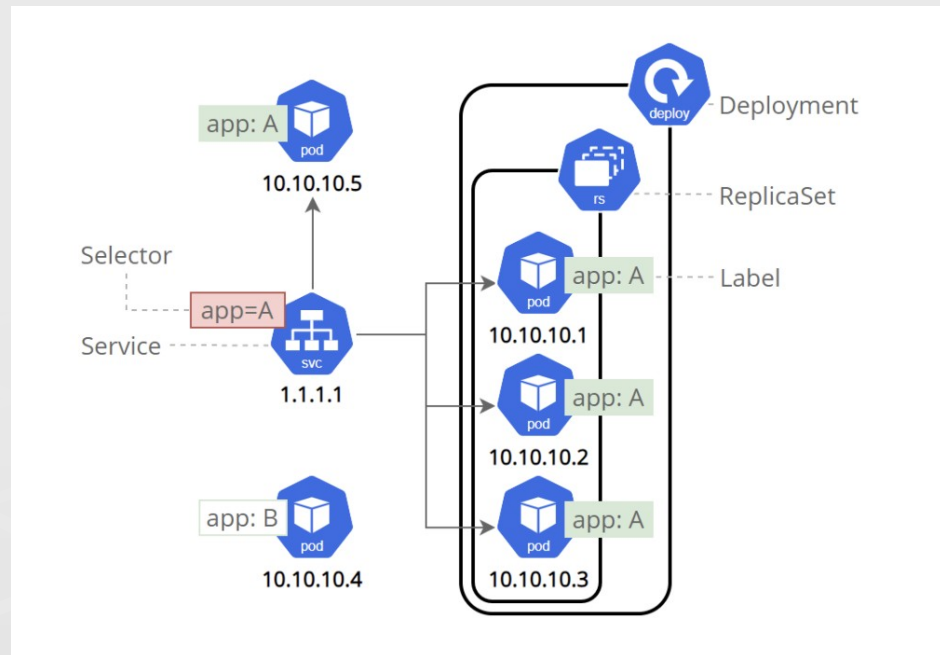
## NodePort

Экспонирует сервис на каждом узле кластера через определенный порт. Позволяет доступ к Pod'ам извне кластера через этот порт. Чаще используется в тестовых средах или для

<https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/>



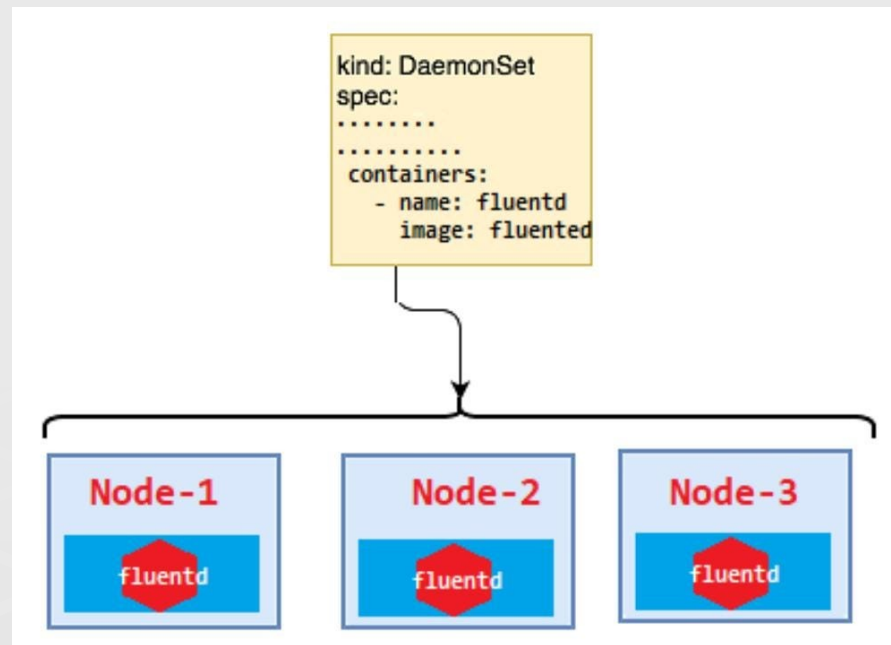
# Основные компоненты Service



- DaemonSet гарантирует, что определённый Pod запущен на каждой доступной ноде в Kubernetes кластере.
- При добавлении новой ноды в кластер, DaemonSet автоматически запускает Pod на этой ноде.
- При удалении ноды, соответствующий Pod автоматически удаляется.

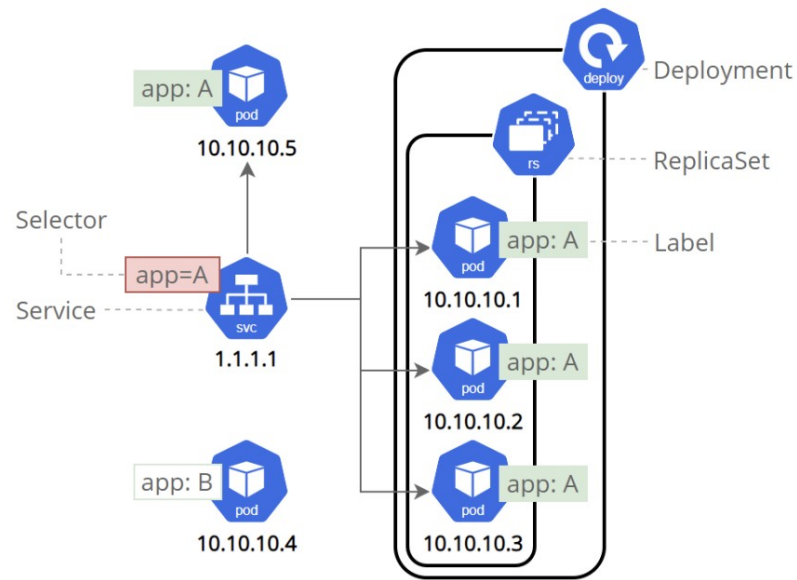
Типичное использование:

- Запуск демона сбора журналов на каждом узле
- Запуск демона мониторинга узла на каждом узле



<https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>

# Архитектура программного обеспечения



# Архитектура программного обеспечения

## Структура системы

Определяется архитектурными стилями и паттернами, такими как микросервисы, многослойная архитектура и т.д.

## Архитектурные решения

Определяют, как система должна быть построена. Они включают в себя правила и ограничения, направленные на достижение определенных архитектурных целей

## Архитектурные характеристики

Нефункциональные требования, например, масштабируемость, отказоустойчивость и т. д.

## Принципы проектирования

Руководящие указания, которые помогают при создании системы, являются подходами для решения типичных задач.

O'REILLY®



## Fundamentals of Software Architecture

An Engineering Approach

Mark Richards & Neal Ford