

Автоматизация разработки и эксплуатации программного обеспечения (осень 2023 года)



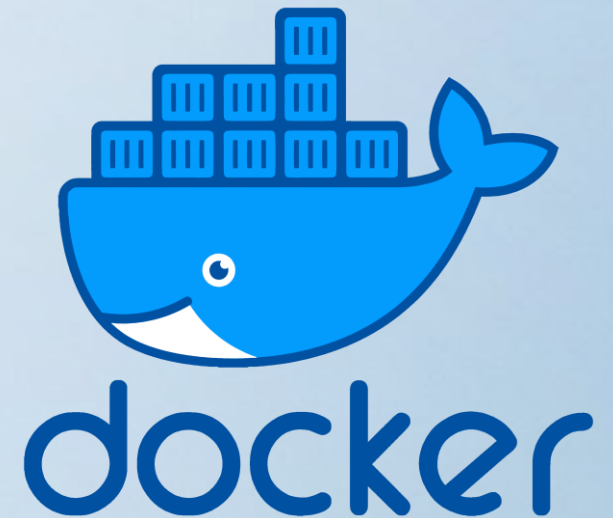
ИУ-5, бакалавриат, курс по выбору



ОСНОВЫ Kubernetes

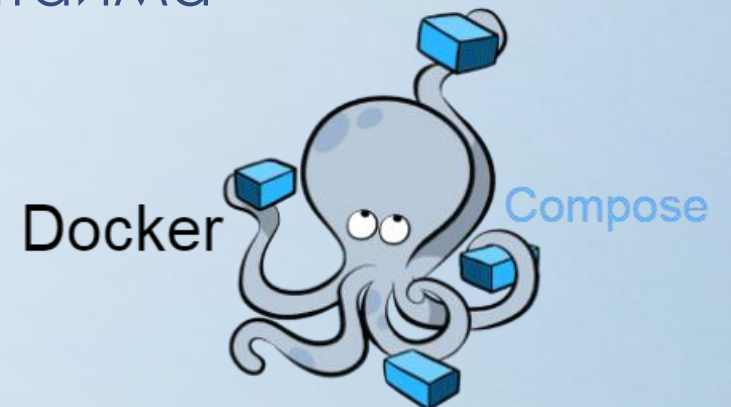
Docker

- Всем знаком
- Запускаем образы
- `docker run`
- `docker exec`
- `docker logs`



Docker compose

- Организует взаимосвязанный набор сервисов
- Организует сетевое взаимодействие и service discovering
- НО! Работает на одном сервере, значит упираемся в физические лимиты
- НО! Нет механизмов обновления без даунтайма



Распределенные приложения

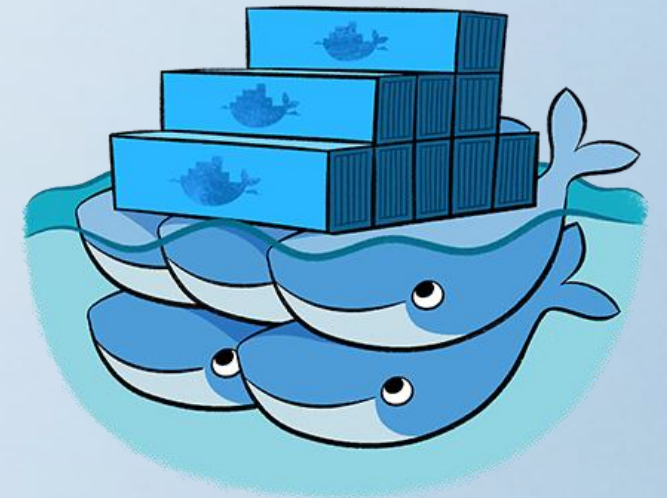
Если говорить про использование приложения под нагрузкой, мы хотим запускать копии приложения, чтобы:

- Защитить приложение от падения сервера, на котором запущен процесс
- Распределить нагрузку по нескольким серверам
- Распределить трафик на работающие копии приложения

Docker swarm

Решение для оркестрации docker контейнеров

- Нативное решение встроенное в докер
- Есть базовые операции
- Низкий порог вхождения



Недостатки Docker swarm

- Основа кластера docker swarm представляют управляющие master узлы и рабочие (worker)
- Ограничен на количеству узлов и контейнеров
- Отсутствует автомасштабирование
- Не удовлетворяет всем потребностям бизнеса

Hashicorp Nomad

Оркестратор от HashiCorp, который скорее представляет собой фреймворк для построения кластерных решений.

Существует community и enterprise версии.

- + Простой в установке, низкий порог входа
- + Поддерживает не только контейнеры
- Начальный функционал сильно ограничен
- Бесплатная версия ограничена
- Небольшое комьюнити



HashiCorp

Nomad

Apache Mesos

Менеджер для работы с различными типами приложениями и разными нагрузками.

- + Можно работать как с контейнерами и не контейнерами
- + Высокая производительность
- Сложность в поддержке и администрировании
- Все сложно с разработкой и поддержкой
- Высокий порог вхождения



Apache
MESOSTM

Kubernetes

Самодостаточный инструмент контейнерной оркестрации со множеством встроенных сервисов.

- Автоматизация развертывания
- Масштабирование (горизонтальное и вертикальное)
- Self-healing
- Основан на технологии контейнеризации (включая docker)
- Стандарт индустрии



Взаимодействие с кластером

Для взаимодействия с кластером используется специальная утилита `kubectl`

```
alias k="kubectl"  
complete -o default -o nospace -F __start_kubectl k
```

Устройство кластера

Нода (Node) – это физический сервер, который подключен к кластеру, на нем выполняется полезная нагрузка.

Остальные ноды – воркеры, на них запускаются приложения. Их уже может быть десятки или сотни

Master ноды

Мастер ноды – это (системные сервера, на которых работают «мозги» кластера), мастер ноды образуют control-plane. В зависимости от требований к отказоустойчивости мастер нод может быть 1-3-5-7 и тд штук на весь кластер.

Worker ноды

Остальные ноды – воркеры, на них запускаются приложения. Их уже может быть десятки или сотни

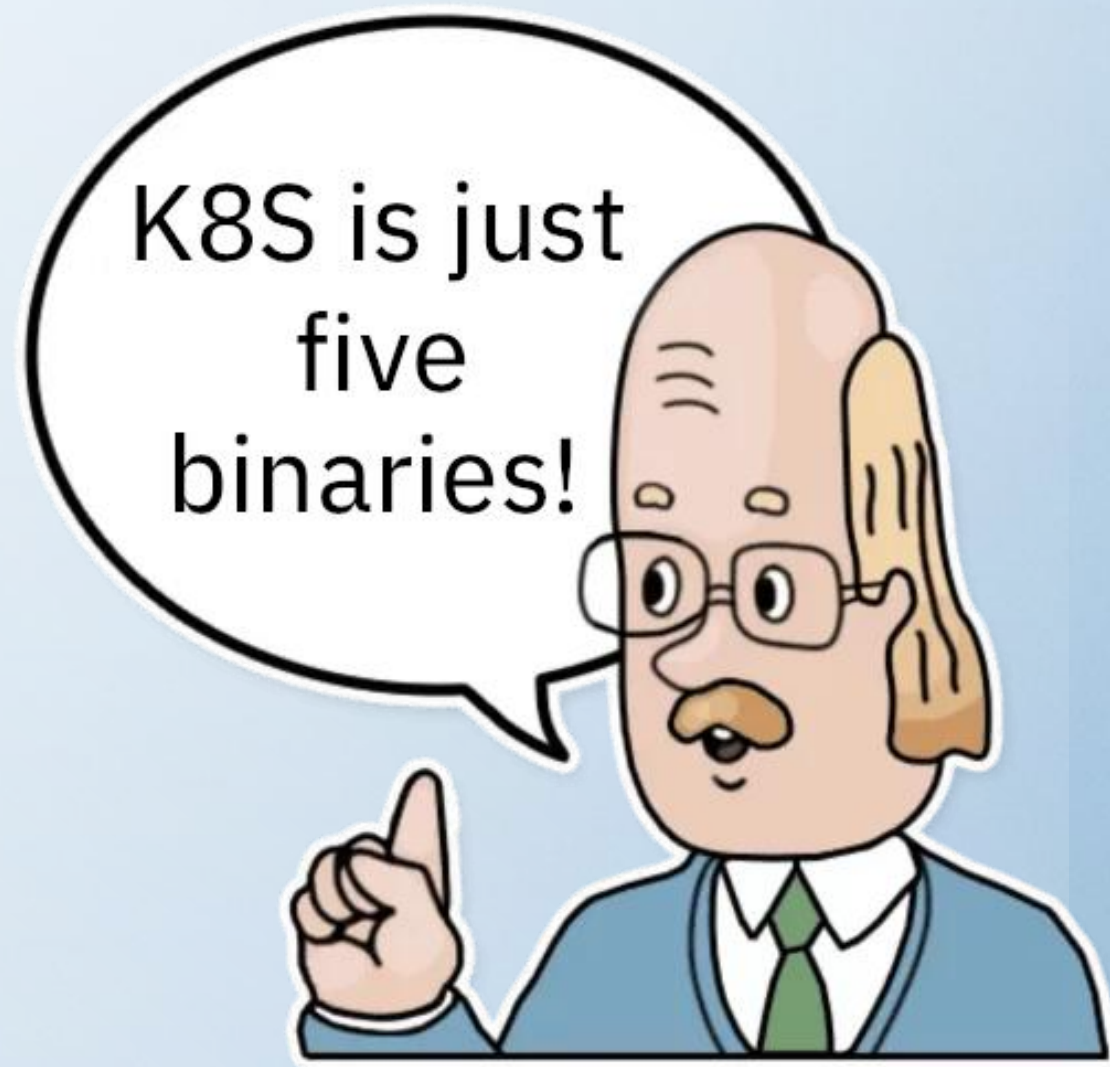
Компоненты

Control plane

- API server
- Controller manager
- Scheduler
- Cloud controller manager *
- Etcd

Worker node

- Kubelet
- Kube-proxy



API server

Сервер API — компонент Kubernetes панели управления, который представляет API Kubernetes. API-сервер — это клиентская часть панели управления Kubernetes.

etcd

Распределённое и высоконадёжное хранилище данных в формате "ключ-значение", которое используется как основное хранилище всех данных кластера в Kubernetes.

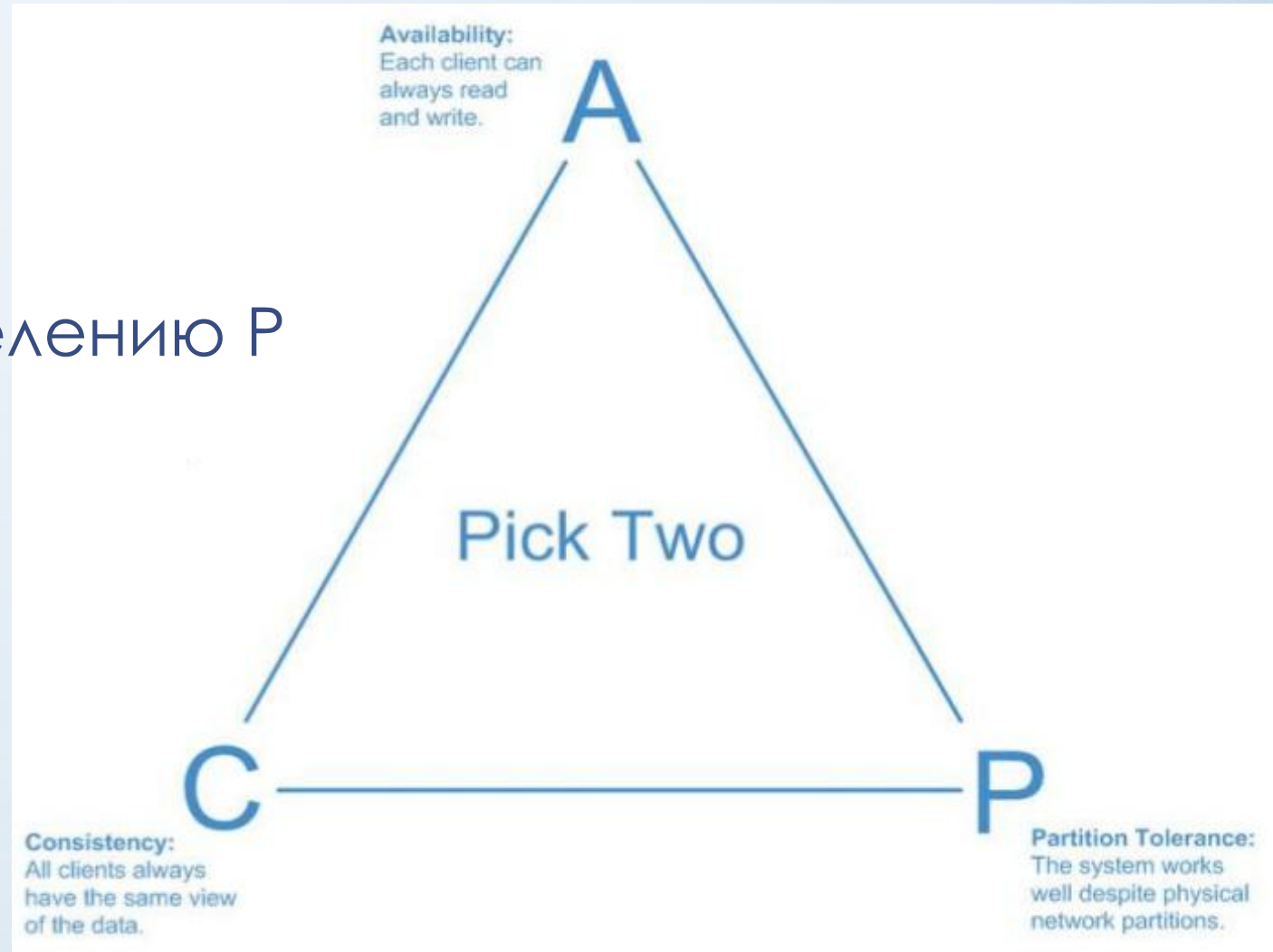


САР теорема

Согласованность C

Доступность A

Устойчивость к разделению P



Алгоритм консенсуса RAFT

<https://thesecretlivesofdata.com/raft/>

Scheduler

Kube-scheduler выбирает ноду, на которой возможно запустить пользовательское приложение.

При планировании развёртывания подов на узлах учитываются множество факторов, включая требования к ресурсам, ограничения, связанные с аппаратными/программными политиками, принадлежности (affinity) и непринадлежности (anti-affinity) узлов/подов, местонахождения данных, предельных сроков.

Controller

kube-controller-manager запускает процессы контроллера.

Эти контроллеры включают:

- Контроллер узла (Node Controller): уведомляет и реагирует на сбои узла.
- Контроллер репликации (Replication Controller): поддерживает правильное количество подов для каждого объекта контроллера репликации в системе.
- Контроллер конечных точек (Endpoints Controller): заполняет объект конечных точек (Endpoints), то есть связывает сервисы (Services) и поды (Pods).
- Контроллеры учетных записей и токенов (Account & Token Controllers): создают стандартные учетные записи и токены доступа API для новых пространств имен.

Kubelet

Агент, работающий на каждом узле в кластере. Он следит за тем, чтобы контейнеры были запущены в поде.

Утилита kubelet принимает набор PodSpecs, и гарантирует работоспособность и исправность определённых в них контейнеров. Агент kubelet не отвечает за контейнеры, не созданные Kubernetes.

Kube-proxy

kube-proxy — сетевой прокси, работающий на каждом узле в кластере, и реализующий часть концепции сервис.

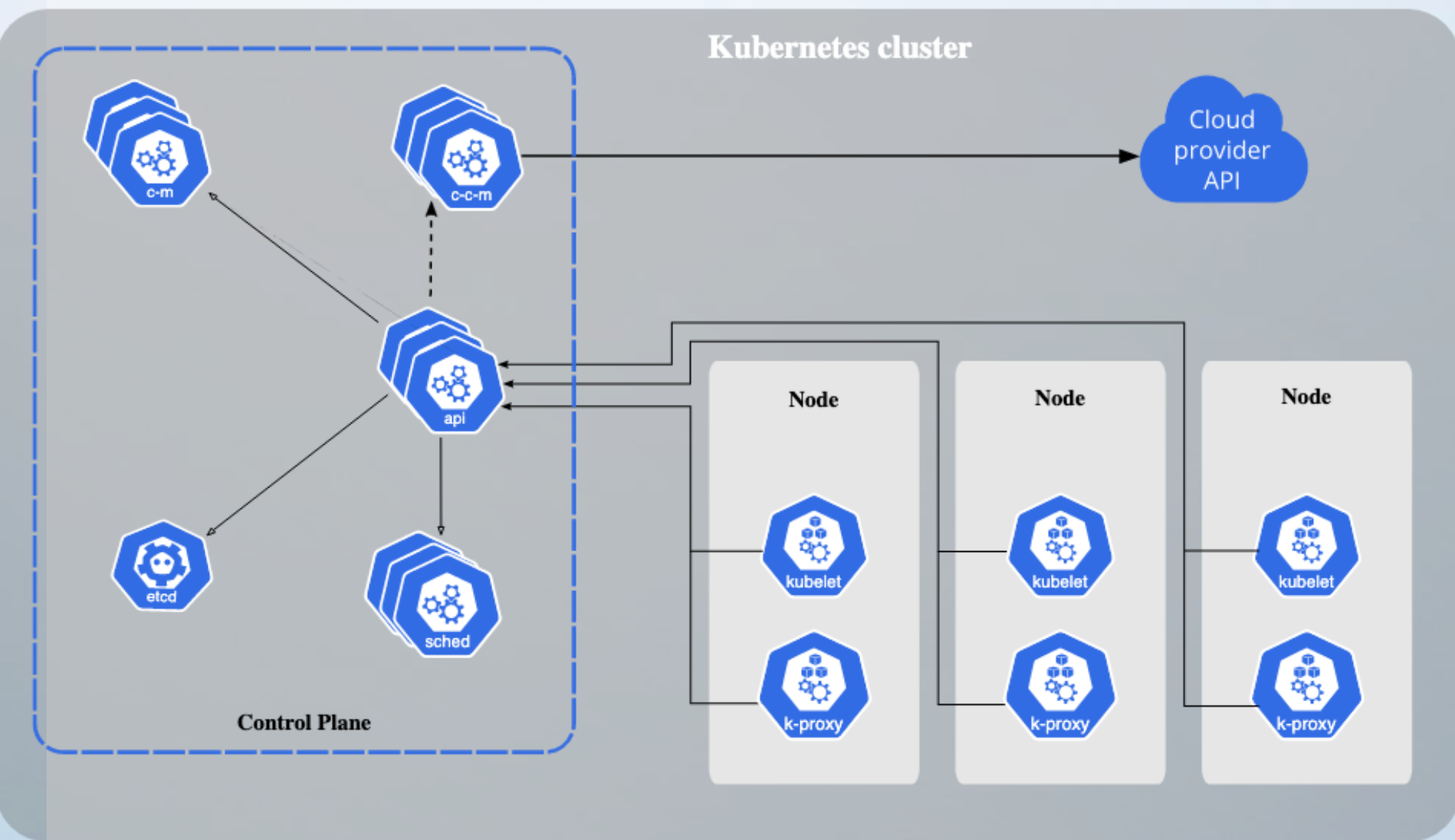
kube-proxy конфигурирует правила сети на узлах. При помощи них разрешаются сетевые подключения к вашим подам изнутри и снаружи кластера.

kube-proxy использует уровень фильтрации пакетов в операционной системы, если он доступен. В противном случае, kube-proxy сам обрабатывает передачу сетевого трафика.

Среда выполнения контейнера

Среда выполнения контейнера — это программа, предназначенная для выполнения контейнеров.

Kubernetes поддерживает несколько сред для запуска контейнеров: Docker, containerd, CRI-O, и любая реализация Kubernetes CRI (Container Runtime Interface).



Namespace

```
k get namespaces  
k get namespace  
k get ns
```

Namespace – логическое изолирование ресурсов (но можно и на сетевом уровне) друг от друга.

Можно выдавать права пользователям на отдельные ns

Можно настраивать лимиты потребления ресурсов на отдельные ns

Переключение по ns происходит либо с помощью опции `-n`, либо через смену контекста. Либо с помощью утилиты `ke`.

Забегаая вперед, ns используются и для обнаружения сервисов, так внутри одного ns сервисы могут найти друг друга по короткому имени `service`, а сервис из другого ns через `service.ns`

```
k -n default  
k config set-context --current --namespace=default
```

Задачи kubernetes

Оркестрация:

- Физический запуск приложения
- Следить за тем, чтобы приложение работало
- Обновление приложения
- Масштабирования

Сетевой доступ:

- Приложение должно быть доступно другим пользователям

Pod

```
k get pods
```

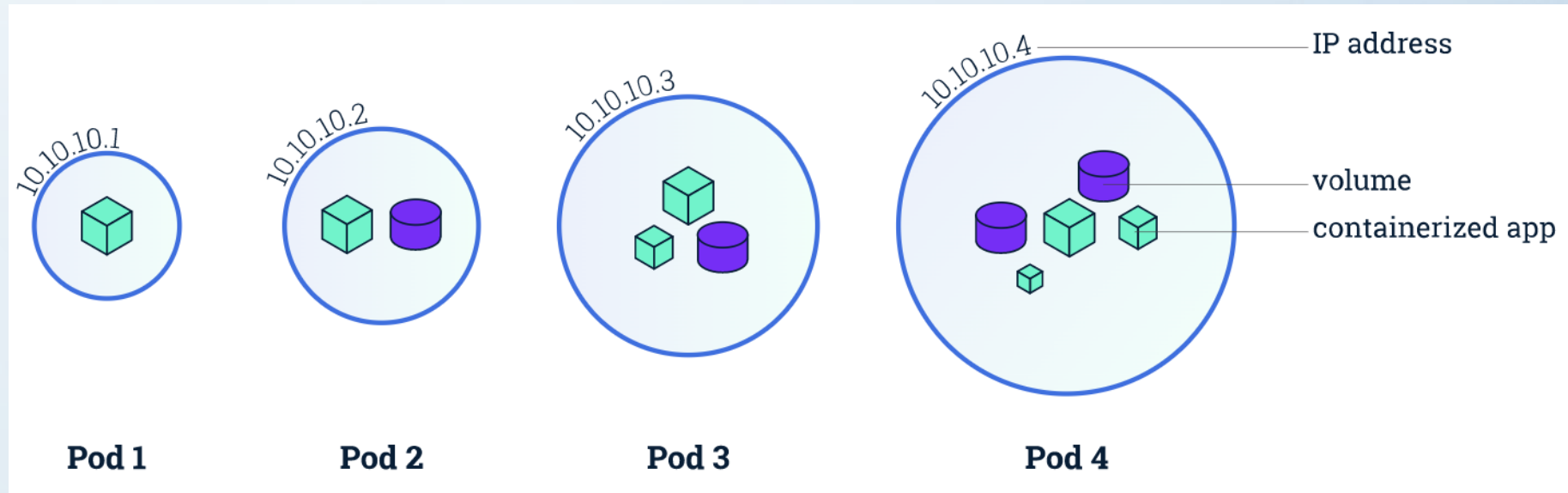
Pod – минимальная единица работы в кубере.

- Каждому поду присваивается IP адрес, который доступен на время жизни пода.
- Pod не изменяемый объект, создав под нельзя ничего в нем поменять, включая образ, версию, команду запуска.
- Если под завершился, его нельзя перезапустить (именно сам завершившийся под)
- Сам под может состоять из одного или несколько контейнеров

Volume (Том)

Общий ресурс хранения для совместного использования из контейнеров, развернутых в подах

Может быть в виде сетевого диска, физического диска или отдельных файлов конфигурации

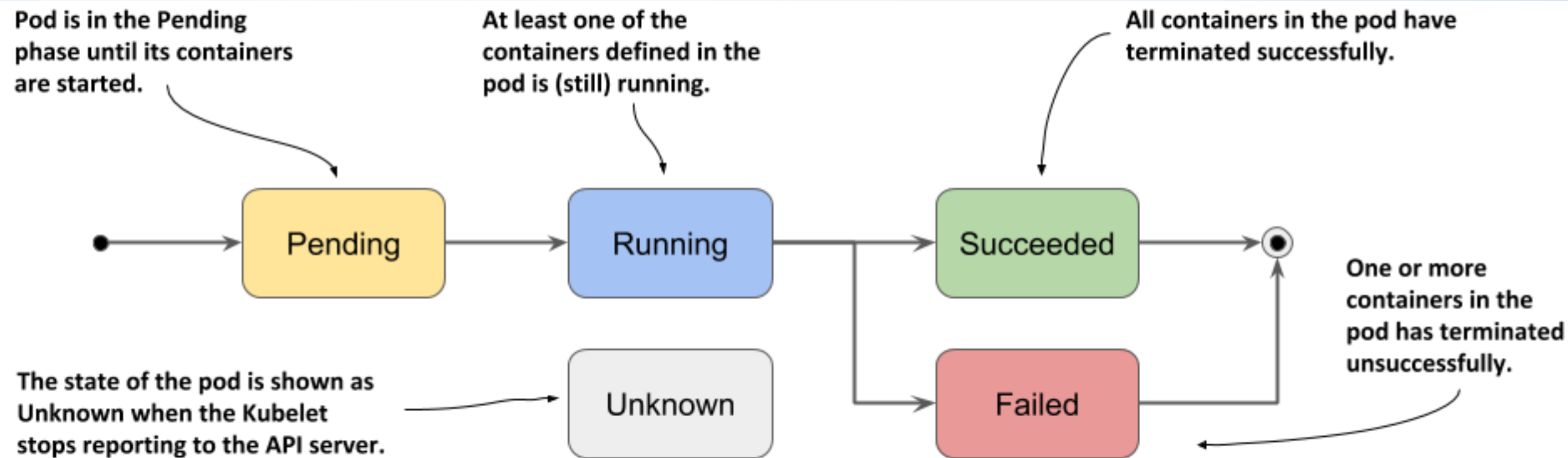


Pod

```
k get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
autotest-86c87cb674-p6wmr	1/1	Running	0	6d2h	10.100.197.129	kubenv-qa-workers-2
creativeserver-649f99b45c-dfgvn	1/1	Running	0	6d2h	10.100.134.192	kubenv-qa-workers-0
mediaserver-79d6646f46-7p7nz	1/1	Running	0	6d2h	10.100.115.0	kubenv-qa-workers-1
nginx-proxy-db7d6fccf-wnqfg	1/1	Running	6	6d2h	10.100.197.130	kubenv-qa-workers-2
phraseapid-5544c64f5b-pvlmf	1/1	Running	0	3d5h	10.100.134.202	kubenv-qa-workers-0
rb3-55866b95bc-q4jlm	1/1	Running	0	2d3h	10.100.134.203	kubenv-qa-workers-0
rb3-frontend-6cb7c88b78-wpfl4	1/1	Running	0	2d3h	10.100.115.9	kubenv-qa-workers-1
rb3-lua-8bd6bdb7d-7qlsq	1/1	Running	0	6d2h	10.100.115.1	kubenv-qa-workers-1
rbstoraged-56d6f96b56-w8db5	1/1	Running	0	6d2h	10.100.115.3	kubenv-qa-workers-1
rbstoreinfo-5dbd9bbd4f-xzwmg	1/1	Running	2	6d2h	10.100.134.194	kubenv-qa-workers-0
rbui-86cc79c7bd-tkj25	0/1	CrashLoopBackOff	1715	6d2h	10.100.134.195	kubenv-qa-workers-0
reklama-mysql-74ccc848c9-74fjs	1/1	Running	0	6d2h	10.100.197.131	kubenv-qa-workers-2
statapid-85c7b46589-ldhvc	1/1	Running	0	3d5h	10.100.134.200	kubenv-qa-workers-0
target-clickhouse-6d87fd98b5-5249m	1/1	Running	0	6d2h	10.100.115.4	kubenv-qa-workers-1
target-etcd-84fbffc7fc-mg24w	1/1	Running	0	6d2h	10.100.115.5	kubenv-qa-workers-1
target-mysql-5cc8cc4cf5-p2t8s	1/1	Running	0	2d3h	10.100.197.136	kubenv-qa-workers-2
target-tarantool20-5d66cb59f7-qlvdx	1/1	Running	0	6d2h	10.100.134.197	kubenv-qa-workers-0
target-zookeeper-85b9686d9b-fbhbm	1/1	Running	0	6d2h	10.100.197.133	kubenv-qa-workers-2
video-download-6c85ccd968-8htc	1/1	Running	0	6d2h	10.100.115.6	kubenv-qa-workers-1

Жизненный цикл пода



Создание пода

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Манифест пода может включать:

- Несколько контейнеров
- Правила валидации работы пода liveness и readiness пробы
- Ограничения по ресурсам limits/requests
- Подключения различных томов (volumes)
- Настройки для фильтрации нод, на которых может быть запущен под

Init container

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app.kubernetes.io/name: MyApp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.28
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox:1.28
    command: ['sh', '-c', "until nslookup myservice.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace) {
  - name: init-mydb
    image: busybox:1.28
    command: ['sh', '-c', "until nslookup mydb.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace) {
```

Replicaset

```
k -n master get replicaset
```

replicaset – группа подов одной версии. Репликасет порождает поды в заданном количестве по заданному шаблону и следит, чтобы их было заданное количество.

На самом деле почти никогда явно не используется и почти всегда управляется через deployment

NAME	DESIRED	CURRENT	READY	AGE
autotest-86c87cb674	1	1	1	6d2h

Deployment

```
k get deploy
```

Deployment – высокоуровневая абстракция, которая позволяет создавать, обновлять и масштабировать приложение.

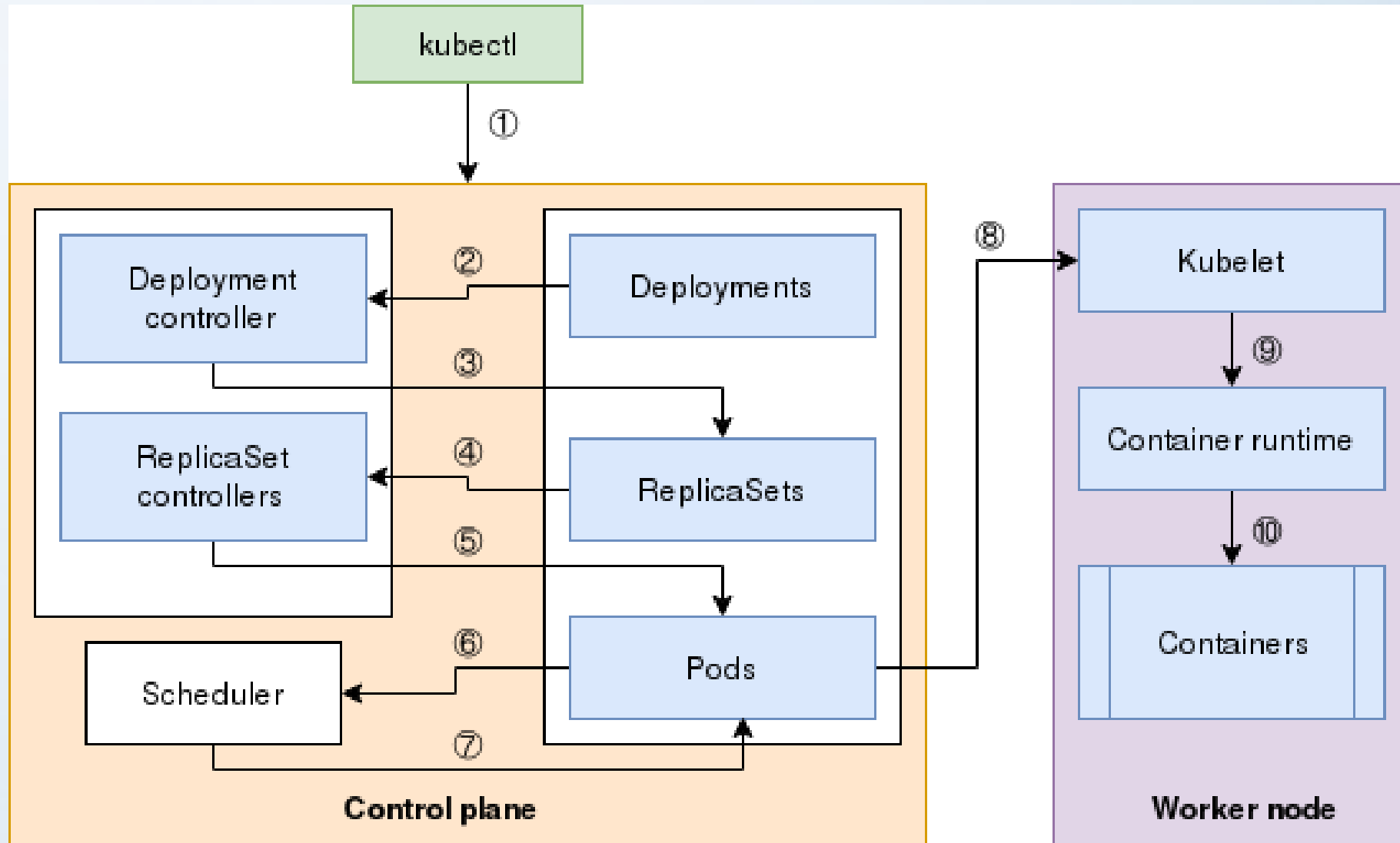
Создание deployment

Помимо метаданных о самом объекте в deployment указывается:

- Число реплик
- Стратегия обновления подов
- Selector, который позволяет определить, какие поды должны управляться деплойментом
- Шаблон пода

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```


Процесс создания подов deployment

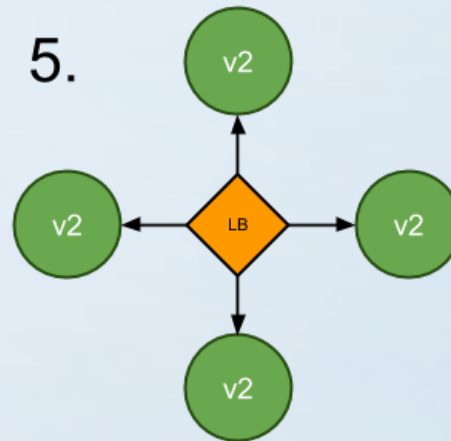
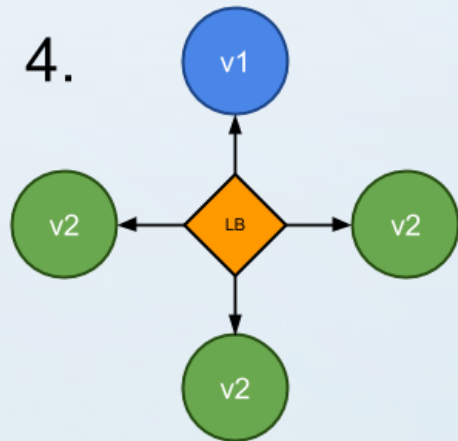
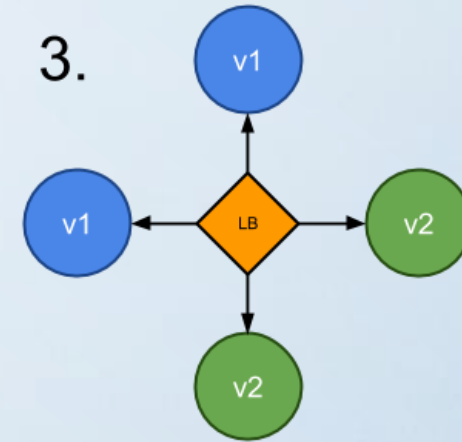
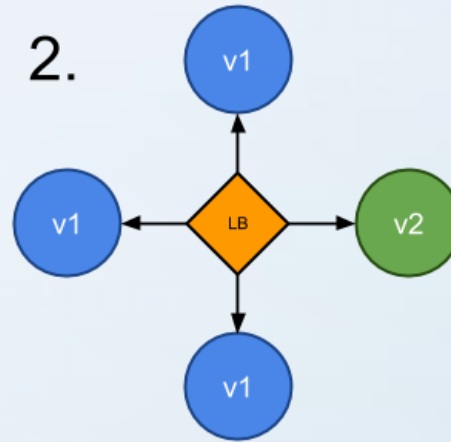
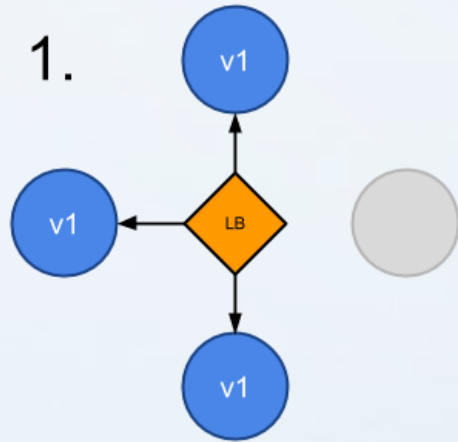


Стратегия обновления

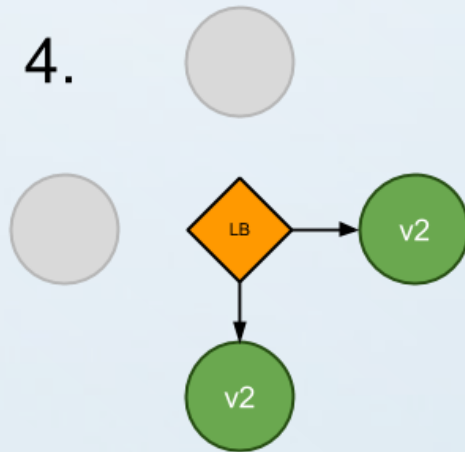
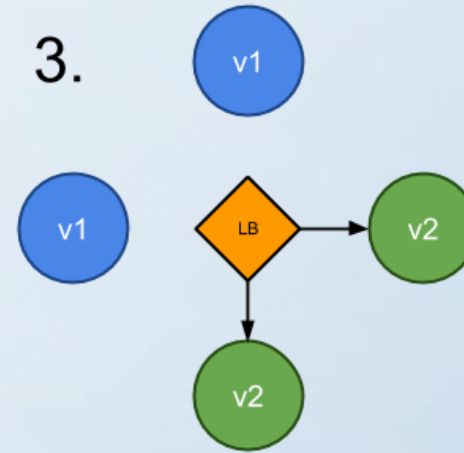
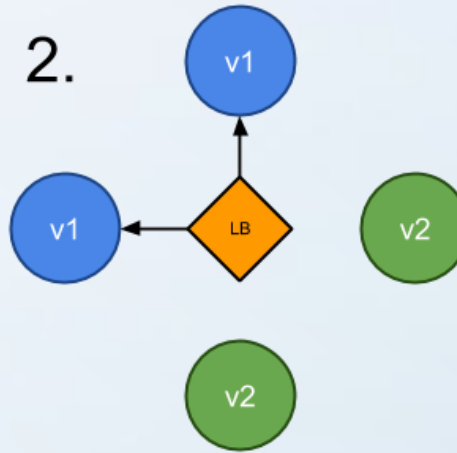
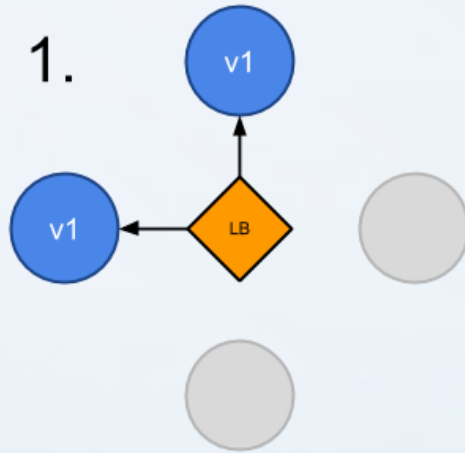
Определяет в каком порядке обновлять поды в deployment

- Recreate
- Rolling update
- Blue/green
- Canary
- A/B testing

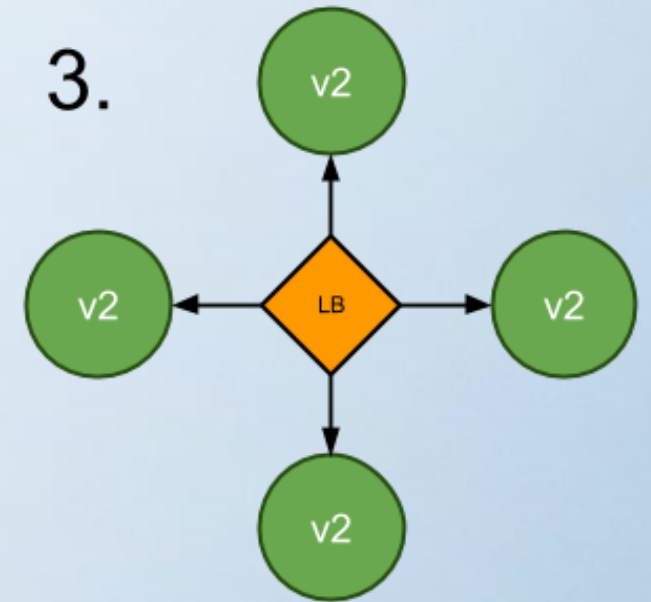
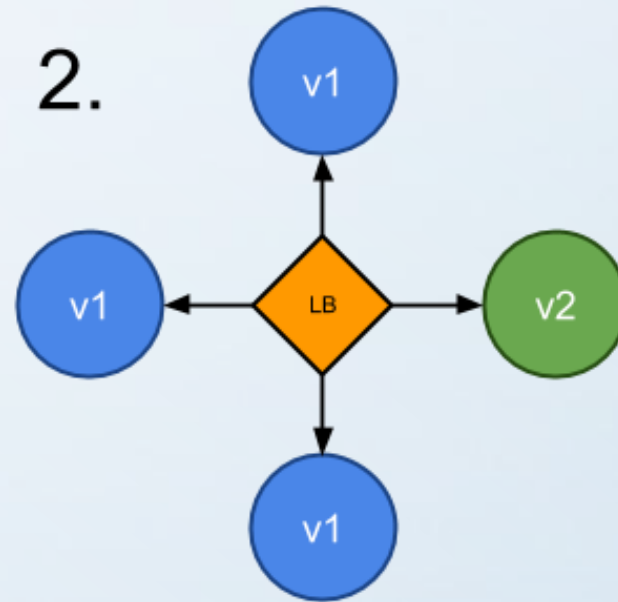
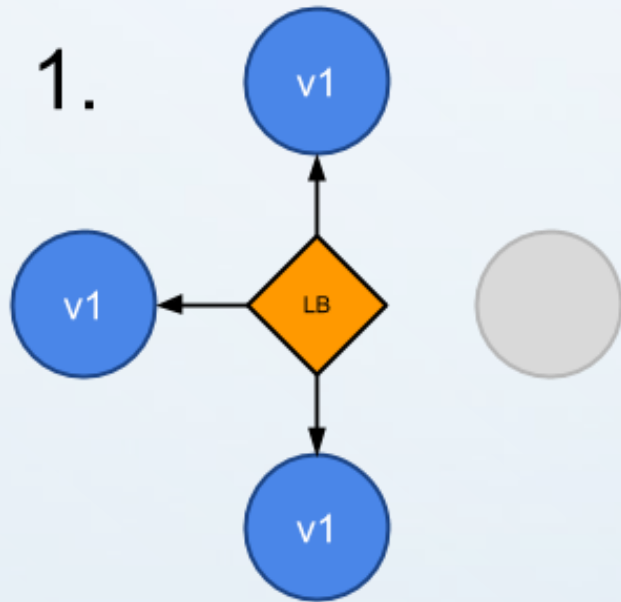
Rolling update



Blue/green

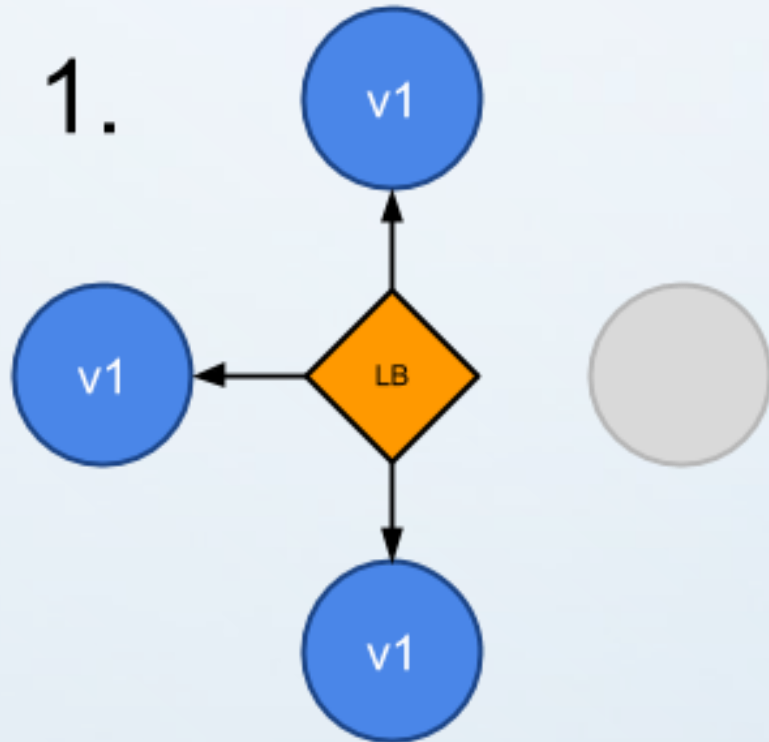


Canary

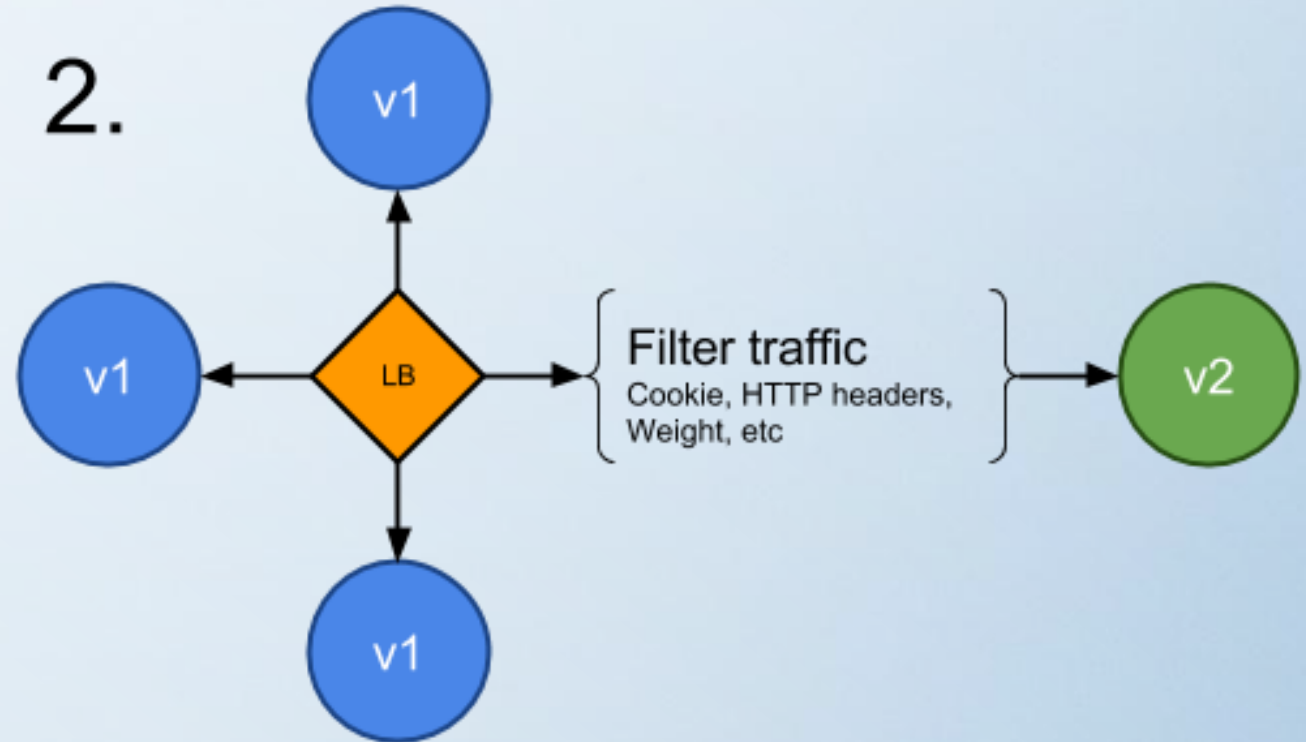


A/B testing

1.



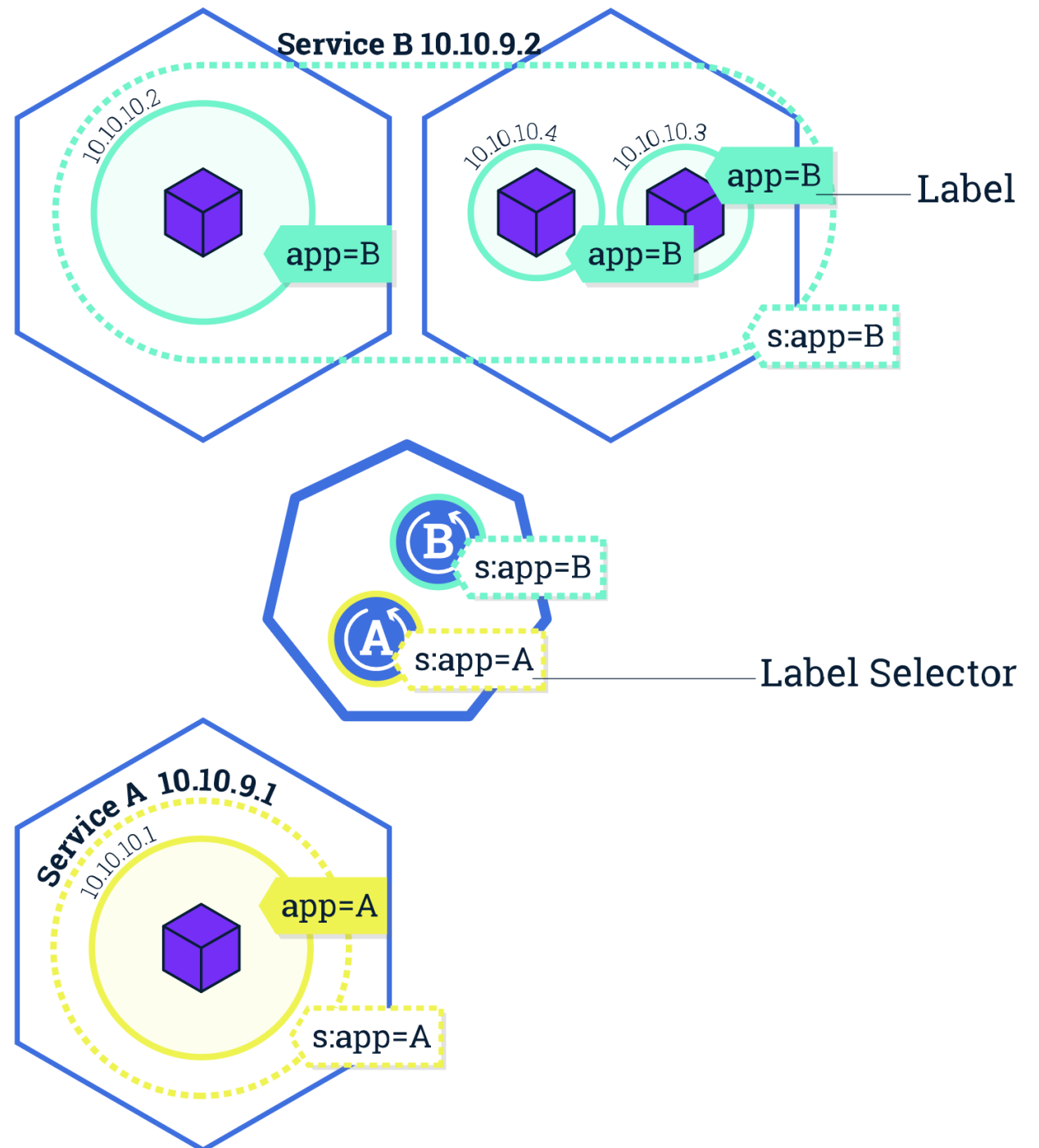
2.



Service

```
k get svc
```

Хоть каждый под имеет свой ip, но каждое пересоздание пода приводит к новому ip, а еще существует масштабирование, поэтому для организации сетевого взаимодействия существует особая сущность Service



DNS (service discovering)

Для каждого сервиса создается локальное для кластера dns имя

`my-svc.my-namespace.svc.cluster.local`

В целом, работать будут и укороченные имена *my-svc* в рамках одного ns или *my-svc.my-namespace* в рамках одного кластера.

Но лучше сразу писать полное имя, так понятнее и надежнее

Определение сервиса

```
kind: Service
apiVersion: v1
```

metadata:

```
  name: hostname-service
```

Make the service available
to network requests from
external clients

spec:

```
  type: NodePort
```

selector:

```
  app: echo-hostname
```

Forward requests to pods
with label of this value

ports:

```
  - nodePort: 30163
```

```
    port: 8080
```

```
    targetPort: 80
```

nodePort

access service via this external port number

port

port number exposed internally in cluster

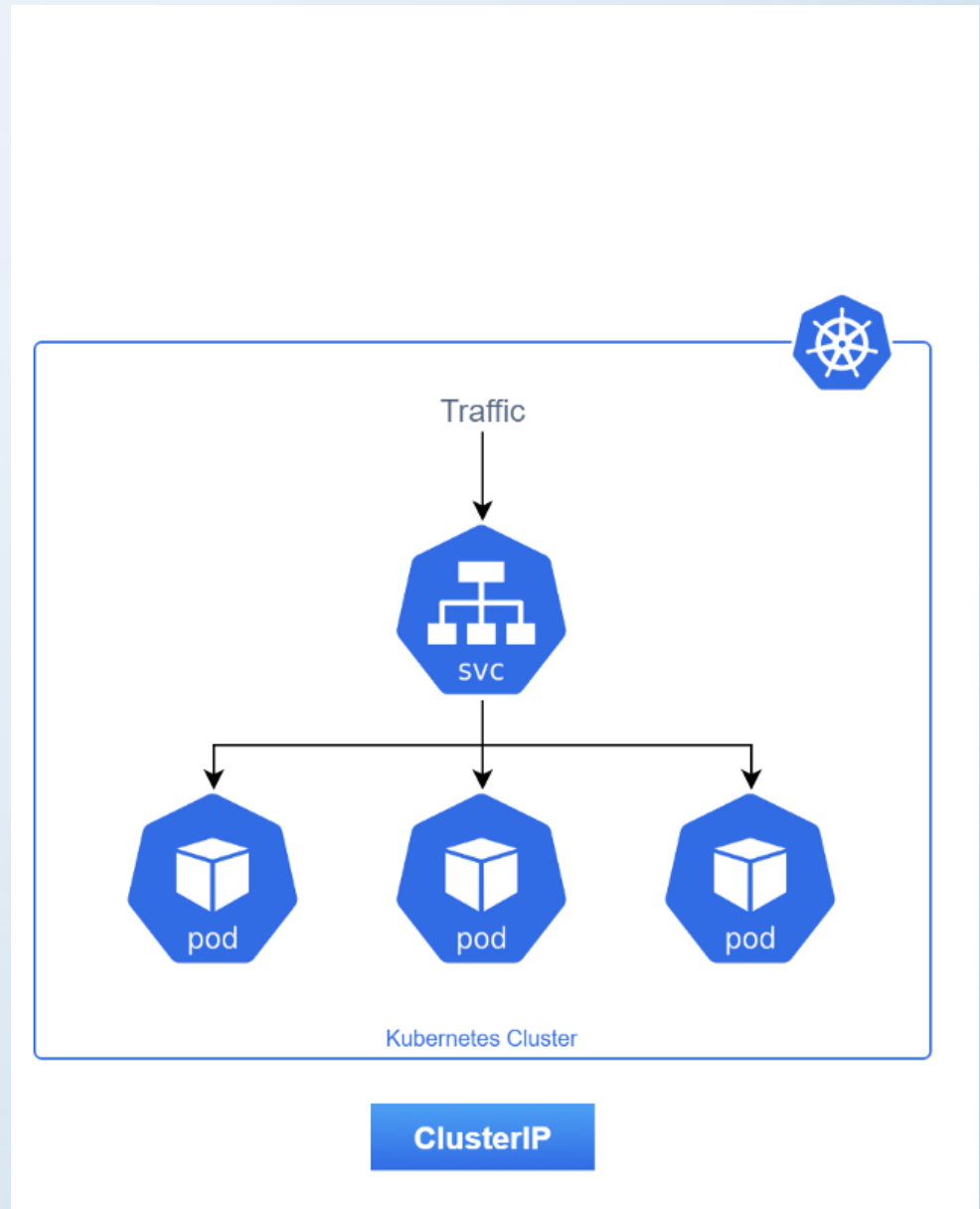
targetPort

port that containers are listening on

Cluster IP

Основной вид сервисов для взаимодействия приложений в кубере между собой.

На сервис выделяется постоянный ip, который не меняется

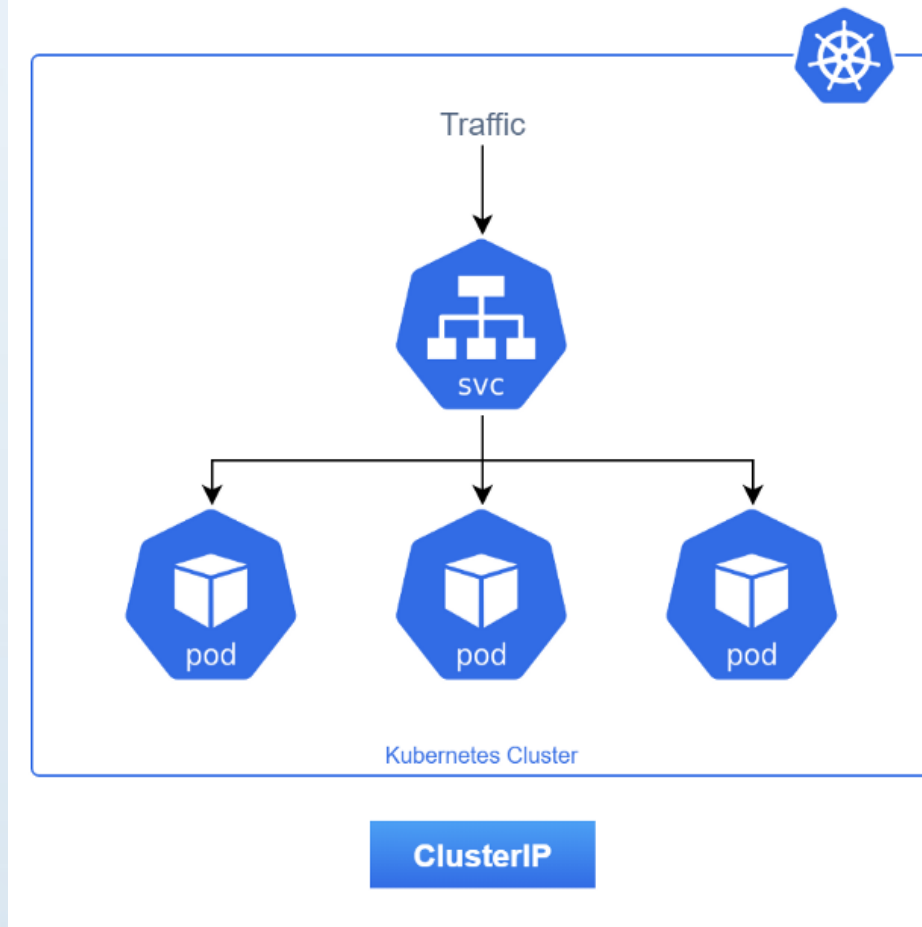


Cluster IP физически

```
Chain KUBE-SVC-I7EAKVFJLYM7WH25 (1 references)
target      prot opt source                destination
KUBE-SEP-LXP5RGX0X6SCIC6C  all  --  anywhere              anywhere
    statistic mode random probability 0.250000000000
KUBE-SEP-XRJTEP3YTXUYFBMK  all  --  anywhere              anywhere
    statistic mode random probability 0.33332999982
KUBE-SEP-0MZR4HWUSCJLN33U  all  --  anywhere              anywhere
    statistic mode random probability 0.500000000000
KUBE-SEP-EELL7LVIDZU4CPY6  all  --  anywhere              anywhere
```

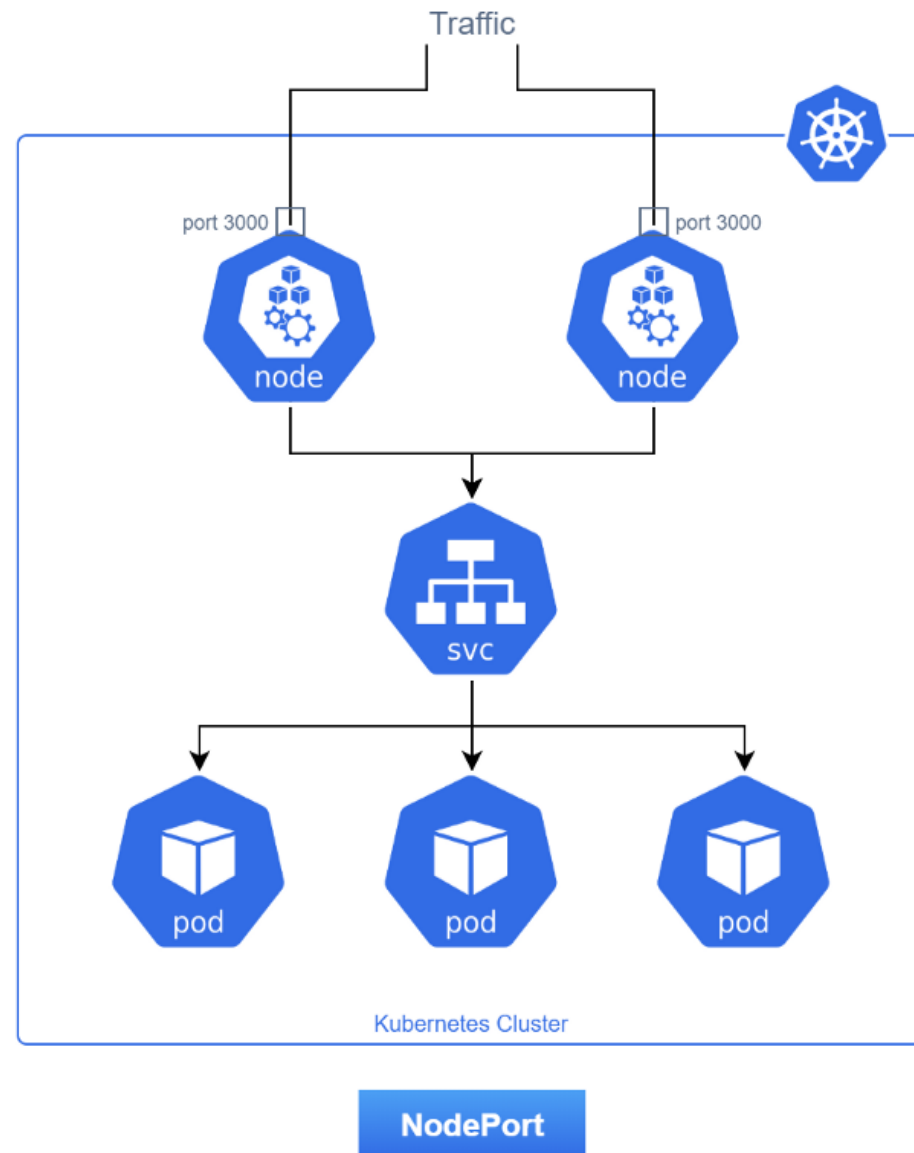
Cluster IP headless

Если при создании clusterip сервиса указать clusterip: none, тогда отдельный ip не выделится, а актуальный список подов сразу будет доступен в dns записи сервиса.



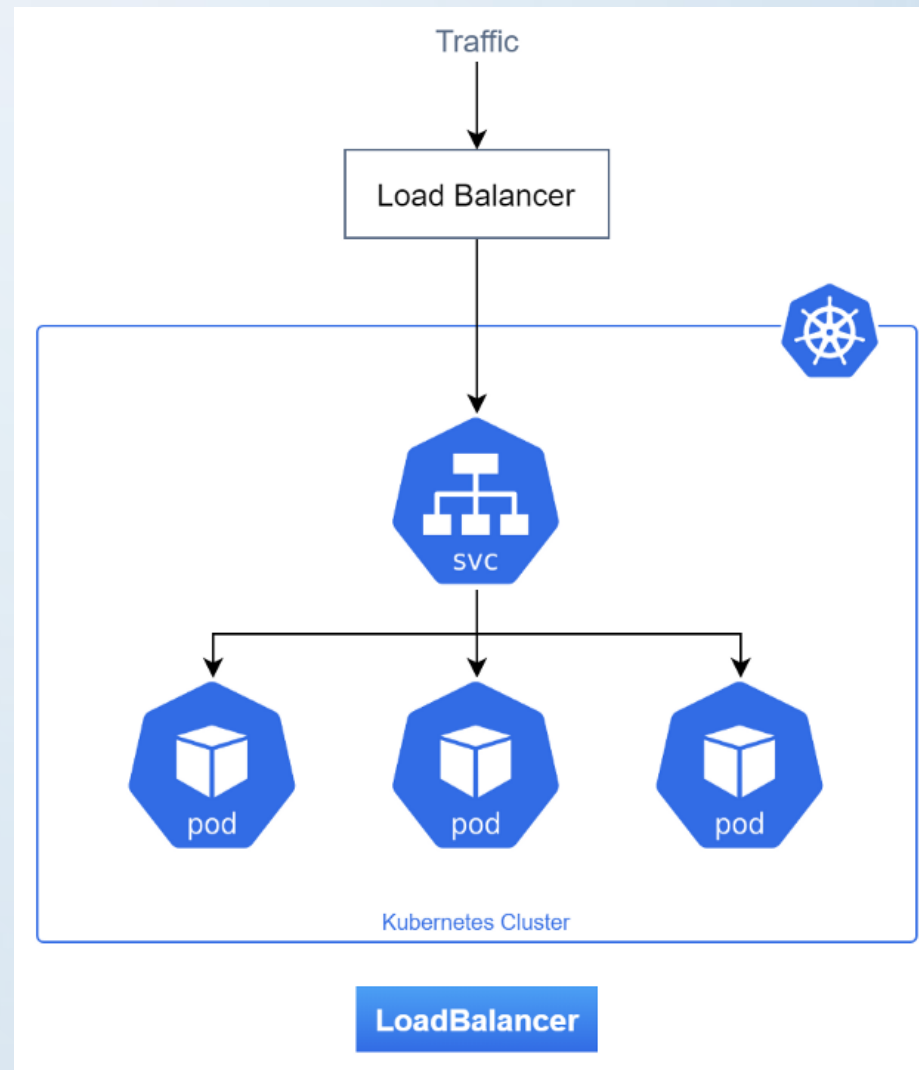
NodePort

Примитивный сервис для того, чтобы сделать приложение доступным из интернета. Работает аналогично bind порта у докера, в итоге порт контейнера связывается в портом у сервера, на которой под запущен.



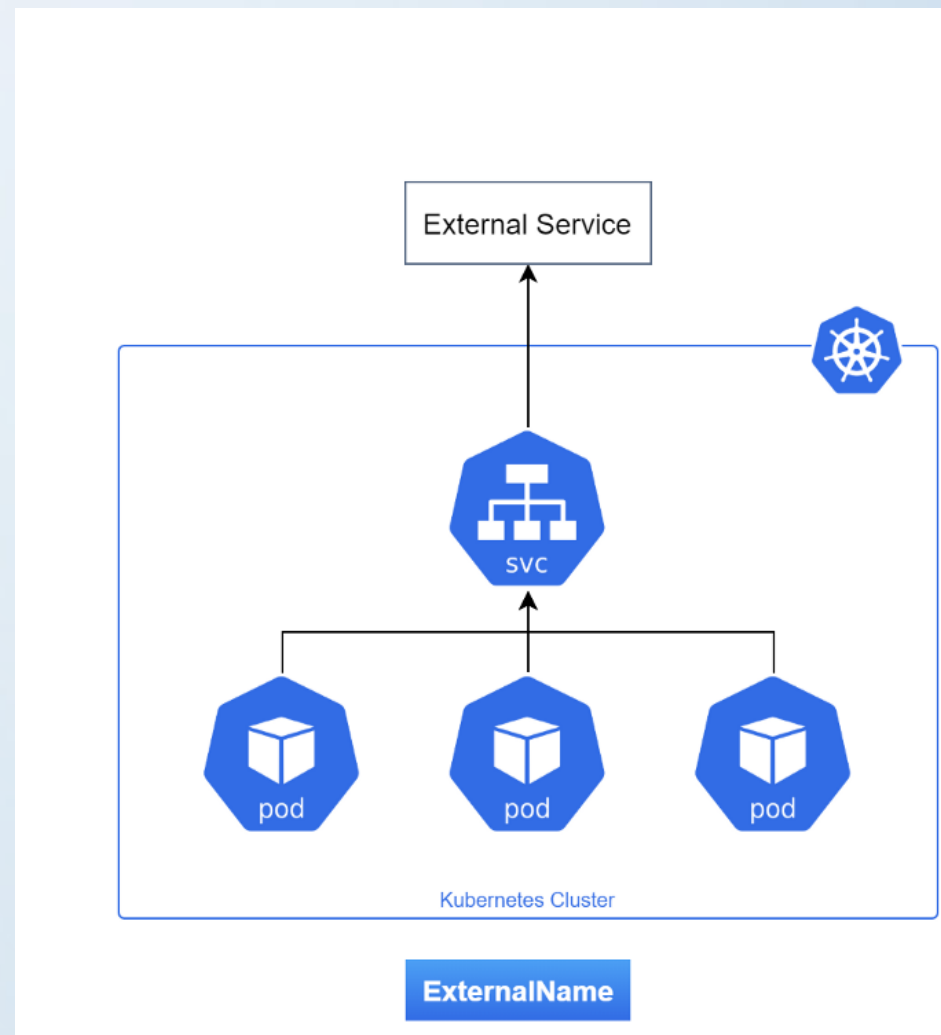
LoadBalancer

Сервис основанный на возможностях облачных провайдеров, работает на сетевом уровне, связывает выделенный ip и поды напрямую

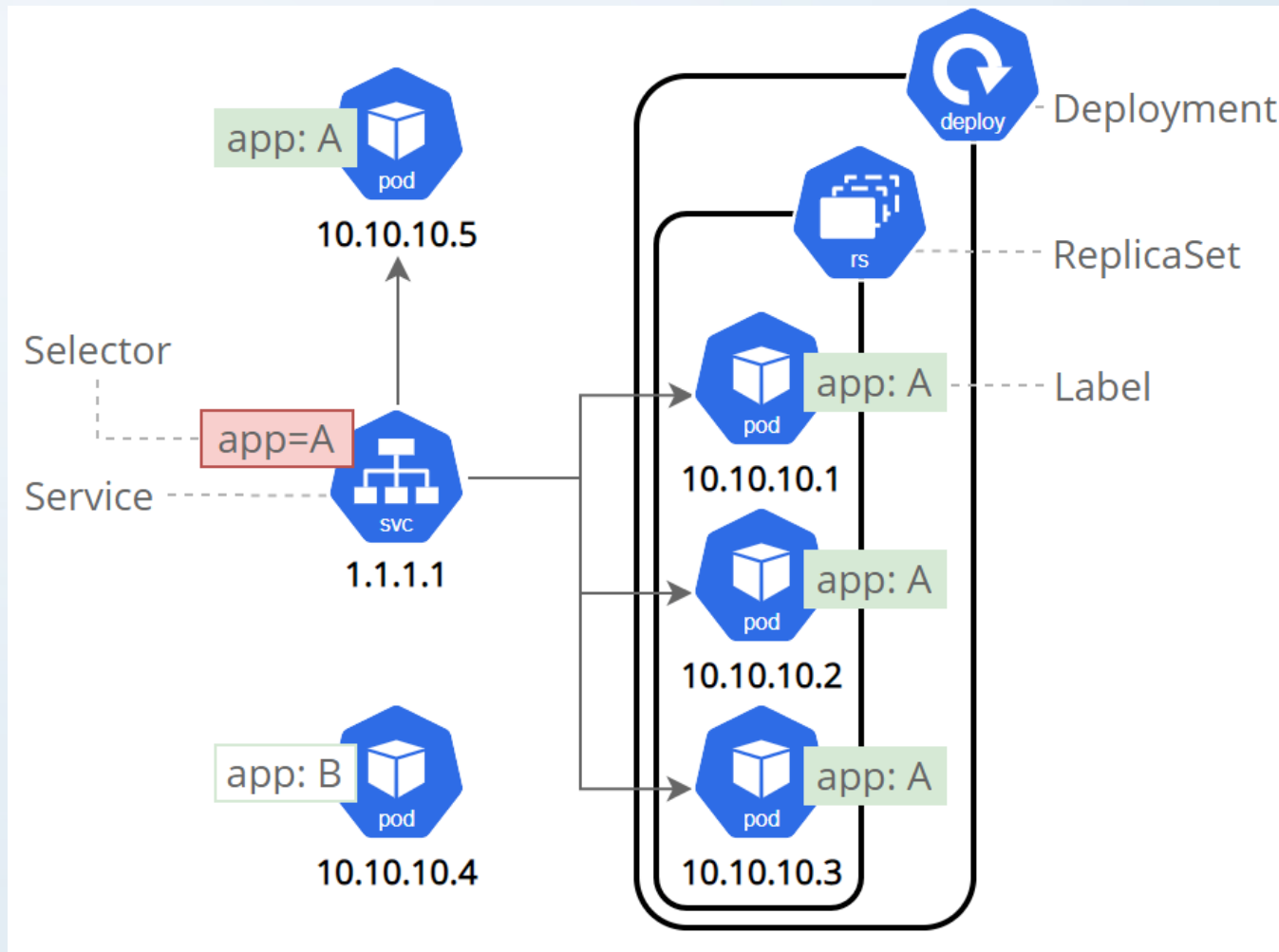


ExternalName

Просто DNS запись в кластере



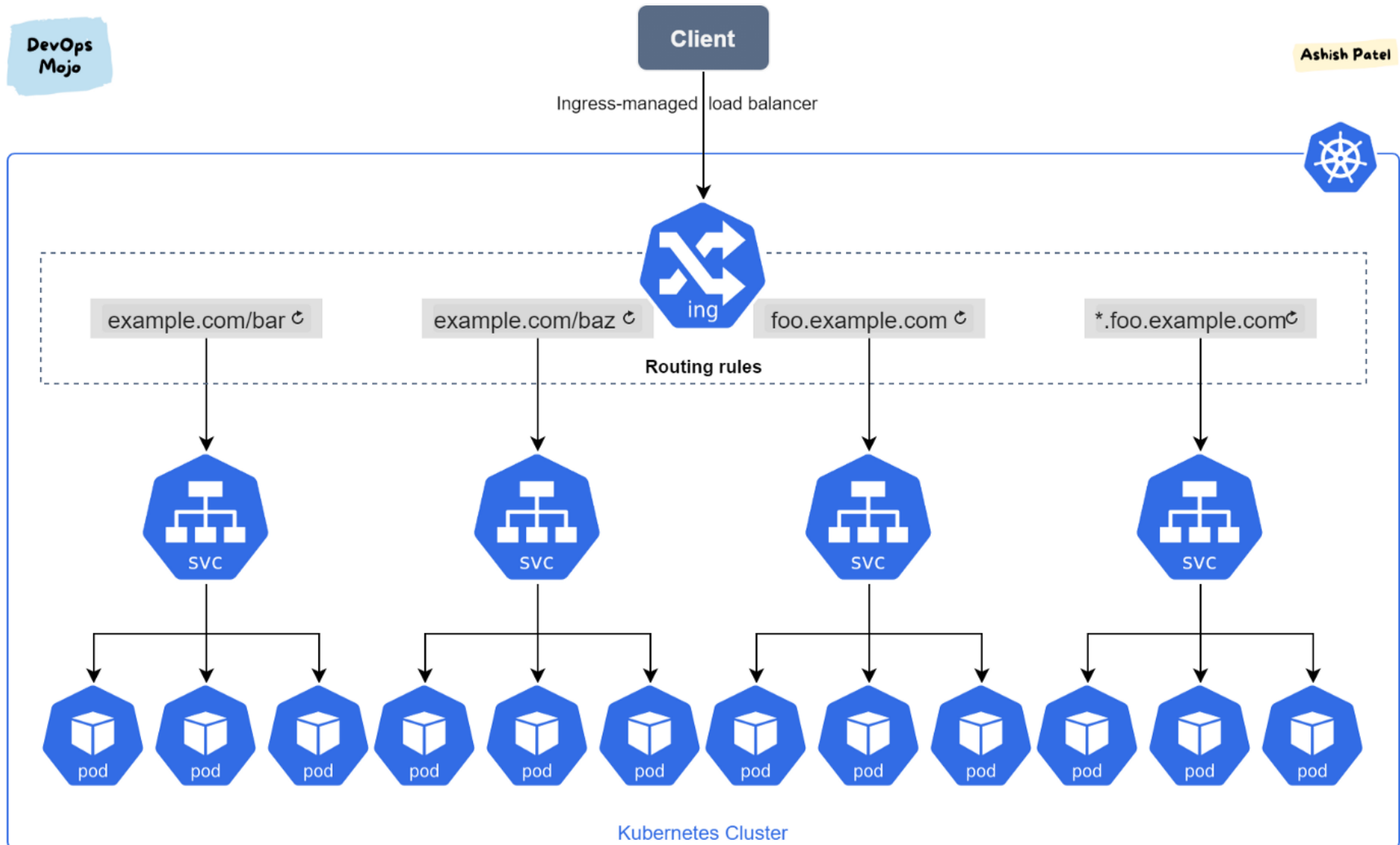
Итоговая схема



Ingress

DevOps
Mojo

Ashish Patel



Kubernetes Cluster

Конфигурация приложений

Kubeway подход подразумевает хранить конфиг отдельно от кода приложения, чтобы можно вносить правки в конфиг без долгого цикла CI/CD

Для открытых файлов конфигурации используется ConfigMap

```
k get cm
```

Для секретов используется secrets, это могут быть как файлы, так и переменные окружения

```
k get secrets
```

Базовая работа в приложениями

- Смотрим список подов находим нужный

```
k -n master get pods
```

- Смотрим логи пода

```
k -n master logs nginx-79d6646f46-7p7nz
```

- Смотрим манифест пода и состояние контейнеров

```
k -n master describe pod nginx-79d6646f46-7p7nz
```

- Заходим в под и выполняем команды

```
k -n master exec -it nginx-79d6646f46-7p7nz -- bash
```

Полезные ссылки

- <https://thesecretlivesofdata.com/raft/>
- <https://kubernetes.io/>
- <https://www.google.com/>