

де-факто стандарт

## 1 Введение

Нужен для развертывания приложений

Docker не способен развертывать приложения на 5-10 серверах, создавая по много контейнерах для разных микросервисах, как kubernetes

Kubernetes - распределяет нагрузку на разные контейнеры, создание контейнеров, перезапуск, остановку и тд

**Kubernetes (k8s)** - система управления контейнерами на различных серверах

- Автоматическое развертывание приложений в контейнерах на разных серверах
- Распределение нагрузки по нескольким серверам
- Автоматическом масштабировании развернутых приложений (в пиковые часы или уменьшать кол-во контейнеров, когда маленькая нагрузка)
- Мониторинг и проверке работоспособности контейнеров (что-то случилось k8s может пересоздать контейнер)
- Замена нерабочих контейнеров (может пересоздать контейнер заново)

Контейнеры создаются и управляются **средой запуска** контейнеров



{A0E0F378-3DF7-426B-8CFC-16AA47D6966E}.png

## Составляющие

**Pod** - самый маленький элемент в мире k8s (k8s создает поды, где внутри запускаются контейнеры)

Под каждый Pod - выделяются ресурсы

- (место на жестком диске{Том, или несколько томов контейнеры могут взаимодействовать с этими томами })
- Общий IP-адрес пода → все контейнеры внутри poda будут иметь этот IP



{36F50FE5-A76D-459D-89D8-441EDDF7E096}.png

Рекомендуется: **ОДИН ПОД - ОДИН КОНТЕЙНЕР** (это эффективней, так как если один контейнер сломается придется перезапускать под, где как раз могут находиться и другие контейнеры, которые тоже придется перезапускать)

Кластер Kubernetes

- Кластер состоит из node (сервер) - нагрузка распределяется по узлам, где сервера могут находиться в разных частях мира, а могут быть в одном месте .
- В каждом node запускаются pod
- В Кластере выбирается одна главная (master node){можно создавать резервную master node} - управляет worker-node. Master node - контролирует нагрузку и распределяет работу между другими-рабочими узлами



{F29E8D82-1C00-4F79-BDB2-0B84CD94EC50}.png

### Сервисах который запускаются на узлах kubernetes

**Kublet** - обеспечивает коммуникацию между разными узлами внутри кластера

**kube-proxy** - отвечает за сетевые ресурсы в каждом узле

**Container Runtime** - создание и контроль контейнеров на узле (включая master node) { → docker}

## 1. Kubelet (`kubelet`)

Роль:

- Это основной "агент" (агент), который работает на каждой ноде (и worker, и master, если master-нода также является worker).
- Отвечает за запуск, остановку и мониторинг контейнеров в подах (Pods), согласно описанию в `PodSpec` (из `kube-apiserver`).
- Обеспечивает связь между узлом и **Control Plane** (передает метрики, логи, статусы подов).

## 2. Kube-Proxy (`kube-proxy`)

Роль:

- Обеспечивает сетевую маршрутизацию и балансировку нагрузки между подами.
- Реализует **Service** (ClusterIP, NodePort, LoadBalancer) через правила iptables/IPVS.

## 3. Container Runtime

Роль:

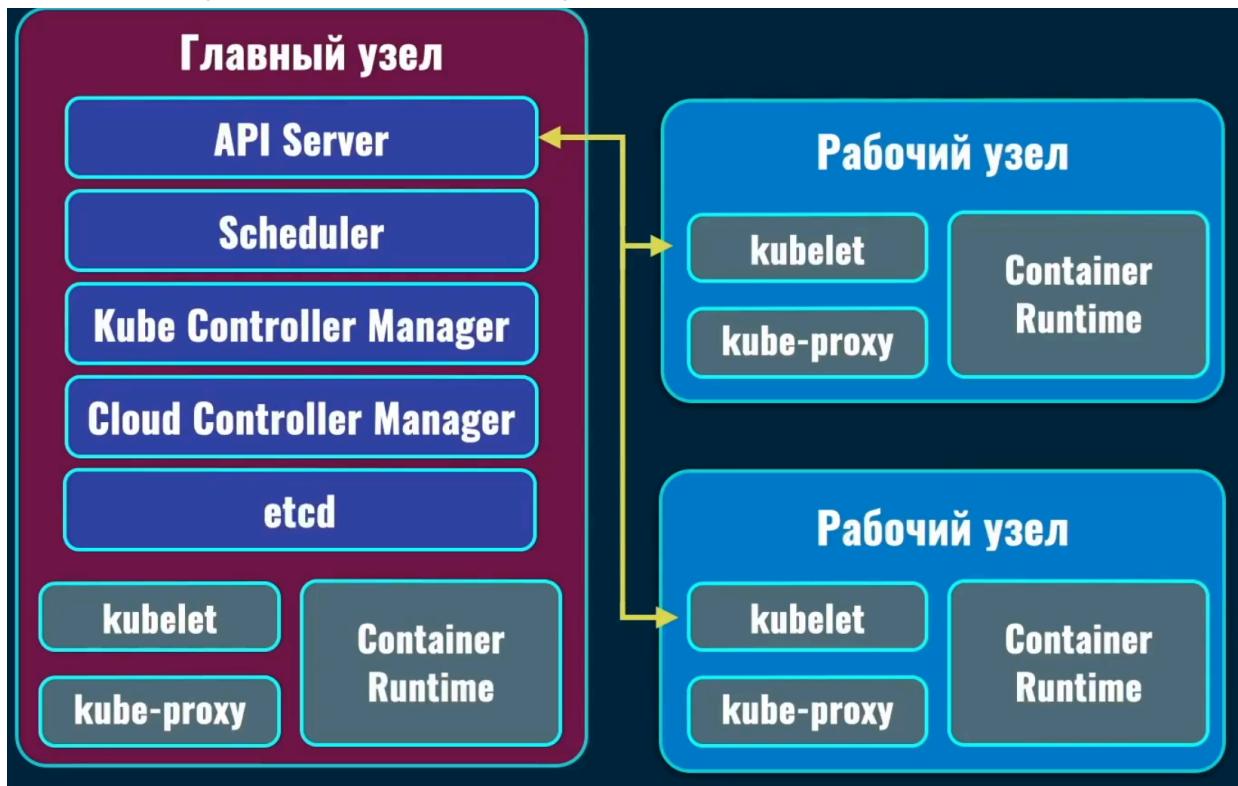
- Это низкоуровневый движок, который запускает контейнеры (например, Docker, containerd, CRI-O).
- Kubelet общается с ним через **CRI** (Container Runtime Interface).

{A7728421-E602-4435-A394-3D5721CA130B}.png

### На master node, дополнительно есть:

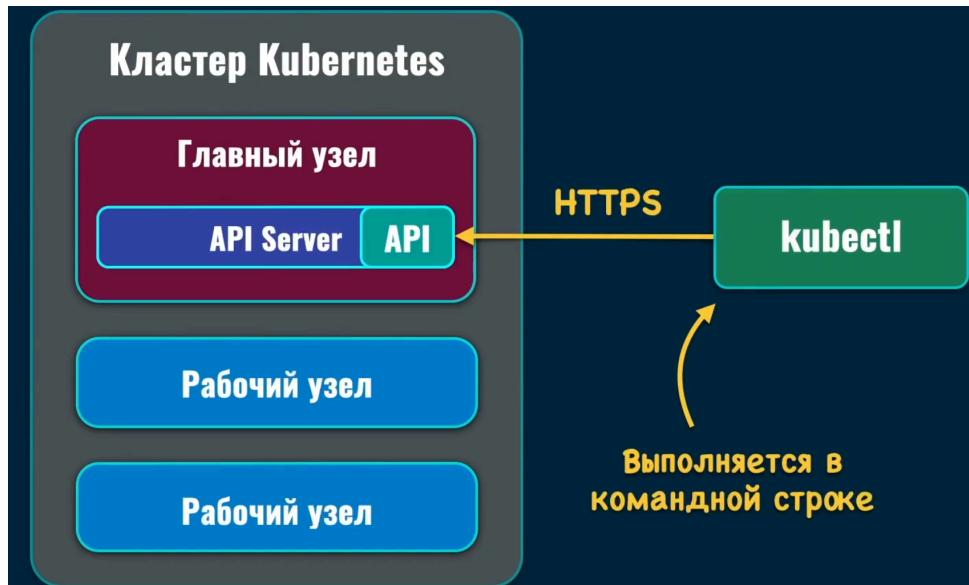
- **API Server** - рабочие узлы используя Kublet взаимодействуют с API server расположенным на главном
- **Scheduler** - управлять и распределять нагрузку между разными серверами
- **Kube Controller Manager** - контролирует все pod в кластере
- **Cloud Controller Manager** - обычно кластер запускает в облаке, и как раз этот менеджер взаимодействует с менеджером на стороне провайдера, который отвечает за создание и расширение нашего кластера
- **etcd** - отвечает за сохранение всех логов сохраняются на master node

- **dns** - для удобствами между node



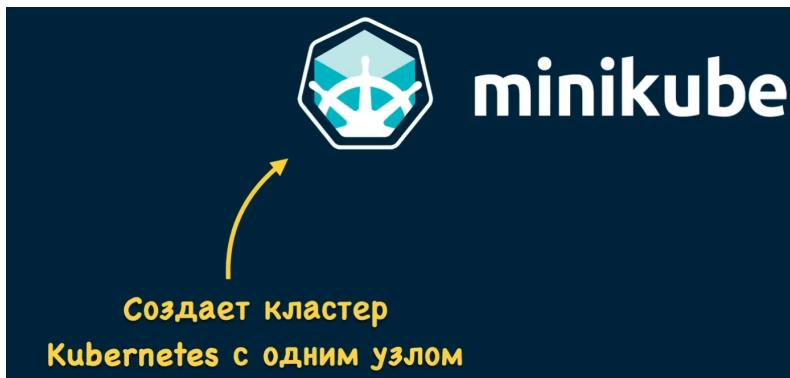
{1384AF7B-6806-4C61-B3C5-00B87BAB13A6}.png

Управление кластером с помощью утилиты kubectl – для командой строки, для управление удаленном кластере связывается через API



{B307DE89-8568-4615-B2F9-6BBC3D5B01A9}.png

## Инструменты



{7E3FD5E2-ABEC-4A90-93D9-A68B4BC86E93}.png

Для запуска узла нужен менеджер виртуальных машин или менеджера контейнеров → docker(VirtualBox, Hyper-V, podman) → создается один контейнер в рамках контейнера будут создаваться все pod: контейнер в контейнере

## 4 Создание кластера с помощью Minikube

**Minikube** — инструмент для запуска Kubernetes **локально** на одной машине.

Разворачивает **одноузловой кластер** на вашем компьютере (через виртуальную машину или Docker)

внутри Minikube - как раз будут запускаться поды

**minikube start** - запустить кластер

```
| □ ● minikube a9d268814f1e k8s-miniku 56119:22 ↗ 10.78% 1 hour ago : |
```

{5EF3D70D-0880-4EB2-BBE5-91A3BF289532}.png

**minikube status** - выводит состояние контейнера

**kubectl version** - отдельная утилита

```
Client Version: v1.32.0  
Kustomize Version: v5.5.0  
Server Version: v1.32.0
```

{745E791F-0992-44F6-A917-C992DE9E914F}.png

kubectl - уже связан с кластером (важно чтобы минерная разница была не больше 1 между клиентом и сервером)

**minikube kubectl version**

**kubectl cluster-info**

```
Kubernetes control plane is running at https://127.0.0.1:64957  
CoreDNS is running at https://127.0.0.1:64957/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

{10B5976A-5685-4CED-A5A8-1F796821B833}.png

То есть одинаково для **minikube kubectl** и для **kubectl**, то есть хотим работать с удаленными кластерами, то лучше использовать отдельную утили **kubectl**

Аlias для kubectl

kubectl →

```
`alias k=kubectl или doskey k=kubectl $*
```

Исследование кластера Minikube

**minikube kubectl get nodes** - информация о node в нашем кластере

control-plane - значит что она master node

NAME	STATUS	ROLES	AGE	VERSION
minikube	Ready	control-plane	7m11s	v1.32.0

{DD629D4D-6B50-4603-9F15-99492BC75A59}.png

**minikube kubectl get pods**

**minikube kubectl get namespaces**

NAME	STATUS	AGE
default	Active	9m7s
kube-node-lease	Active	9m7s
kube-public	Active	9m8s
kube-system	Active	9m8s

{16EAFA2D-E7C7-4DED-8767-8350C85517CC}.png

Здесь и есть запущенные поды

NAME	READY	STATUS	RESTARTS	AGE
coredns-668d6bf9bc-mg75c	1/1	Running	0	11m
etcd-minikube	1/1	Running	0	11m
kube-apiserver-minikube	1/1	Running	0	11m
kube-controller-manager-minikube	1/1	Running	0	11m
kube-proxy-ch58b	1/1	Running	0	11m
kube-scheduler-minikube	1/1	Running	0	11m
storage-provisioner	1/1	Running	1 (10m ago)	11m

{DF4B3AD3-36CA-484E-81A9-40922F87753A}.png

несколько подов, которые работают на Master-node

Чтобы подключиться к ноде нужен ip

Запущенные поды и контейнеры внутри

**minikube ip** - посмотреть ip адрес под

PS C:\Users\Delphington> ping 192.168.49.2

Обмен пакетами с 192.168.49.2 по с 32 байтами данных:  
Превышен интервал ожидания для запроса.  
Превышен интервал ожидания для запроса.  
Превышен интервал ожидания для запроса.  
Превышен интервал ожидания для запроса.

{844F957E-D303-4F75-BED8-4CA69873F4AB}.png

Мы не можем достучаться так как это внутри контейнера

**minikube ssh** - попасть внутрь node (но обычно это не надо так взаимодействия происходит через kubectl) и тут уже можем делать что хотим docker ps

Наш docker и docker внутри node - разный (node абсолютная независима от компьютера и там свой docker)

## 4 Создание подов

Deployed - состоит из многих подов (5-10)

`minikube kubectl -- run my-nginx-pod --image=nginx` - будет создан контейнер внутри пода

`minikube kubectl describe pod my-nginx-pod` - описание пода

Если мы зайдем внутрь поды, то пинг контейнера работать будет, под - доступен

```
PS C:\Users\Delphington> minikube ssh
docker@minikube:~$ ping 10.244.0.3
PING 10.244.0.3 (10.244.0.3) 56(84) bytes of data.
64 bytes from 10.244.0.3: icmp_seq=1 ttl=64 time=1.75 ms
64 bytes from 10.244.0.3: icmp_seq=2 ttl=64 time=0.101 ms
64 bytes from 10.244.0.3: icmp_seq=3 ttl=64 time=0.079 ms
64 bytes from 10.244.0.3: icmp_seq=4 ttl=64 time=0.451 ms
```

{D4D293E5-A75E-49D8-8689-E19F1D598A1B}.png

Внутри контейнера

`curl 10.244.0.3` - проверяет работоспособность контейнера в поде

```
docker@minikube:~$ docker ps | grep nginx
3352b239a38b    nginx          "/docker-entrypoint..."   10 minutes ago   U
b 10 minutes           k8s_my-nginx-pod_my-nginx-pod_default_a24bbc68-4459-4114-9266
-257752ee8916_0
66ce515c0a2d    registry.k8s.io/pause:3.9  "/pause"        11 minutes ago   U
b 11 minutes           k8s_POD_my-nginx-pod_default_a24bbc68-4459-4114-9266-257752ee
3916_0
```

{5756BE99-4E7A-4FEF-9C09-9B6538F63EE0}.png

создалось 2 контейнера, второй контейнер создается post-container - создается для каждого ПОДА, которого мы создаем, с нашим контейнером может происходить все что угодно, но забронированные ресурсы для пода всегда должны быть → поэтому нужен второй контейнер

Для каждого пода создается - подконтейнер, который отвечает за резервирования места на диске

Все контейнеры внутри пода - разделяют IP адрес самого пода

## Полное описание pod

```
PS C:\Users\Delphington> minikube kubectl -- get pods -o wide
NAME      READY   STATUS    RESTARTS   AGE     IP           NODE   NOMINATED NODE   READINESS GATES
my-nginx-pod  1/1    Running   0          122m   10.244.0.3   minikube <none>        <none>
```

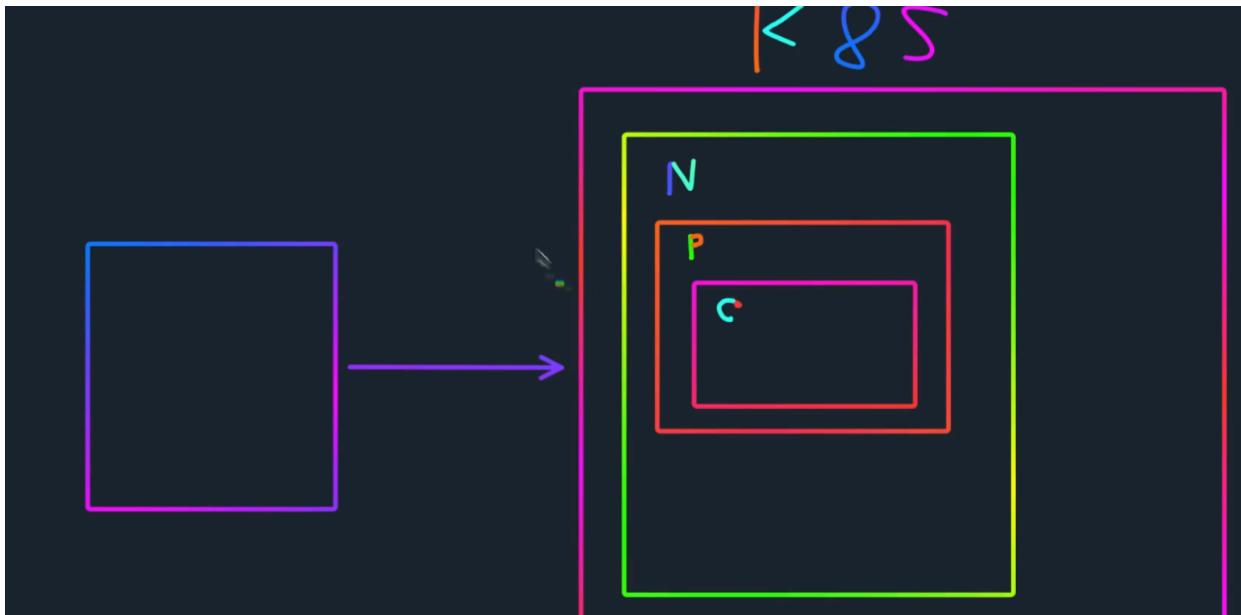
{9C8BE548-A584-4F98-8293-B0FF6266083F}.png

ip node != ip pod

### Почему мы не можем достучаться до внутреннего пода по айпи ?

Слева компьютер наш, он пытается законектитись с кластером k8c, где лежит нода, в который лежит под, в которой лежит отдельный контейнер

Connection происходит именно с pod



{9AB00281-2984-4146-8925-BD0D424AA95C}.png

N - node

P - pod

C - container

# 5 Создания деплоимента

## 1. Кластер (Cluster)

- Включает в себя **ноды (узлы)** и **Control Plane** (управляющие компоненты: `kube-apiserver`, `scheduler`, `controller-manager`, `etcd`).

## 2. Нода (Node)

- Физическая или виртуальная машина, на которой работают **поды (Pods)**.
- Может быть:
  - Worker Node** — запускает пользовательские поды.
  - Master Node** (если не используется отдельный Control Plane).

## 3. Пространство имён (Namespace)

- Виртуальный "отдел" внутри кластера для изоляции ресурсов (например, `default`, `kube-system`, `prod`, `dev`).
- Deployment** и другие объекты создаются внутри неймспейса.

## 4. Deployment

- Контроллер, который управляет **репликами подов (Pods)** через `ReplicaSet`.
- Отвечает за:
  - Развертывание приложения.
  - Обновление (*rolling update*) и откат (*rollback*).
  - Масштабирование (увеличение/уменьшение числа подов).



{CD0E43B4-A4C5-4FE9-92EB-699DACACE437}.png

**Deployment** — это объект Kubernetes, который управляет **развертыванием и обновлением** ваших приложений (Pods). Он обеспечивает:

- Декларативное описание** желаемого состояния приложения (сколько реплик, какой образ контейнера).
- Автоматическое развертывание и масштабирование** Pod'ов.
- Обновление без downtime** (*rolling updates*) и откат (*rollback*) при ошибках.

## 1. Зачем нужен Deployment?

Без Deployment'a пришлось бы вручную создавать и обновлять Pod'ы, что неудобно и ненадежно.

{E38FBE34-0E75-463E-98C2-D7BFF282A2D1}.png

У всех подов в деплоименте будут одинаковые образы

```
minikube kubectl -- create deployment my-nginx-deploy --  
image=nginx
```

```

PS C:\Users\Delphington> .\kubectl create deployment my-nginx-deploy --image=nginx
deployment.apps/my-nginx-deploy created
PS C:\Users\Delphington> .\kubectl get pods
NAME           READY   STATUS            RESTARTS   AGE
my-nginx-deploy-75488fc988-cgdmk  0/1    ContainerCreating   0          9s
PS C:\Users\Delphington> .\kubectl get pods
NAME           READY   STATUS            RESTARTS   AGE
my-nginx-deploy-75488fc988-cgdmk  1/1    Running          0          33s
PS C:\Users\Delphington>

```

{AF6F2D1E-7893-4600-A7F2-9385BF9415A5}.png

## Описание deploy

```

> k describe deploy my-nginx-deploy
Name:                   my-nginx-deploy
Namespace:              default
CreationTimestamp:      Sat, 25 Nov 2023 18:27:43 +0000
Labels:                 app=my-nginx-deploy
Annotations:            deployment.kubernetes.io/revision:
Selectors:              app=my-nginx-deploy

```

{48A73642-DDCF-4158-8B82-B796D89CE4EA}.png

Есть деплоймент в рамках деплоймента было создано набор replic из ReplicaSet, а потом событие

Селектор - нужен чтобы сопоставить поды с deployment

deployment - replic - pod

Каждый под является репликой

## Увеличение реплик

```

PS C:\Users\Delphington> .\kubectl scale deployment my-nginx-deploy --replicas=3
deployment.apps/my-nginx-deploy scaled
PS C:\Users\Delphington> .\kubectl get deploy
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
my-nginx-deploy  1/3       3           1           84m

```

{681CBCE7-304E-4241-BC09-77427A9B8DE8}.png

## У всех подов свои ip-адреса

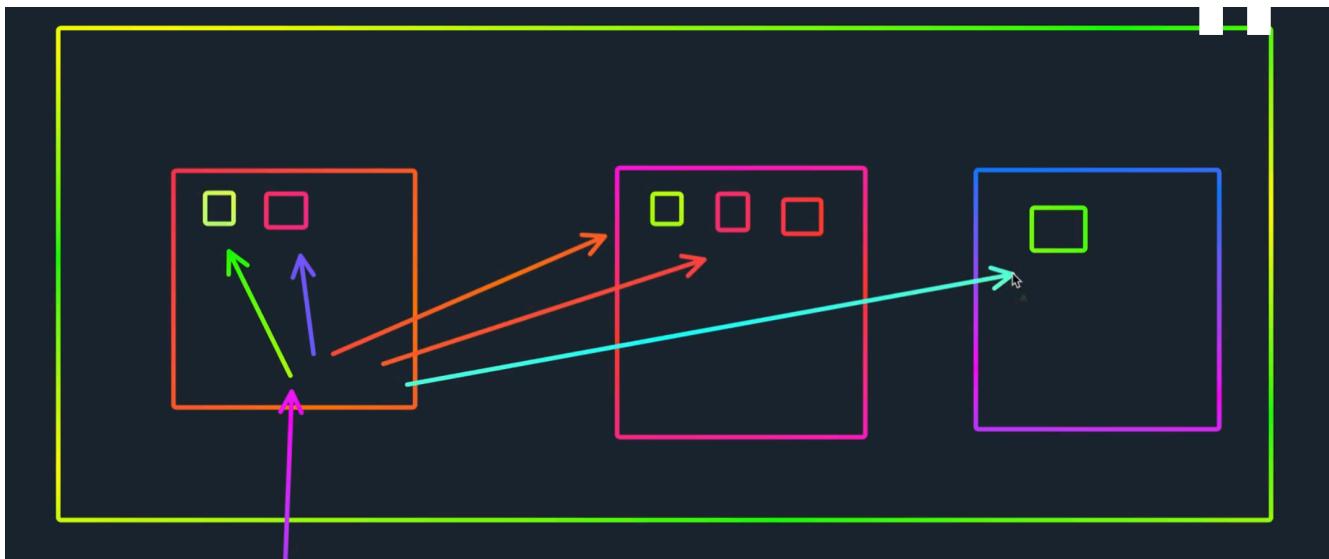
```

PS C:\Users\Delphington> .\kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP           NODE   NOMINATED NODE   READINESS GATES
my-nginx-deploy-75488fc988-44sn6  1/1    Running   0          14m  10.244.0.6  minikube <none>        <none>
my-nginx-deploy-75488fc988-5pbz5  1/1    Running   0          14m  10.244.0.5  minikube <none>        <none>
my-nginx-deploy-75488fc988-cgdmk  1/1    Running   0          98m  10.244.0.4  minikube <none>        <none>

```

{1FE9C6A7-8836-4CB3-9A77-5FFF316FFB1B}.png

чтобы достучаться до пода → нужна перейти внутрь любой ноды  
Зайдя в ноду можем достучаться в любой под (но снаружи эти поды не доступны)



{C216E108-1AB2-4C9C-ABAD-0D5D6F86F7F5}.png

салатовый - кластер

квадратики - ноды

внутри квадратиков поды

Если в deployment мы сказали что должно быть 15 подов, и удалим один из создавшихся подов, то kubernetes создаст новый

Как мы можем подключиться к pod если у них все время динамические IP адреса, все поды = реплики. Пользователю не важно в какой именно под он попадет, чтобы мы смогли снаружи класстера подключаться к какому-нибудь поду используются **сервисы**

Сервисы - дополнения к деплоинментам. Используя сервисы, можно подключаться к любому виду подов внутри определенного деплоинмента используя один **IP** адресс.

## 6 Создание сервисов

все поды + все деплоименты

k get pods				
NAME	READY	STATUS	RESTARTS	AGE
my-nginx-deploy-785cb5c9f4-fzm62	1/1	Running	0	16h
my-nginx-deploy-785cb5c9f4-kb55t	1/1	Running	0	15h
my-nginx-deploy-785cb5c9f4-qbpxp	1/1	Running	0	15h
my-nginx-deploy-785cb5c9f4-r2lvk	1/1	Running	0	15h
my-nginx-deploy-785cb5c9f4-t4lk2	1/1	Running	0	15h

k get deploy				
NAME	READY	UP-TO-DATE	AVAILABLE	AGE
my-nginx-deploy	5/5	5	5	16h

{27A8F546-2F71-4A28-9EAC-F625AF05A8F8}.png

## Kluster IP

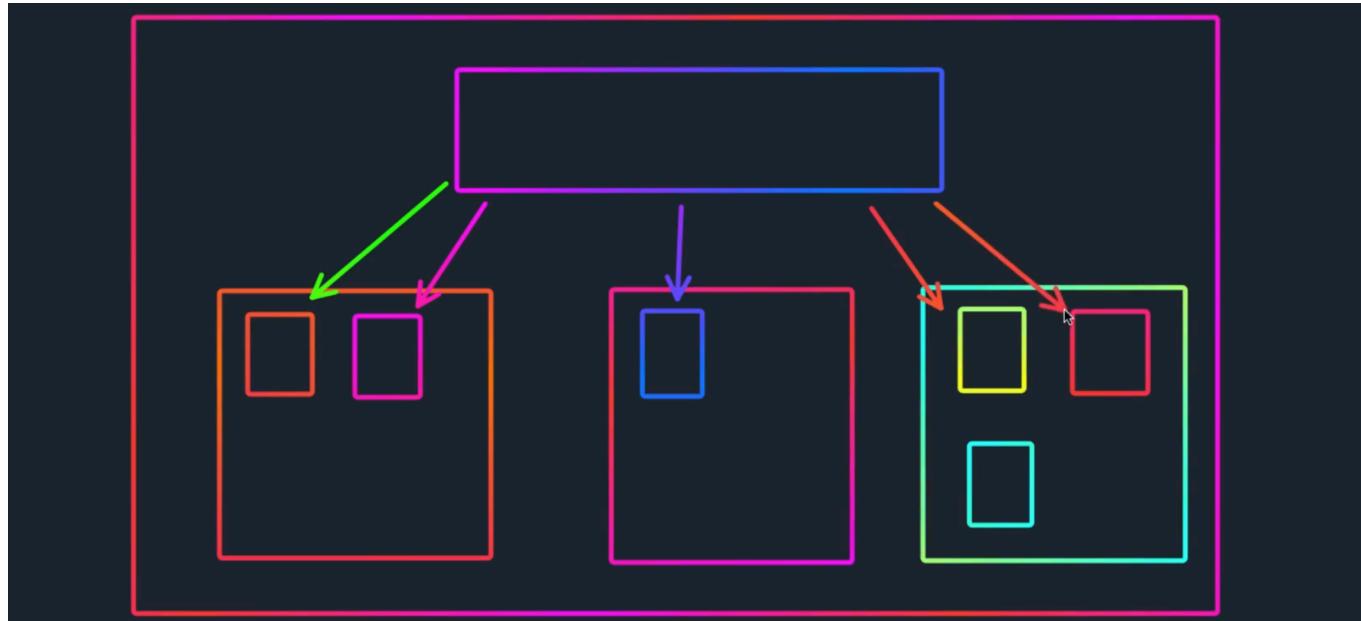
Сервис позволяет используя один IP-адрес подключить к любому поду (но заранее не знаем к какому именно поду мы подключимся)

Самый большой - класстер

поменьше - ноды

меньше - поды, поды находятся в одном деплоименте

прямоугольник сверху - сервис



{E89D036C-1D75-4753-8D9B-720ADE546DD6}.png

То есть у нас расположены поды на разных нодах, но в одном deployment, и через сервис(самый верхний прямоугольничек)

создается виртуальный IP, где мы можем достучаться до любого пода

## Получение сервиса

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	3h6m

{7E89E49A-5F7F-4036-9239-7F5E0D815EEF}.png

kubernetes - сложный сервис, нужен чтобы направить трафик внутри кластера к IP - сервису на master node

```
> k describe service kubernetes
Name:                  kubernetes
Namespace:             default
Labels:                component=apiserver
                      provider=kubernetes
Annotations:           <none>
Selector:              <none>
Type:                 ClusterIP
IP Family Policy:     SingleStack
IP Families:          IPv4
IP:                   10.96.0.1
IPs:                  10.96.0.1
Port:                 https 443/TCP
TargetPort:            8443/TCP
Endpoints:            192.168.49.2:8443
Session Affinity:     None
Events:               <none>
```

{496A07B8-8D81-4B0F-90AA-E56FE8932450}.png

при обращении на кластер идет редирект запроса на нужный pod

## Создания сервиса

```
.\kubectl expose deploy my-nginx-deploy --port=8080 -  
-target-port=80
```

8080 - порт на который хотим чтобы перенаправлялся трафик, 80 - порт nginx внутри пода

```
PS C:\Users\Delphington> .\kubectl get services  
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP  PORT(S)    AGE  
kubernetes     ClusterIP  10.96.0.1    <none>       443/TCP    3h17m  
my-nginx-deploy ClusterIP  10.98.252.201  <none>       8080/TCP   91s  
{2DCC80BB-37C1-4700-9755-242DE5928776}.png
```

И теперь мы можем подключаться к deployment по IP 10.98.252.201 по порту 8080, и сервис будет перенаправлять запросы на один из подов в deployment

но ClusterIP - не подходит, если нужен нужно подключиться к нему снаружи

переходим внутрь классера и выполняем запрос

```
> minikube ssh  
docker@minikube:~$ curl 10.108.142.1:8080
```

```
{0B6F6F58-0394-4A13-8360-84FA271DC2B1}.png
```

ну там пришел ответ!

---

Сервис NodePort - открывает определенный порт на каждом узле, где есть deployment

```
.\kubectl expose deploy my-nginx-deploy --  
type=NodePort --port=8888 --target-port=80
```

Создаем сервис, с типом NodePort по порту 8888

```
PS C:\Users\Delphington> .\kubectl get services  
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP  PORT(S)        AGE  
kubernetes     ClusterIP  10.96.0.1    <none>       443/TCP       3h37m  
my-nginx-deploy  NodePort   10.108.164.81  <none>       8888:30937/TCP  2m32s  
PS C:\Users\Delphington>
```

{3F7CF0BF-7DEB-4505-B4F4-130B6E4A715A}.png

Порт был открыт на узле 30937, было несколько нод - было бы несколько портов

Благодаря такому сервису, мы можем подключиться снаружи к деплоименту → зная IP node и порт 30937

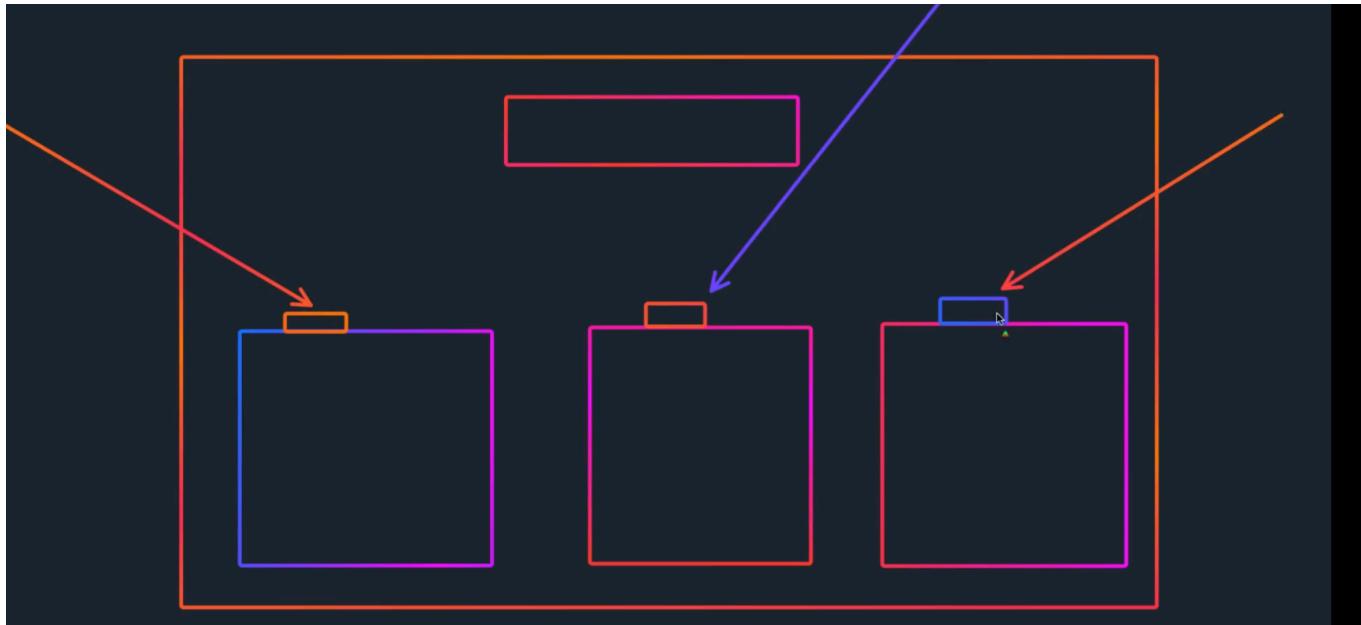
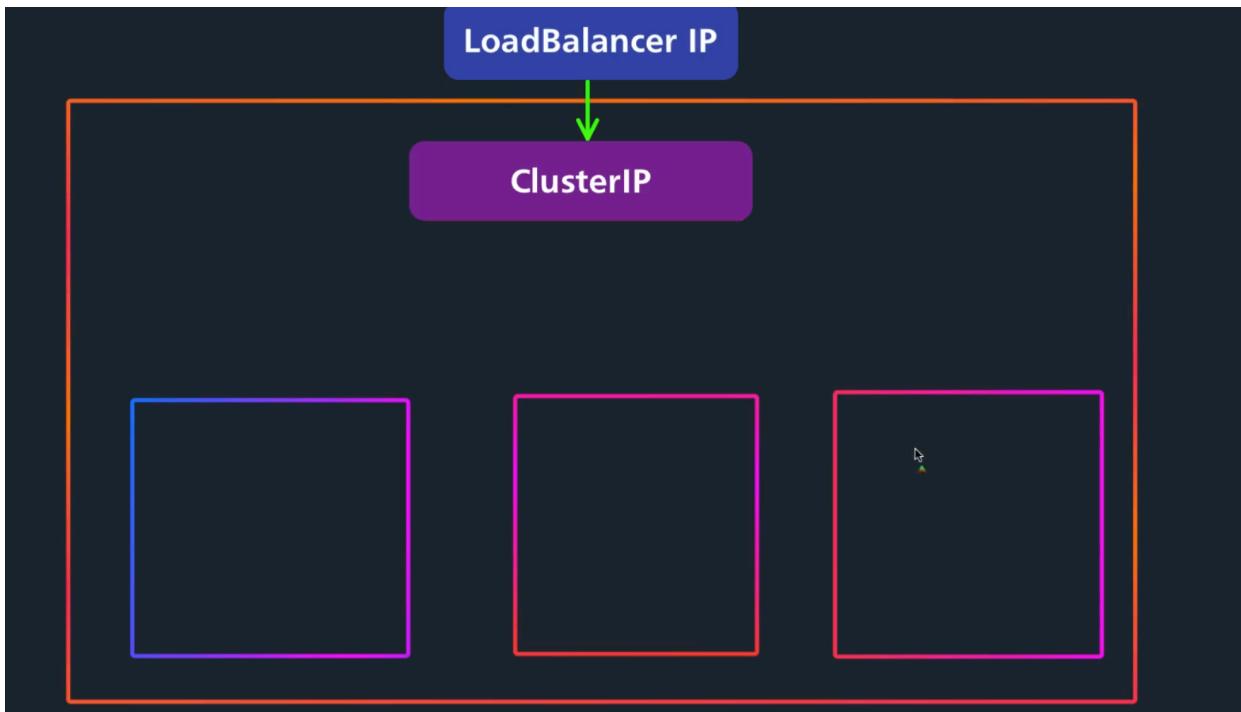


image-1.png

### Сервис LoadBalance

Создается external IP, который связывается с IP сервисом внутри кластера K8s



{06C789C1-D3A2-424E-9FA4-3252A16ECF8B}.png

Для клиента предоставляется 1 IP адрес, который используется для подключения к подам

Для каждого deployment - создается отдельный IP-адрес

```
> minikube tunnel
```

Тунель для подключения через внешний IP ко внутреннему кластеру k8s

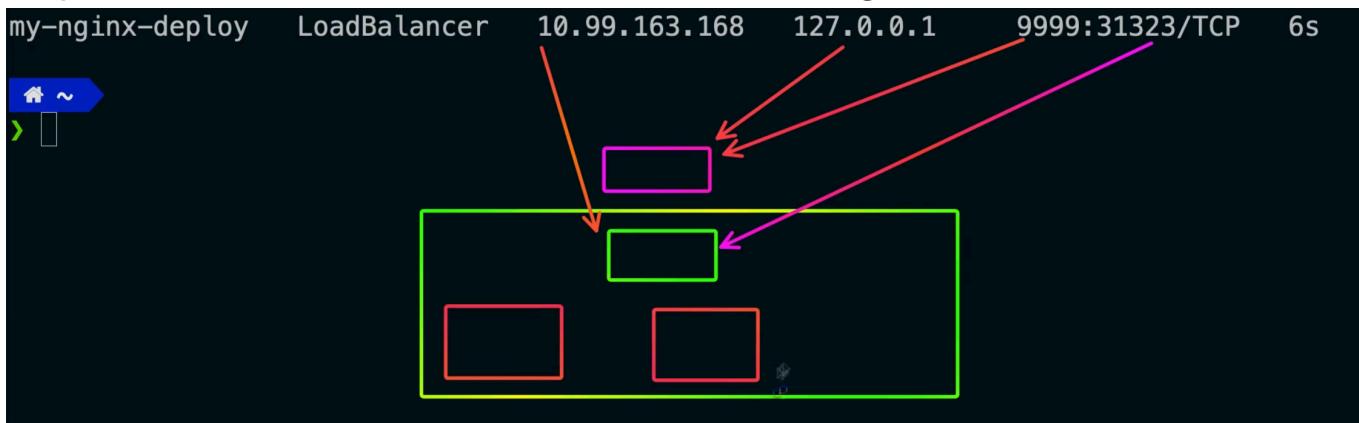
```
.\kubectl expose deploy my-nginx-deploy --type=LoadBalancer -  
-port=9999 --target-port=80
```

```
PS C:\Users\Delphington> .\kubectl get services
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP     PORT(S)        AGE
kubernetes     ClusterIP  10.96.0.1   <none>        443/TCP       3h52m
my-nginx-deploy LoadBalancer  10.106.33.229  127.0.0.1    9999:31833/TCP  24s
PS C:\Users\Delphington>
```

{422DECA7-E26A-42A6-BEAF-D772EE2BAA27}.png

External-IP - появился благодаря тому, что мы запустили тунель

<http://localhost:9999/> - позволяет войти в nginx



{5884EBE9-6E65-4541-908D-D0BA3C9ABAFA}.png

Создания deployment с dockerhub - нашего образа

```
.\kubectl create deploy k8s-web-hello --image=delphington/k8s-web-hello-ru:latest
```

## 7 Создания веб приложения

k get pods					
NAME	READY	STATUS	RESTARTS	AGE	
k8s-web-hello-56f7558d6c-hzg8q	1/1	Running	0	102s	

{55257105-1FB6-4B5A-9E33-04A4EB663279}.png

k8s-web-hello-56f7558d6c-hzg8q

{E0DC4BBE-7B83-4D90-A382-3000CA5FC2AF}.png

что выделенно - это replicaSet, а что не выделено уникальный индификатор пода

Создем сервис для нашего деплоимента

1. Создаем тунель, чтобы можно было обращаться к классеру из вне

```
> minikube tunnel
✓ Tunnel successfully started

✖ NOTE: Please do not close this terminal as this process must stay alive for
the tunnel to be accessible ...
```

{384FA5A1-BD50-4461-A5A2-DA0D36174EDD}.png

## 2. Создаем сервис

```
> k expose deploy k8s-web-hello --type=LoadBalancer --port=3333 --target-port=3
000
service/k8s-web-hello exposed
```

{61F4DB4D-497D-43FA-AC8F-17D6073F7B95}.png

## 3. Все сервисы

k get svc						
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	
k8s-web-hello	LoadBalancer	10.97.246.92	127.0.0.1	3333:32725/TCP	6s	
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	25h	

{6DA0133B-D36E-4CC3-9E08-E3545650EE50}.png

## 4. Все порты

k get pods					
NAME	READY	STATUS	RESTARTS	AGE	
k8s-web-hello-56f7558d6c-hzg8q	1/1	Running	0	20m	

{270A32F1-D16F-4CA1-85E4-230F48630FAB}.png

## 5. Увеличим количество реплик

```
> k scale deploy k8s-web-hello --replicas=7
deployment.apps/k8s-web-hello scaled
```

minikube ~

k get pods					
NAME	READY	STATUS	RESTARTS	AGE	
k8s-web-hello-56f7558d6c-dk4jq	1/1	Running	0	3s	
k8s-web-hello-56f7558d6c-fbt9x	1/1	Running	0	3s	
k8s-web-hello-56f7558d6c-hzg8q	1/1	Running	0	23m	
k8s-web-hello-56f7558d6c-n2lwn	1/1	Running	0	3s	
k8s-web-hello-56f7558d6c-ntv6f	1/1	Running	0	3s	
k8s-web-hello-56f7558d6c-p2mlf	1/1	Running	0	3s	
k8s-web-hello-56f7558d6c-x2mzv	1/1	Running	0	3s	

{9711C9ED-5A52-4210-A105-7C81CC887C9B}.png

# 8 Обновления приложения

kubectl rollout status deploy k8s-web-hello

1. deployment "k8s-web-hello" successfully rolled out
  - Деплоймент (развёртывание) с именем `k8s-web-hello` успешно завершил обновление.
  - Все новые поды (Pods) созданы, старые — удалены (если использовалась стратегия `RollingUpdate`).
  - Приложение работает в желаемом состоянии (Desired State).

{19B3EAAB-FBCB-4E37-BF67-744D21AEA9A4}.png

Замена образов:

```
> k set image deploy k8s-web-hello k8s-web-hello-ru=bstashchuk/k8s-web-hello-ru:2.0
.0
deployment.apps/k8s-web-hello image updated
```

{4F681D61-AFD6-4E5F-9F64-4E153D0756F7}.png

Часть запущена с ново версией, часть обновляется (rolling update)

```
> k get pods
NAME                               READY   STATUS    RESTARTS   AGE
k8s-web-hello-56f7558d6c-dk4jq   1/1     Terminating   0          6h31m
k8s-web-hello-56f7558d6c-fbtx9   1/1     Terminating   0          6h31m
k8s-web-hello-56f7558d6c-g2q4g   1/1     Terminating   0          6h28m
k8s-web-hello-56f7558d6c-n2lwn   1/1     Terminating   0          6h31m
k8s-web-hello-56f7558d6c-ntv6f   1/1     Terminating   0          6h31m
k8s-web-hello-56f7558d6c-p2mlf   1/1     Terminating   0          6h31m
k8s-web-hello-56f7558d6c-x2mzv   1/1     Terminating   0          6h31m
k8s-web-hello-9f9658788-j5kgc   1/1     Running     0          14s
k8s-web-hello-9f9658788-k7zb2   1/1     Running     0          16s
```

{807A7ADA-C4B5-4930-954E-5F89E5D4E0E4}.png

После деплоймента, показывается история реплик

```
> k describe deploy k8s-web-hello
```

{A04EEF7B-7C7D-4020-B420-DA9A23297FDD}.png

```
OldReplicaSets:  k8s-web-hello-56f7558d6c (0/0 replicas created)
NewReplicaSet:   k8s-web-hello-9f9658788 (7/7 replicas created)
Events:
```

{C68E5822-94C3-4CED-A3BA-0ED4CE840DF6}.png

## 9 YAML файлы для деплойментов и сервисов

Все мы делали в императивном виде

## декларативный YAML

YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-web-hello # название деплоимента
spec:
  replicas: 5
  selector: ## поиск для подов
    matchLabels:
      app: k8s-web-hello
  template:
    metadata:
      labels:
        app: k8s-web-hello
    spec: # спецификация для контейнеров
      containers:
        - name: k8s-web-hello # список с "-"
          image: bstashchuk/k8s-web-hello
      resources:
        limits:
          memory: "128Mi"
          cpu: "250m"
      ports:
        - containerPort: 3000
```

Создаем деплоимент

```
> k apply -f deployment.yaml
deployment.apps/k8s-web-hello created
```

{D6E6D314-C1A4-4720-815F-296B05A17BF5}.png

```
> k get deploy  
NAME READY UP-TO-DATE AVAILABLE AGE  
k8s-web-hello 1/1 1 1 11s
```

{D0EE0FDA-9DB2-4A9B-A7D6-C9DAE1101FC2}.png

```
> k get pods  
NAME READY STATUS RESTARTS AGE  
k8s-web-hello-5978b7c96b-b7cdn 1/1 Running 0 22s
```

{03CC0640-4A9D-476F-BE8F-E1A3A6010F70}.png

## Увеличиваем количество подов:

replicas:

```
1 apiVersion: apps/v1  
2 kind: Deployment  
3 metadata:  
4   name: k8s-web-hello  
5 spec:  
6   replicas: 5  
7   selector:  
8     matchLabels:  
9       app: k8s-web-hello  
10  template:  
11    metadata:  
12      labels:  
13        app: k8s-web-hello  
14    spec:  
15      containers:  
16        - name: k8s-web-hello  
17          image: hstachchuk/k8s-web-hello:ru
```

{ABF838D3-1D4E-4477-9BD1-A261DF6931A7}.png

## Применяем изменения

```
> k apply -f deployment.yaml  
deployment.apps/k8s-web-hello configured
```

{58FF7F50-E641-49EC-99DE-0ADC78F83920}.png

## И теперь 5 подов

```
> k get pods  
NAME READY STATUS RESTARTS AGE  
k8s-web-hello-5978b7c96b-8bw99 1/1 Running 0 39s  
k8s-web-hello-5978b7c96b-8cfgs 1/1 Running 0 39s  
k8s-web-hello-5978b7c96b-b7cdn 1/1 Running 0 12m  
k8s-web-hello-5978b7c96b-gdvqh 1/1 Running 0 39s  
k8s-web-hello-5978b7c96b-m7qvz 1/1 Running 0 39s
```

{ED141D44-A6A7-42CC-AAB2-683428A959A3}.png

## Создание сервиса service.yaml

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: k8s-web-hello
5  spec:
6    type: LoadBalancer
7    selector:
8      app: k8s-web-hello
9    ports:
10   - port: 3030
11     targetPort: 3000
```

{D7CA9642-F0BA-42A2-A5C9-FD23F7756167}.png

**kubectl apply -f service.yaml** - применяем

**minikube dashboard** - UI представление

## Очистка данных

```
> k delete -f deployment.yaml -f service.yaml
deployment.apps "k8s-web-hello" deleted
service "k8s-web-hello" deleted

📂 ~/Desktop/k8s
> k get svc
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kubernetes     ClusterIP   10.96.0.1      <none>          443/TCP     2d1h

📂 ~/Desktop/k8s
> k get deploy
No resources found in default namespace.
```

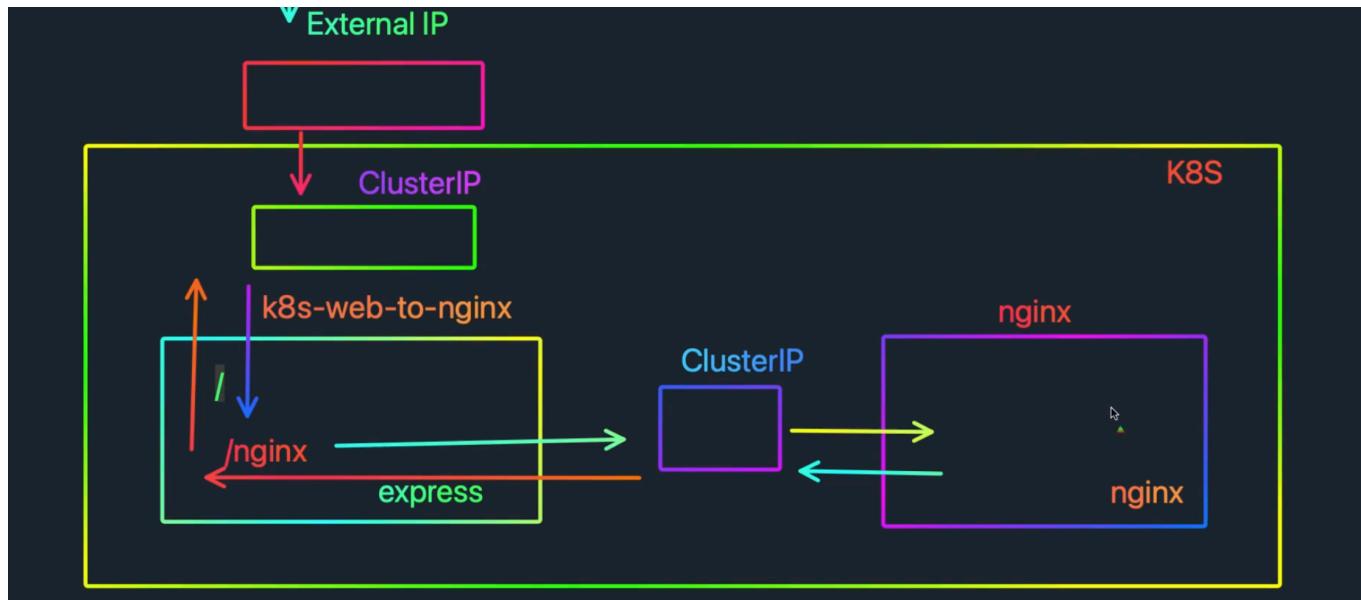
{4E588EE1-F061-4A52-9912-1A5E246DB94D}.png

# 10 Создание двух деплоиментов

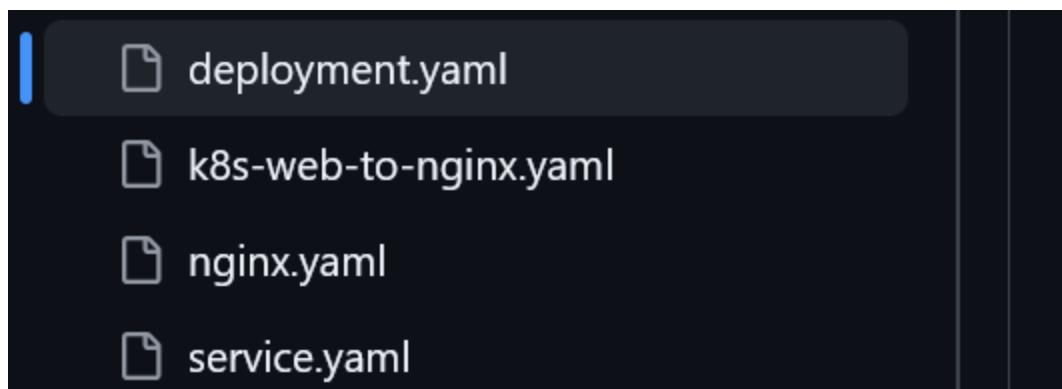
Создаем два деплоимента

справа nginx - базовый образ + сервис clusterIP (можно обращаться только внутри класстера)

слева - деплоймент наш кастомный имеет 2 endpoints для перенаправления на nginx и на себя, и создан сервис loadbalancer для обращения из вне



{B466B2B9-FE79-4FF8-AB54-A9DA8C362A65}.png



{6233D847-545B-465C-8AEA-41F4E50B9D07}.png

## Для первого деплоймента

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-web-hello
spec:
  replicas: 5
  selector:
    matchLabels:
      app: k8s-web-hello
  template:
    metadata:
      labels:
        app: k8s-web-hello
    spec:
      containers:
        - name: k8s-web-hello
          image: bstashchuk/k8s-web-hello
      resources:
        limits:
          memory: "128Mi"
          cpu: "250m"
      ports:
        - containerPort: 3000
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: k8s-web-hello
spec:
  type: LoadBalancer
  selector:
    app: k8s-web-hello
  ports:
    - port: 3030
      targetPort: 3000
```

### Для второго:

k8s-web-to-nginx.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: k8s-web-to-nginx
spec:
  type: LoadBalancer
  selector:
    app: k8s-web-to-nginx
  ports:
    - port: 3333
      targetPort: 3000
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-web-to-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8s-web-to-nginx
  template:
    metadata:
      labels:
        app: k8s-web-to-nginx
    spec:
      containers:
        - name: k8s-web-to-nginx
          image: bstashchuk/k8s-web-to-nginx
      resources:
```

```
limits:  
    memory: "128Mi"  
    cpu: "250m"  
ports:  
- containerPort: 3000
```

nginx.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - port: 80
--- # ниже добавляем сервис
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx
    resources:
      limits:
        memory: "128Mi"
```

```
    cpu: "125m"
  ports:
    - containerPort: 80
```

Создаем

```
~/Desktop/k8s ➔
> k apply -f k8s-web-to-nginx.yaml -f nginx.yaml
deployment.apps/k8s-web-to-nginx created
service/k8s-web-to-nginx created
deployment.apps/nginx created
service/nginx created
```

{9CF646E5-4121-4FA0-A9CA-85EE22EB2AF8}.png