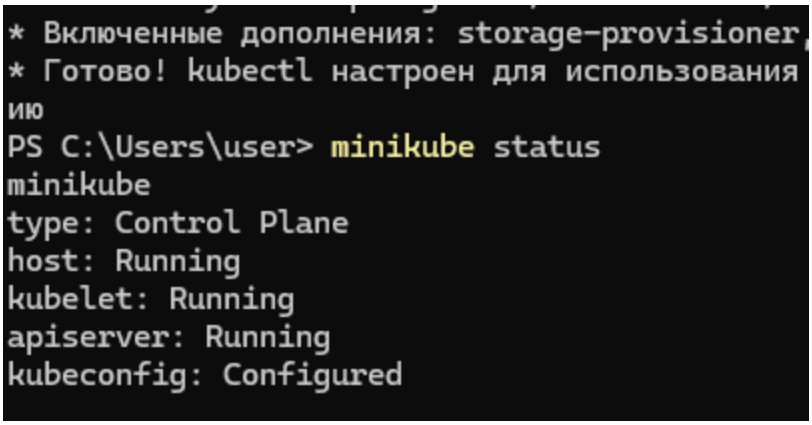
# Minikube

## Создание кластера K8S (Kubernetes) с помощью Minikube

Пишем:

```
minikube start
```

после установки




Это нормальное состояние кластера Kubernetes,
В докере появятся image и container от minikube. (запустился внутри одного контейнера докер)
Внутри этого контейнера будут запускаться все ПОДЫ Kubernetes, внутри которых будут запускаться контейнеры

Информация о ноде в нашем кластере:

```
kubectl get nodes
```

> kubectl - можно установить отдельно на комп, а вообще в миникуб встроенный есть.

Информация о ПОДАХ:

```
kubectl get pods

kubectl get pods -o wide  "- так можно ip еще увидеть"
```

Системное пространство имён:
(тут есть запущенные поды)

```
kubectl get namespaces
```

Посмотреть:

```
kubectl get pods --namespace=kube-system
```

```
PS C:\Users\user> kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   37m   v1.32.0
PS C:\Users\user> kubectl get pods
No resources found in default namespace.
PS C:\Users\user> kubectl get namespaces
NAME              STATUS   AGE
default           Active   37m
kube-node-lease   Active   37m
kube-public       Active   37m
kube-system       Active   37m
PS C:\Users\user> kubectl get pods --namespace=kube-system
NAME                               READY   STATUS    RESTARTS      AGE
coredns-668d6bf9bc-xfxdf           1/1     Running   0             37m
etcd-minikube                      1/1     Running   0             37m
kube-apiserver-minikube            1/1     Running   0             37m
kube-controller-manager-minikube   1/1     Running   0             37m
kube-proxy-vt8lt                   1/1     Running   0             37m
kube-scheduler-minikube            1/1     Running   0             37m
storage-provisioner                1/1     Running   1 (36m ago)   37m
PS C:\Users\user>
```

Для того что бы подключиться внутрь НОДЫ, надо посмотреть ip адресы НОДЫ (Внутри докер контейнера невозможно)

```
minikube ip
```

Зато есть команда

```
minikube ssh
```

с её помощью мы попадем внутрь НОДЫ, которая работает внутри контейнера Docker

Внутри мы можем посмотреть все контейнеры Docker запущены в данный момент

```
docker ps
```

все эти контейнеры являются служебными по умолчанию, пока мы не создали новых

# СОЗДАНИЕ ПОДОВ

## Создать ПОД

```
kubectl run "название" --image="название образа"
```

## Удалить под:

```
kubectl delete pod "название пода"
```

> Что бы создать под уже из собранного изображения локально, надо загрузить изображение в minikube

```
minikube image load "название images"
```

Посмотреть образы в миникубе:

```
minikube image ls
```

Далее

```
kubectl run "название pod" --image="название images" --image-pull-policy=Never
```

- Параметр `--image-pull-policy=Never` указывает Kubernetes не пытаться загружать образ из внешнего реестра (например, Docker Hub).

Детальная информация ПОДА:

```
kubectl describe pod "название pod"
```

# Пример рабочего сценария

Пробрасываем порты

```
kubectl port-forward pod/my-golang-app 8080:8080
```

В другом окне терминала отправьте запрос:

```
curl -v http://localhost:8080
```

и в браузере тоже запускается
Логи:

```
kubectl logs my-golang-app
```

```
PS C:\Users\user\Desktop\учеба сам\go\project> kubectl logs my-gol
2025/03/06 20:31:04 Сервер запущен на :8080
2025/03/06 20:54:55 Получен запрос: POST /calculate
2025/03/06 20:54:55 Данные запроса: {Num1:4 Num2:3 Op:+}
2025/03/06 20:54:55 Отправлен ответ: {Result:7 Error:}
2025/03/06 20:54:58 Получен запрос: POST /calculate
2025/03/06 20:54:58 Данные запроса: {Num1:4 Num2:3 Op:+}
2025/03/06 20:54:58 Отправлен ответ: {Result:7 Error:}
2025/03/06 20:54:59 Получен запрос: POST /calculate
2025/03/06 20:54:59 Данные запроса: {Num1:4 Num2:3 Op:+}
2025/03/06 20:54:59 Отправлен ответ: {Result:7 Error:}
2025/03/06 20:55:00 Получен запрос: POST /calculate
2025/03/06 20:55:00 Данные запроса: {Num1:4 Num2:2 Op:+}
2025/03/06 20:55:00 Отправлен ответ: {Result:6 Error:}
2025/03/06 20:55:06 Получен запрос: POST /calculate
2025/03/06 20:55:06 Данные запроса: {Num1:4 Num2:2 Op:+}
2025/03/06 20:55:06 Отправлен ответ: {Result:6 Error:}
2025/03/06 20:55:07 Получен запрос: POST /calculate
2025/03/06 20:55:07 Данные запроса: {Num1:4 Num2:4 Op:+}
2025/03/06 20:55:07 Отправлен ответ: {Result:8 Error:}
2025/03/06 20:55:08 Получен запрос: POST /calculate
2025/03/06 20:55:08 Данные запроса: {Num1:4 Num2:4 Op:+}
2025/03/06 20:55:08 Отправлен ответ: {Result:8 Error:}
PS C:\Users\user\Desktop\учеба сам\go\project>
```

Как выглядит контейнер который запущен на узле (определенный контейнер):

```
docker ps | grep "my-golang-app" - название контейнера
```

выполняется данная команда когда мы внутри ноды:

```
minikube ssh
```

*! выше описана эта команда !*

```
docker@minikube:~$ docker ps | grep my-golang-app
3e80a8e4518a   5e9b73de4257                  "./myapp"                13 minutes ago
Up 13 minutes             k8s_my-golang-app_my-golang-app_default_9440882f-b007-4e88-9
58a-4c05d8ed620a_2
dfe701892daa   registry.k8s.io/pause:3.10   "/pause"                 13 minutes ago
Up 13 minutes             k8s_POD_my-golang-app_default_9440882f-b007-4e88-958a-4c05d8e
d620a_2
docker@minikube:~$
```

Получается 2 контейнера (принадлежат одному и тому же поду)

образ для 1го - 5e9b73de4257
образ для 2го - registry.k8s.io/pause:3.10

почему два? хотя поднимали один

Непосредственно веб-сервер работает/отвечает/доступен в контейнере 5e9b73de4257

Второй контейнер registry.k8s.io/pause:3.10 создается для каждого пода который мы создаём. Он создаётся для того что бы зарезервировать пространство ресурсов для определенного пода. Его называют еще "служебный контейнер", так как в поде с первым контейнером может произойти что угодно - удалён/заменён/остановлен, ресурсы для 2го пода останутся **неизменными**

Подключиться внутрь контейнера:

```
docker exec -it "id контейнера" sh
```

```
/app # hostname
my-golang-app
/app # hostname -i
10.244.0.9
/app #
```

IP адрес контейнера такой же как и его ПОДА, IP адрес это один из ресурсов ПОДА (т.е второй контейнер registry.k8s.io/pause:3.10 резервирует IP)

# Deployment

## создание:

```
kubectl create deployment "название деплоймента" --image="название образа"
```

У всех ПОДОВ в deployment будут одинаковые образы.

## удаление :

```
kubectl delete deployment "название деплоймента"
```

Тут опять возникает необходимость в команде `imagePullPolicy: Never` при создании деплоймента, но тут, уже, нельзя так просто вписать её, как при создании обычного одного пода.

> Сразу натыкаемся на необходимость в создании и конфигурировании YAML-манифеста

Посмотреть существующие deploy'и:

```
kubectl get deployments
```

Посмотреть информацию о деплойментах:

```
kubectl describe deploy "Название деплоймента"
```

Изменить количество подов в деплойменте:

```
k scale deploy "название деплоймента" --replicas="кол-во(int)"
```

Посмотрим поды с ip:

```
kubectl get pods -o wide
```

```
NAME                                 READY   STATUS    RESTARTS      AGE   IP            NODE
my-golang-deploy-6bfbcf9584-72xmj    1/1     Running   0             18m   10.244.0.21   minikube
my-golang-deploy-6bfbcf9584-78rvl    1/1     Running   1 (42h ago)   43h   10.244.0.19   minikube
my-golang-deploy-6bfbcf9584-m6jvl    1/1     Running   1 (42h ago)   43h   10.244.0.18   minikube
my-golang-deploy-6bfbcf9584-q6pjj    1/1     Running   0             18m   10.244.0.20   minikube
my-golang-deploy-6bfbcf9584-t28wv    1/1     Running   1 (42h ago)   43h   10.244.0.16   minikube
```

У каждого пода свой IP адрес.
IP адрес начинается с 10. это значит для этих подов внутри кластера выделяются приватные IP адреса, а они недоступны для маршрутизации в интернете.

> Что бы подключаться к какому либо поду снаружи или изнутри Кластера Kubernetes используются Service (сервисы), это дополнение к деплойментам. Благодаря сервисам можно используя один единственный IP адрес подключаться к любому из подов в рамках определенного деплоймента

## Манифест YAML

Я создал в корневой папке проекта файл `deployment.yaml`

```
/project
  ├── app
  │     └── main.go
  │     └── go.mod
  ├── public
  │     └── index.html
  ├── Dockerfile
  └── deployment.yaml  # <-- Здесь создадим манифест
```

Настроил его вот так:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-golang-deploy  # Имя Deployment
spec:
  replicas: 3  # Количество реплик (подов)
  selector:
    matchLabels:
      app: my-golang-app  # Лейбл для выбора подов
  template:
    metadata:
      labels:
        app: my-golang-app  # Лейбл для подов
    spec:
      containers:
      - name: my-golang-app  # Имя контейнера
        image: my-golang-app  # Имя образа
        imagePullPolicy: Never  # Используем локальный образ
        ports:
        - containerPort: 8080  # Порт контейнера
```

Кстати, лейблы это метаданные и они представляют собой пары ключ-значение. Лейблы помогают группировать объекты по определённым критериям. Например:

- `app: my-golang-app` — для идентификации всех ресурсов, связанных с приложением.
- `environment: production` — для разделения ресурсов по окружениям (production, staging, development).
  Лейблы используются для выбора объектов. Например:
- **Deployment** использует лейблы для выбора подов, которыми он управляет.

- **Service** использует лейблы для выбора подов, на которые он направляет трафик. Лейблы позволяют фильтровать объекты при выполнении команд `kubectl`

```
kubectl get pods -l app=my-golang-app
```

~~Лирическое отступление закончилось~~, далее:

> заходим в папку и пишем команду что бы применить манифест:

```
kubectl apply -f deployment.yaml
```

можно проверить статус:

```
kubectl get deployments
kubectl get pods
```

пример с пробросом портов:

```
PS C:\Users\user\Desktop\учеба сам\go\project> k get deploy
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
my-golang-deploy   3/3     3            3           19s
PS C:\Users\user\Desktop\учеба сам\go\project> k get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-golang-deploy-6bfbcf9584-78rvl   1/1     Running   0          28s
my-golang-deploy-6bfbcf9584-m6jvl   1/1     Running   0          28s
my-golang-deploy-6bfbcf9584-t28wv   1/1     Running   0          28s
PS C:\Users\user\Desktop\учеба сам\go\project> kubectl port-forward my-golang-deploy-6bfbcf9584-m6jvl 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
Handling connection for 8080
```

**Преимущества YAML-манифестов**

- **Гибкость**: можно настроить любые параметры (например, ресурсы, переменные окружения, volumes).
- **Повторяемость**: Манифесты можно хранить в репозитории и использовать для развёртывания в любом кластере.
- **Версионность**: можно отслеживать изменения в манифестах с помощью Git.

# Service

**Service** в Kubernetes — это абстракция, которая обеспечивает доступ к приложениям, работающим в кластере. Он решает несколько важных задач, таких как балансировка нагрузки, стабильный доступ к подам и управление сетевым трафиком.
Показать доступные сервисы:

```
kubectl get services
```

Детальная информация о сервисе:

```
kubectl describe service "название сервиса"
```

Удалить сервис:

```
kubectl delete svc "название сервиса" //"svc - короткий вариант service"
```

## ClusterIP

Служебный сервис, используется для того что бы направить трафик внутри кластера к API сервису на мастер узле (снаружи к его IP адресу нет)

Получить описание:

```
kubectl describe service kubernetes
```

### Создание сервиса ClusterIP для деплоймента:

```
kubectl expose deploy "название деплоймента" --port="номер порта"
```

Если внутренний порт в ПОДАХ отличается от заданного нами порта то необходимо добавить опцию `--target-port=80`.
В таком случае поды станут доступны через порт который мы указали

```
PS C:\Users\user> k expose deploy my-golang-deploy  --port=8080
service/my-golang-deploy exposed
PS C:\Users\user> k get svc
NAME              TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes        ClusterIP   10.96.0.1      <none>        443/TCP    3d2h
my-golang-deploy  ClusterIP   10.102.92.93   <none>        8080/TCP   15s
PS C:\Users\user>
```

```
PS C:\Users\user> k describe service my-golang-deploy
Name:                     my-golang-deploy
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=my-golang-app
Type:                     ClusterIP
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.102.92.93
IPs:                      10.102.92.93
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
Endpoints:                10.244.0.20:8080,10.244.0.18:8080,10.244.0.16:8080 + 2 more...
Session Affinity:         None
Internal Traffic Policy:  Cluster
Events:                   <none>
PS C:\Users\user>
```

> Как видим тут есть 5 доступных эндпоинтов и при обращении на IP 10.192.92.93, идет перенаправление запросов на один из эндпоинтов, т.е выполняется балансировка нагрузки между ними

Опять же 10.192.92.93 этот IP доступен только внутри кластера!

Получается что такой тип сервисов подходит тогда когда у нас например есть деплоймент в рамках кластера Kubernetes с которым должны взаимодействовать только другие деплойменты и который не должен быть доступен снаружи.

## Создание сервиса типа NodePort:

### Создание сервиса

```
kubectl expose deploy "название деплоймента" --type=NodePort --port="номер порта" --target-port="номер порта"
```

```
PS C:\Users\user> k get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP          3d2h
my-golang-deploy  NodePort    10.106.237.31   <none>        8080:31380/TCP   75s
PS C:\Users\user>
```

Видно что появился NodePort сервис у которого есть пара портов
где 31380 был открыт на узле для этого сервиса. Т.е открывается данный порт на каждой ноде. А порт 8080 это порт сервиса внутри кластера кубернетес.
Благодаря NodePort мы можем подключиться снаружи к деплойменту используя IP адрес узла и порт 31380

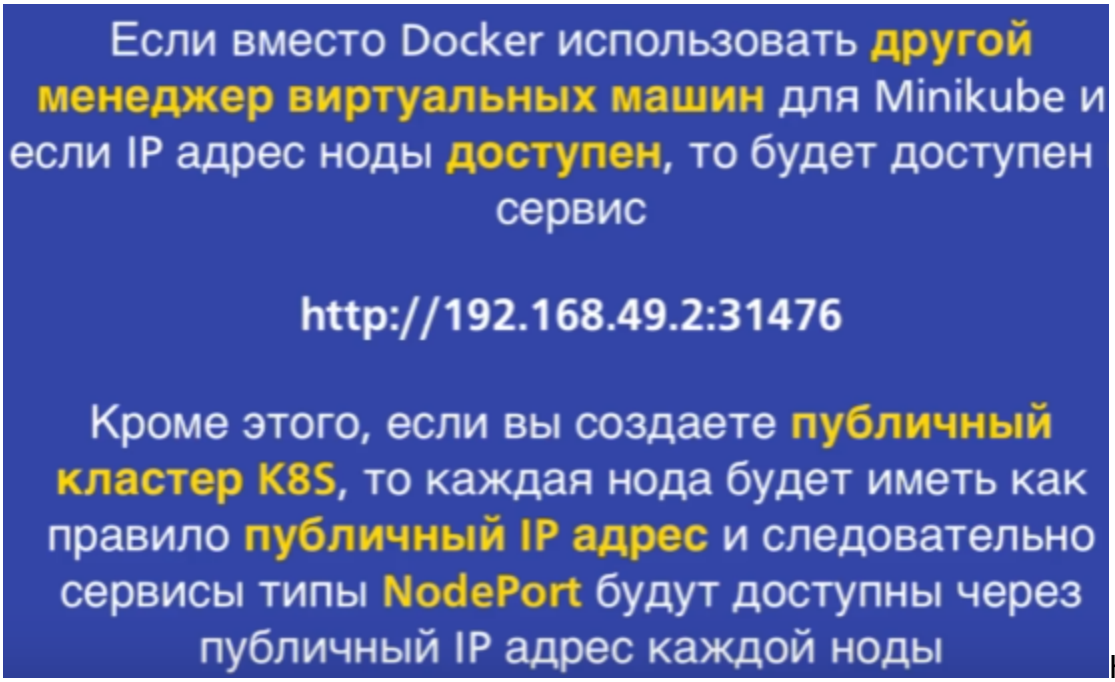
смотрим инфу о сервисе

```
k describe service "название сервиса"
```

В итоге можем взять IP адрес НОДЫ

```
minikube ip
```

добавить к нему NodePort "31380" и подключиться снаружи кластера, но так как я использую Docker (Кластер кубернетес внутри контейнера докер) и IP адрес ноды в таком случае недоступен из вне.



Но можно создать тоннель:

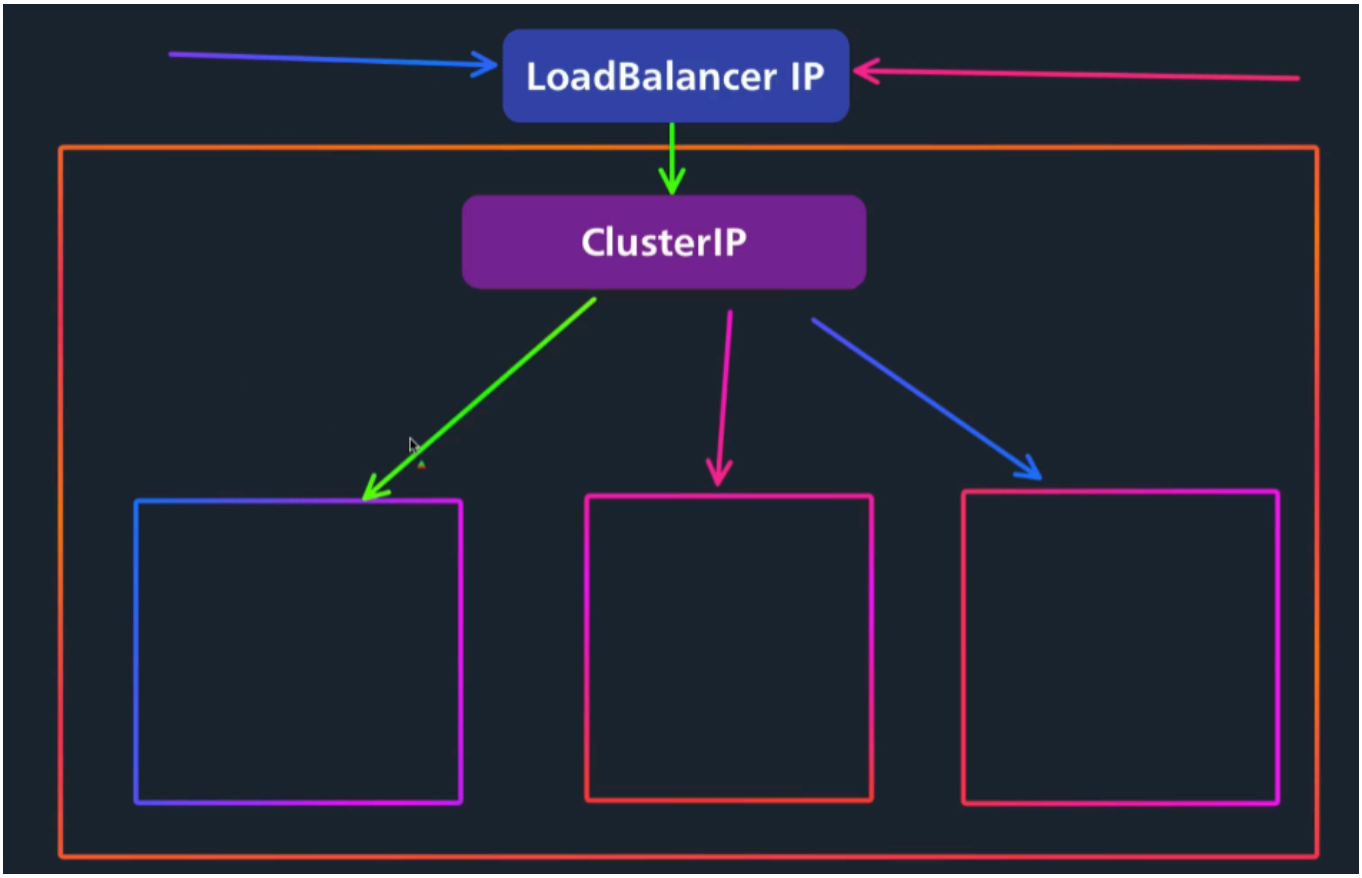
```
minikube service "название сервиса" --url
```

Тем самым можно подключиться к подам

## Создание сервиса типа LoadBalancer:

```
Когда мы создаём сервис типа NodePort на каждом узле открывается определенный порт (один и тот же порт в рамках одного и того же сервиса) и используя IP адрес любого из узлов можно подключиться к определенному сервису снаружи. Для этого необходимо знать IP адрес определенного узла через который мы хотим подключиться, потому это бывает не совсем удобно.
В таком случае можно использовать еще один тип сервиса LoadBalancer
```

В случае с LoadBalancer создаётся отдельный IP адрес (называется External IP) он связывается с IP адресом сервиса внутри кластера кубернетес и через этот один внешний IP можно подключаться к сервису и следовательно к одному из узлов в деплойменте




Для начала введём команду:

```
minikube tunnel
```

С помощью этой команды создаётся тоннель благодаря которому мы сможем подключаться через внешний IP определенного сервиса внутрь кластера кубернетес

```
PS C:\Users\user> minikube tunnel
✅  Tunnel successfully started

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

Создание сервиса:

```
kubectl expose deploy "название деплоймента" --type=LoadBalanccer --
port="номер внешнего порта" --target-port="номер целевого порта"
```

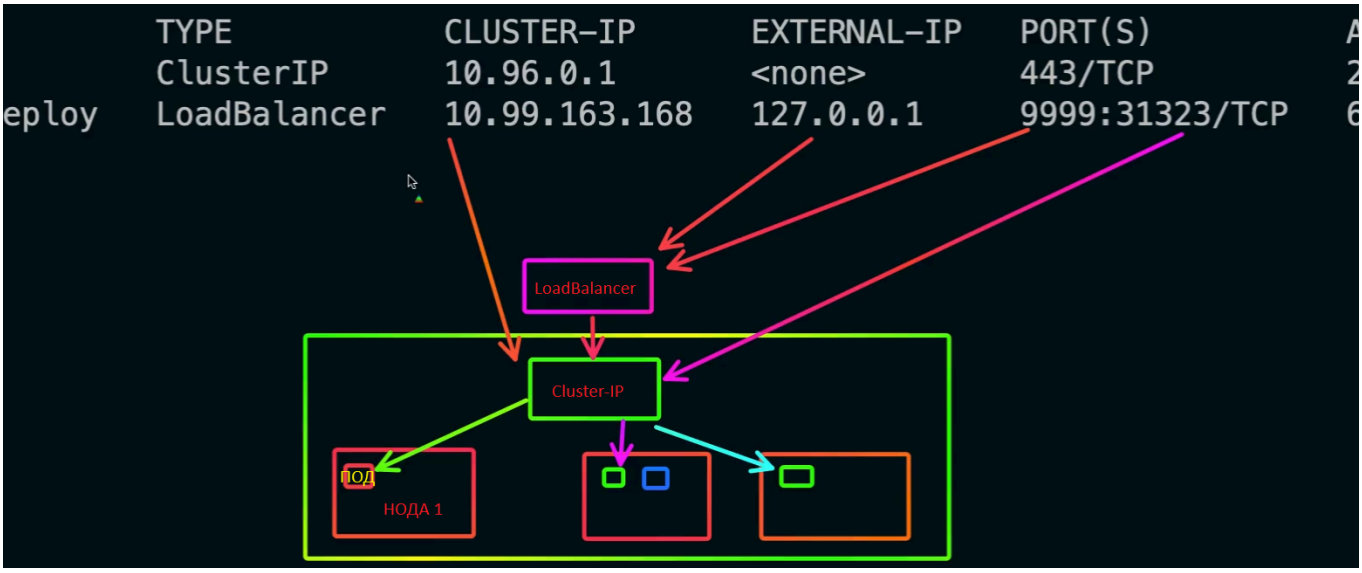
```
PS C:\Users\user> k expose deploy my-golang-deploy --type=LoadBalancer --port=9999 --target-port
=8080
service/my-golang-deploy exposed
PS C:\Users\user> k get svc
NAME              TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes        ClusterIP      10.96.0.1        <none>        443/TCP          3d21h
my-golang-deploy  LoadBalancer   10.110.241.227   127.0.0.1     9999:30445/TCP   4s
PS C:\Users\user>
```

```
PS C:\Users\user> minikube tunnel
✅  Tunnel successfully started

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

🏃  Starting tunnel for service my-golang-deploy.
```

В другом окне командной строки, появилось сообщение что был создан тоннель для сервиса
Теперь используя внешний IP (так как это не продакшн и я по естественным причинам не хочу покупать сейчас внешний IP, миникуб тунель выделил мне как вешний IP просто по сути локалхост) и порт 9999 можем подключиться внутрь кластера и попасть условно на "сайт" приложения http://127.0.0.1:9999/




В реальных проектах чаще всего используют тип LoadBalancer (используют один внешний IP для доступа к целому деплойменту, независимо от ого сколько в деплойменте подов)

# Контроллер и minikube

1. **Прямое подключение через IP** (если контроллер доступен из сети кластера, что маловероятно).
2. **Использование Service с типом ExternalName**.
3. **Использование VPN или сетевого моста** (если контроллер находится в другой сети).

"**ExternalName** — это тип Service, который создает DNS-запись внутри кластера Kubernetes, указывающую на внешний ресурс (например, IP-адрес или доменное имя).

Проверить подключение к контроллеру внутр кластера:

```
minikube ssh
ping 192.168.0.1 "- какой то ip адрес"
```

## Пример манифеста Service:

```
apiVersion: v1
kind: Service
metadata:
  name: s7-controller
spec:
  type: ExternalName
  externalName: 192.168.0.1
```

- Kubernetes должен создать DNS-запись `s7-controller`, которая будет указывать на IP-адрес контроллера ( `192.168.0.1` ).

тогда в Go можно использовать вместо IP, dns адрес

```
handler := gos7.NewTCPClientHandler("s7-controller", 0, 1)
```

### Как я вижу эту систему:

```
**ExternalName Service**:
```

- Используется для подключения к внешнему ресурсу (например, контроллеру Siemens S7) изнутри кластера.
- Он **не привязан к Pod'ам** через селекторы, а просто создает DNS-запись, указывающую на внешний IP-адрес
  **NodePort/LoadBalancer Service**:
- Используется для предоставления доступа к вашему приложению (например, веб-интерфейсу) извне.
- Он **привязан к Pod'ам** через селекторы.
  Таким образом, можно создать:
- Один **ExternalName Service** для подключения к контроллеру.

- Один **NodePort/LoadBalancer Service** для доступа к веб-интерфейсу вашего приложения.