10. Работа с Service

10.1. Создание Service через команду

1. Создадим деплоймент с тремя репликами:

```
kubectl create deployment svc-deploy --image=k8sphp  # Создает Deployment с указанным образом контейнера
kubectl scale deployment svc-deploy --replicas=3  # Масштабируем до 3 реплик
kubectl get pods  # Проверяем созданные поды
```

10.2. Создание Service типа ClusterIP

1. Создадим сервис, который будет балансировать трафик между подами:

```
kubectl expose deployment svc-deploy --type=ClusterIP --port=80  # Создает ClusterIP сервис на порту 80
kubectl get services  # Проверяем созданный сервис
```

2. Проверим работу балансировки через несколько запросов:

```
curl <IP_сервиса>  # Должны попадать на разные поды при каждом вызове
```

3. Удалим сервис:

```
kubectl delete service svc-deploy  # Удаляем созданный сервис
```

10.3. Создание Service типа NodePort

1. Создадим сервис с доступом через внешний порт узла:

```
kubectl expose deployment svc-deploy --type=NodePort --port=80

# Получаем внешний IP-адрес узла для тестирования
kubectl get nodes -o wide  # В столбце EXTERNAL-IP будет указан внешний IP узла

# Используем этот IP-адрес с портом из команды 'kubectl get service' для доступа через браузер
kubectl get service  # Проверяем полученный порт
```

2. Проверим доступ через браузер:

```
http://<IP_узла>:<PORT>
```

3. Удалим сервис:

```
kubectl delete service svc-deploy
```

10.4. Создание Service типа LoadBalancer (Только для облачных провайдеров)

1. Создаем сервис с типом LoadBalancer:

```
kubectl expose deployment svc-deploy --type=LoadBalancer --port=80
```

```
kubectl get service  # Проверяем внешний IP балансировщика
```

2. Удаляем сервис:

```
kubectl delete service svc-deploy
```

10.5. Создание Service через манифест

Вариант 1: ClusterIP для одного контейнера

1. Создаем файл `service-1-clusterip-single.yaml`:

```
apiVersion: apps/v1  # Версия API для Deployment
kind: Deployment
metadata:
  name: my-web-deploy  # Имя Deployment
  labels:
    app: my-k8s-deploy
spec:
  replicas: 3  # Количество реплик
  selector:
    matchLabels:
      project: SuperApp
  template:
    metadata:
      labels:
        project: SuperApp  # Метка, по которой Service будет находить поды
    spec:
      containers:
      - name: SuperApp-web
        image: k8sphp:latest
        ports:
        - containerPort: 80  # Порт, на котором контейнер принимает запросы

# Для проверки доступности сервиса можно использовать команду curl:
# curl <IP_сервиса>:80
# Или открыть браузер по адресу http://<IP_сервиса>:80
# В случае успешного ответа приложение вернет страницу или сообщение о работе
сервиса.
---
apiVersion: v1  # Версия API для Service
kind: Service
metadata:
  name: my-single-pod-service  # Имя сервиса
  labels:
    env: prod
    owner: IvanKlimarev
spec:
  selector:
    project: SuperApp  # Выбираем поды по метке
  ports:
    - name: app-listener
      protocol: TCP
      port: 80  # Порт на сервисе
      targetPort: 80  # Порт на поде
  type: ClusterIP
```

2. Применяем манифест:

```
kubectl apply -f service-1-clusterip-single.yaml
kubectl get svc  # Проверяем созданный сервис
```

Вариант 2: ClusterIP для многоконтейнерного пода

1. Создаем файл `service-2-clusterip-multi.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web-deploy-multi-pods
  labels:
    app: my-k8s-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      project: MegaApp
  template:
    metadata:
      labels:
        project: MegaApp
    spec:
      containers:
      - name: my-web
        image: k8sphp:latest
        ports:
        - containerPort: 80
      - name: not-my-web
        image: tomcat
        ports:
        - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: my-multi-pods-service
  labels:
    env: prod
    owner: IvanKlimarev
spec:
  selector:
    project: MegaApp
  ports:
    - name: my-web-app-listener
      protocol: TCP
      port: 80
      targetPort: 80
    - name: not-my-web-app-listener
      protocol: TCP
      port: 8888
      targetPort: 8080
  type: ClusterIP

# Чтобы протестировать оба контейнера, выполните следующие команды:
# 1. Для контейнера с приложением на порту 80:
# curl <IP_сервиса>:80
# 2. Для контейнера с приложением на порту 8080:
# curl <IP_сервиса>:8888
# Ожидается разный ответ от каждого контейнера, демонстрирующий их
независимую работу.
```

2. Применяем манифест:

```
kubectl apply -f service-2-clusterip-multi.yaml
```

```
kubectl get svc  # Проверяем созданный сервис
```

10.6. Удаление ресурсов

1. Удаляем все созданные ресурсы:

```
kubectl delete -f service-1-clusterip-single.yaml
kubectl delete -f service-2-clusterip-multi.yaml
kubectl delete svc --all  # Удаляем все сервисы
```