

Департамент образования города Москвы  
Государственное автономное образовательное учреждение  
высшего образования города Москвы  
«Московский городской педагогический университет»

Институт цифрового образования  
Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

«Интеграция и развертывание программного обеспечения с помощью  
контейнеров»

Лабораторная работа №3.2

Тема: «Развертывание приложения в Kubernetes»

Выполнила:

Студентка группы АДЭУ-211

Кравцова Алёна Евгеньевна

Руководитель:

Босенко Т.М

Москва

2025

Цель работы: освоить процесс развертывания приложения в Kubernetes с использованием Deployments и Services.

Задачи:

- Создать Deployment для указанного приложения;
- Создать Service для обеспечения доступа к приложению;
- Проверить доступность приложения через созданный Service;
- Выполнить индивидуальное задание.

Вариант 6. Разверните приложение на Spring Boot, использующее базу данных PostgreSQL, в Kubernetes. Создайте Deployment для Spring Boot и PostgreSQL, а также Service для доступа к приложению.

Ход выполнения:

Шаг 1. Установка Kubernetes и инструментов

Minikube — это локальный Kubernetes-кластер для разработки, который необходимо установить (Рис. 1).

```
kravtsovaee@alenaubuntu:~$ cd projects/employee-managment
kravtsovaee@alenaubuntu:~/projects/employee-managment$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 119M 100 119M 0 0 808k 0 0:02:31 0:02:31 --:--:-- 9295k
kravtsovaee@alenaubuntu:~/projects/employee-managment$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
[sudo] password for kravtsovaee:
kravtsovaee@alenaubuntu:~/projects/employee-managment$
```

Рис. 1 – Установка Minikube

Проверим установку (Рис. 2).

```
kravtsovaee@alenaubuntu:~/projects/employee-managment$
kravtsovaee@alenaubuntu:~/projects/employee-managment$ minikube version
minikube version: v1.35.0
commit: dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty
kravtsovaee@alenaubuntu:~/projects/employee-managment$
kravtsovaee@alenaubuntu:~/projects/employee-managment$
```

Рис. 2 – Проверка Minikube

Далее необходимо установить kubectl, инструмент с помощью которого будет производиться управление кластером (Рис. 3).

```
kravtsovaee@alenaubuntu:~$ curl -LO "https://dl.k8s.io/release/$(curl -sL https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
  0     0    0     0    0     0      0      0  --:--:-- --:--:-- --:--:--    0
100 138 100 138    0     0   653    0  --:--:-- --:--:-- --:--:--   657
100 54.6M 100 54.6M    0     0  8197k    0  0:00:06 0:00:06 --:--:--  8686k
kravtsovaee@alenaubuntu:~$
kravtsovaee@alenaubuntu:~$ chmod +x kubectl
kravtsovaee@alenaubuntu:~$ sudo mv kubectl /usr/local/bin/
kravtsovaee@alenaubuntu:~$ kubectl version --client
Client Version: v1.32.3
Kustomize Version: v5.5.0
```

Рис. 3 – Установка kubectl

Запустим minikube (Рис. 4) и проверим статус работоспособности (Рис. 5).

```
kravtsovaee@alenaubuntu:~$ minikube start
🐹 minikube v1.35.0 on Ubuntu 24.04 (vbox/amd64)
🌟 Automatically selected the docker driver. Other choices: none, ssh
👉 Using Docker driver with root privileges
🔧 Starting "minikube" primary control-plane node in "minikube" cluster
📦 Pulling base image v0.0.46 ...
📦 Downloading Kubernetes v1.32.0 preload ...
> preloaded-images-k8s-v18-v1...: 333.57 MiB / 333.57 MiB 100.00% 6.11 Mi
> gcr.io/k8s-minikube/kicbase...: 500.31 MiB / 500.31 MiB 100.00% 6.16 Mi
🔥 Creating docker container (CPUs=2, Memory=2200MB) ...
📦 Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
  ▪ Generating certificates and keys ...
  ▪ Booting up control plane ...
  ▪ Configuring RBAC rules ...
🔗 Configuring bridge CNI (Container Networking Interface) ...
🔍 Verifying Kubernetes components...
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟 Enabled addons: storage-provisioner, default-storageclass
🏁 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
kravtsovaee@alenaubuntu:~$
```

Рис. 4 – Запуск minikube

```
kravtsovaee@alenaubuntu:~$
kravtsovaee@alenaubuntu:~$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Рис. 5 – Проверка работоспособности

## Шаг 2. Создание docker образа

Переключаемся на локальный Docker Minikube, то есть перенаправляем Docker на использование встроенного реестра Minikube вместо стандартного локального Docker-демона (Рис. 6).

```
kravtsovaee@alenaubuntu:~/projects/employee-managment$
kravtsovaee@alenaubuntu:~/projects/employee-managment$
kravtsovaee@alenaubuntu:~/projects/employee-managment$ eval $(minikube docker-env)
kravtsovaee@alenaubuntu:~/projects/employee-managment$ ls
build.gradle  docker-compose.yml  Dockerfile  minikube-linux-amd64  pom.xml  prometheus.yml  settings.gradle  src
kravtsovaee@alenaubuntu:~/projects/employee-managment$
kravtsovaee@alenaubuntu:~/projects/employee-managment$
```

Рис. 6 – Переключение на Minikube

Создаём Dockerfile для Spring Boot в корне проекта (Рис. 7).

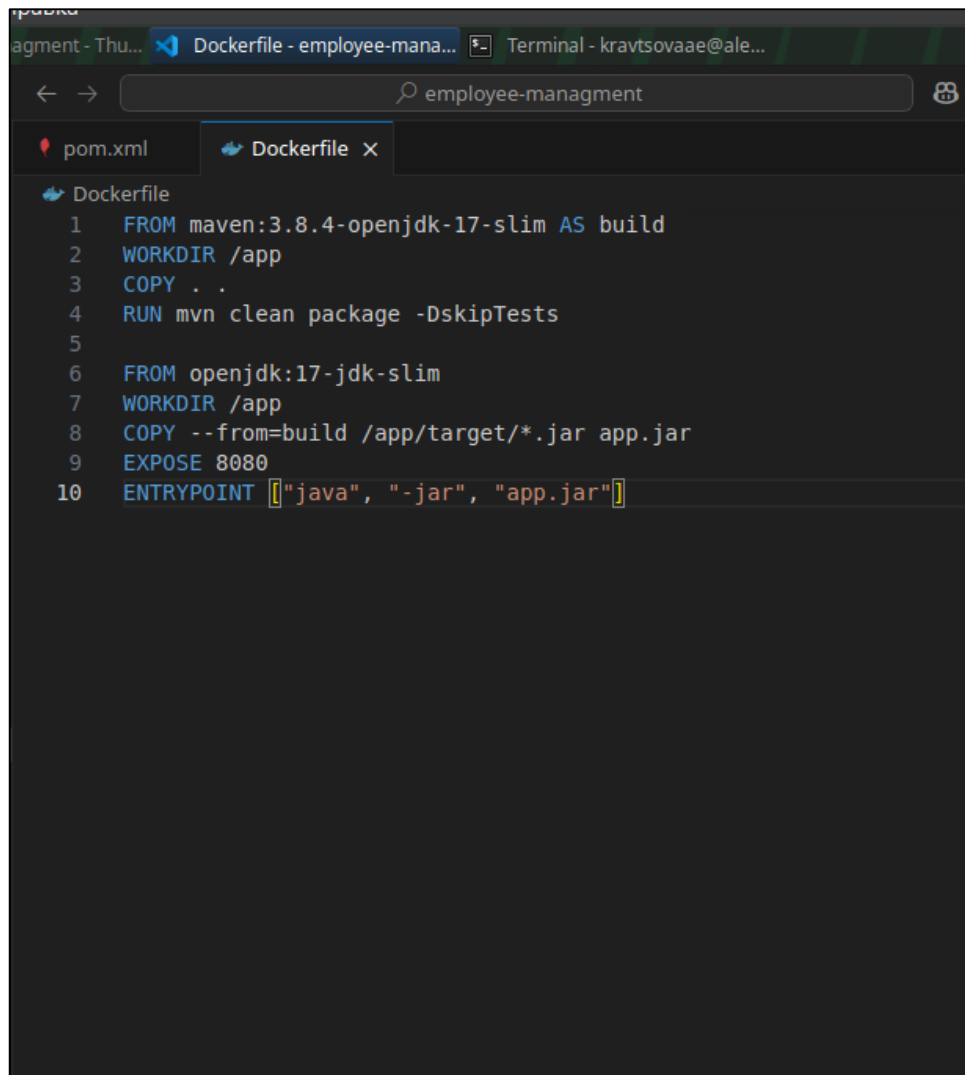


Рис. 7 – Dockerfile для приложения

Собираем Docker-образ с помощью команды *docker build -t spring-app:latest* (Рис. 8).

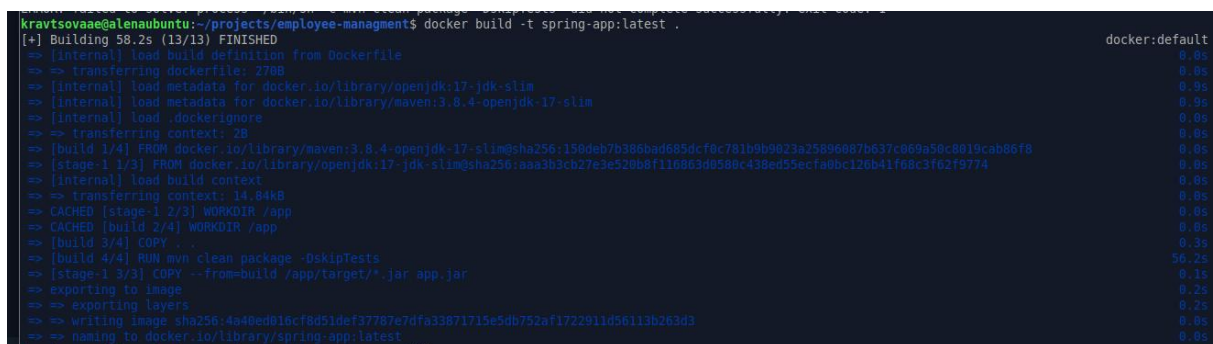


Рис. 8 – Сборка docker-образа

Проверяем, что образ создан (Рис. 9).

```
=> => naming to docker.io/library/spring-app:latest
kravtsova@alenaubuntu:~/projects/employee-managment$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
spring-app           latest             4a40ed016cf8       About a minute ago 458MB
registry.k8s.io/kube-apiserver    v1.32.0           c2e17080014a       3 months ago       97MB
registry.k8s.io/kube-scheduler    v1.32.0           a389e107f4ff       3 months ago       69.6MB
registry.k8s.io/kube-controller-manager v1.32.0         8cab3d2a8bd0       3 months ago       89.7MB
registry.k8s.io/kube-proxy        v1.32.0           040f9f8aac8c       3 months ago       94MB
registry.k8s.io/etcd              3.5.16-0          a9e7e6b294ba       6 months ago       150MB
registry.k8s.io/coredns/coredns   v1.11.3           c69fa2e9cbf5       7 months ago       61.8MB
registry.k8s.io/pause             3.10              873ed7510279       10 months ago      736kB
gcr.io/k8s-minikube/storage-provisioner v5                6e38f40d628d       3 years ago        31.5MB
kravtsova@alenaubuntu:~/projects/employee-managment$
```

Рис. 9 – Проверка образа

### Шаг 3: Настройка манифестов Kubernetes

Создадим деплойменты и сервисы для PostgreSQL, Spring Boot, Prometheus и Grafana.

PostgreSQL Deployment. Создадим файл postgres-deployment.yml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:15
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_DB
              value: "employees_db"
            - name: POSTGRES_USER
              value: "admin"
            - name: POSTGRES_PASSWORD
              value: "admin"
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgres-storage
      volumes:
        - name: postgres-storage
          emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: postgres
```

```
spec:
  selector:
    app: postgres
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
  type: ClusterIP
```

## Spring Boot Deployment. Создадим spring-app-deployment.yml.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: spring-app
  template:
    metadata:
      labels:
        app: spring-app
    spec:
      containers:
        - name: spring-app
          image: spring-app:latest
          imagePullPolicy: Never
          ports:
            - containerPort: 8080
          env:
            - name: SPRING_DATASOURCE_URL
              value: "jdbc:postgresql://postgres:5432/employees_db"
            - name: SPRING_DATASOURCE_USERNAME
              value: "admin"
            - name: SPRING_DATASOURCE_PASSWORD
              value: "admin"
---
apiVersion: v1
kind: Service
metadata:
  name: spring-app
spec:
  selector:
    app: spring-app
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: NodePort
```

## Prometheus Deployment. Создадим prometheus-config.yml.

```
global:
  scrape_interval: 15s
```

```

scrape_configs:
  - job_name: 'spring-boot-app'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['spring-app:8080']

```

Далее создадим prometheus-deployment.yml:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus
          ports:
            - containerPort: 9090
          volumeMounts:
            - name: config-volume
              mountPath: /etc/prometheus
      volumes:
        - name: config-volume
          configMap:
            name: prometheus-config
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: 'spring-boot-app'
        metrics_path: '/actuator/prometheus'
        static_configs:
          - targets: ['spring-app:8080']
---
apiVersion: v1
kind: Service
metadata:
  name: prometheus
spec:
  selector:

```

```

    app: prometheus
ports:
  - protocol: TCP
    port: 9090
    targetPort: 9090
type: NodePort

```

**Grafana Deployment.** Создадим grafana-deployment.yml:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
        - name: grafana
          image: grafana/grafana
          ports:
            - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: grafana
spec:
  selector:
    app: grafana
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  type: NodePort

```

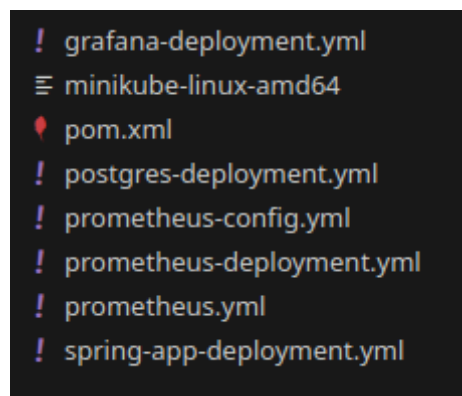


Рис. 10 – Созданные манифесты

**Шаг 4.** Разворачивание в Kubernetes.



Применяем манифесты (Рис. 11).

```
kravtsova@alenaubuntu: ~/projects/employee-managment$ kubectl apply -f postgres-deployment.yml
deployment.apps/postgres created
service/postgres created
kravtsova@alenaubuntu: ~/projects/employee-managment$ kubectl apply -f spring-app-deployment.yml
deployment.apps/spring-app created
service/spring-app created
kravtsova@alenaubuntu: ~/projects/employee-managment$ kubectl apply -f prometheus-deployment.yml
deployment.apps/prometheus created
configmap/prometheus-config created
service/prometheus created
kravtsova@alenaubuntu: ~/projects/employee-managment$ kubectl apply -f grafana-deployment.yml
deployment.apps/grafana created
service/grafana created
```

Рис. 11 – Применение манифестов

Проверим статус запуска (Рис. 12).

```
kravtsova@alenaubuntu: ~/projects/employee-managment$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
grafana-85b785d45d-xnphw            1/1     Running   0           48s
postgres-7ddd46465c-cpjs8           1/1     Running   0           78s
prometheus-55c94d4c4d-7b7jf         1/1     Running   0           56s
spring-app-6b7b4845d8-m5l9w         1/1     Running   2 (52s ago) 63s
kravtsova@alenaubuntu: ~/projects/employee-managment$ kubectl get svc
NAME              TYPE          CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
grafana           NodePort      10.100.148.198 <none>        3000:30303/TCP   51s
kubernetes        ClusterIP     10.96.0.1      <none>        443/TCP          22m
postgres          ClusterIP     10.98.151.88   <none>        5432/TCP         81s
prometheus        NodePort      10.109.153.189 <none>        9090:30810/TCP   59s
spring-app        NodePort      10.107.129.156 <none>        8080:32052/TCP   66s
```

Рис. 12 – Проверка статуса

## Шаг 5. Проверка сервисов

Открываем Minikube Service с помощью команды `minikube service spring-app` (Рис. 13).

```
kravtsova@alenaubuntu: ~/projects/employee-managment$ minikube service spring-app
-----|-----|-----|-----|
| NAMESPACE | NAME   | TARGET PORT | URL                               |
|-----|-----|-----|-----|
| default   | spring-app | 8080        | http://192.168.49.2:32052       |
|-----|-----|-----|-----|
🔗 Opening service default/spring-app in default browser...
```

Рис. 13 – Запуск сервиса

Minikube откроет браузер с приложением Spring Boot (Рис. 14).

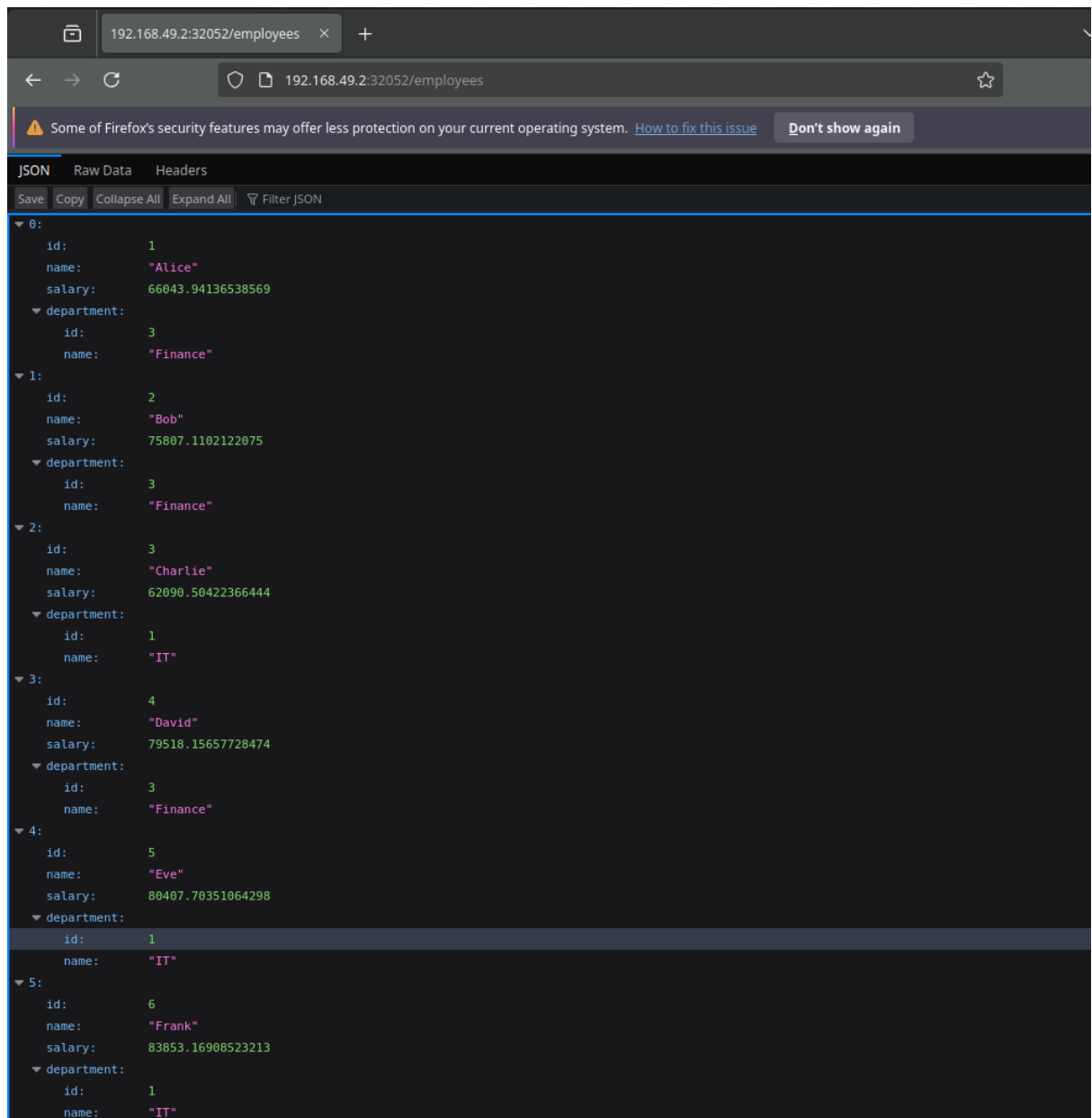


Рис. 14 – Запущенное приложение по учету сотрудников

Контрольные вопросы.

1. Что такое Pod, Deployment и Service в Kubernetes?

Pod — это наименьшая разворачиваемая единица в Kubernetes. В отличие от Docker, где основной единицей является контейнер, в Kubernetes контейнеры запускаются внутри Pod. Pod можно представить, как «обертку» вокруг одного или нескольких контейнеров, тесно связанных между собой и совместно использующих ресурсы.

Deployment — это объект Kubernetes, который обеспечивает декларативное управление разворачиванием и обновлением Pods.

Service — это абстракция, которая определяет сетевой доступ к набору Pod'ов.

## 2. Каково назначение Deployment в Kubernetes?

Deployment описывает желаемое состояние приложения: сколько реплик Pods должно быть запущено, какую версию контейнера использовать и как обновлять приложение.

## 3. Каково назначение Service в Kubernetes?

Поскольку IP-адреса Pods могут меняться, сервис решает эту проблему, создавая устойчивую точку доступа для взаимодействия с Pod'ами. Он автоматически направляет трафик на актуальные живые Pod'ы, входящие в его группу.

## 4. Как создать Deployment в Kubernetes?

Сперва необходимо написать манифест, который описывает желаемое состояние объекта. Также в типе указать деплоймент (`kind: Deployment`). Далее необходимо его применить командой `kubectl apply -f` и проверить работоспособность.

## 5. Как создать Service в Kubernetes и какие типы Services существуют?

Сперва необходимо написать манифест, который описывает желаемое состояние объекта. Также в типе указать деплоймент (`kind: Service`). Далее необходимо его применить командой `kubectl apply -f` и проверить работоспособность.

Типы сервисов в Kubernetes следующие:

- ClusterIP (Как только запрос поступает на сервис по определённому адресу, сервис знает, что нужно переслать этот запрос одному из модулей, зарегистрированных в качестве конечных точек сервиса. Доступен только внутри кластера);

- Узловой порт (это сервис, доступный на статическом порту на каждом рабочем узле в кластере. Сервис NodePort делает внешний трафик

доступным на статическом или фиксированном порту на каждом рабочем узле, в отличие от сервиса ClusterIP, который был доступен только внутри кластера);

- Балансировщик нагрузки (позволяет сделать сервис доступным извне с помощью функции балансировки нагрузки поставщика облачных услуг);

- Безголовый Сервис (это тип сервиса, который позволяет клиенту напрямую взаимодействовать с модулями. Это полезный инструмент для создания распределённых приложений. Headless-сервисы используются при развёртывании приложений с отслеживанием состояния в Kubernetes).

Заключение: в ходе выполнения работы был получен опыт работы с Kubernetes, а также информация об основных его составляющих и их принципов работы. Задание было успешно выполнено и приложение Spring boot было запущено с применением технологий Kubernetes.