Департамент образования и науки города Москвы
Государственное автономное образовательное учреждение высшего образования города Москвы
«Московский городской педагогический университет»
Институт цифрового образования
Департамент информатики, управления и технологий

Макарова Виктория Сергеевна

ЛАБОРАТОРНАЯ РАБОТА 3.2

Развертывание приложения в Kubernetes

Интеграция и развертывание программного обеспечения с помощью контейнеров

Направление подготовки
38.03.05 Бизнес-информатика

Профиль подготовки
Аналитика данных и эффективное управление

Курс обучения: 4
Форма обучения: очная
Преподаватель: кандидат технических наук,
доцент Босенко Тимур Муртазович

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.

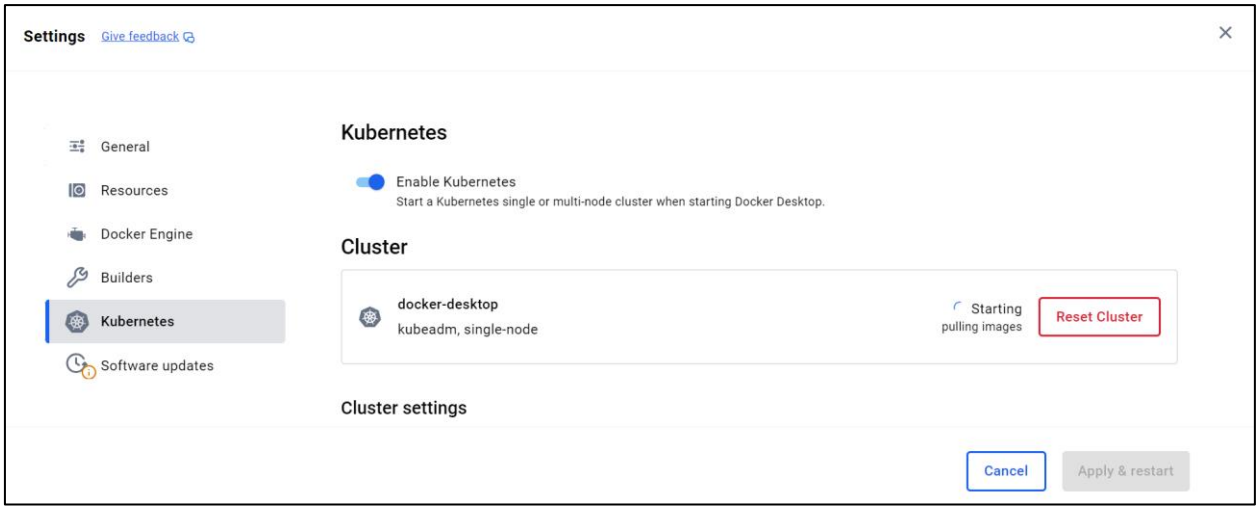
Выполнить индивидуальное задание.

Вариант 7

Разверните приложение на ASP.NET Core, использующее базу данных SQL Server, в Kubernetes. Создайте Deployment для ASP.NET Core и SQL Server, а также Service для доступа к приложению.

Ход работы

Включение K8S в Docker Desktop



Проверка подключения kubectl version k8s

```
makarovavs@DESKTOP-7K05KMD:~$ kubectl version
Client Version: v1.31.4
Kustomize Version: v5.4.2
The connection to the server localhost:8080 was refused - did you specify the right host or port?
```

Проверка статуса работы кластера

```
makarovavs@DESKTOP-7K05KMD:~$ kubectl cluster-info
Kubernetes control plane is running at https://kubernetes.docker.internal:6443
CoreDNS is running at https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

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

Проверка работоспособности нодов

```
makarovavs@DESKTOP-7K05KMD:~$ kubectl get nodes
NAME             STATUS   ROLES           AGE   VERSION
docker-desktop   Ready    control-plane   84s   v1.31.4
```

Создание структуры проекта.

Создаем корневую папку проекта

```
makarovavs@DESKTOP-7K05KMD:~$ mkdir AspNetSqlK8s
makarovavs@DESKTOP-7K05KMD:~$ cd AspNetSqlK8s
```

Создаем новый веб-проект ASP.NET Core

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s$ dotnet new webapp -n MyWebApp -o src
The template "ASP.NET Core Web App (Razor Pages)" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore/8.0-third-party-notices for deta
ils.

Processing post-creation actions...
Restoring /home/makarovavs/AspNetSqlK8s/src/MyWebApp.csproj:
  Determining projects to restore...
  Restored /home/makarovavs/AspNetSqlK8s/src/MyWebApp.csproj (in 238 ms).
Restore succeeded.
```

Добавление поддержки Entity Framework Core

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s$ cd src
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/src$ dotnet add package Microsoft.EntityFrameworkCore.SqlServer
  Determining projects to restore...
  Writing /tmp/tmpTQSEVw.tmp
info : X.509 certificate chain validation will use the fallback certificate bundle at '/snap/dotnet-sdk/256/sdk/8.0.407/trustedroots/
codesignctl.pem'.
info : X.509 certificate chain validation will use the fallback certificate bundle at '/snap/dotnet-sdk/256/sdk/8.0.407/trustedroots/
timestampctl.pem'.
info : Adding PackageReference for package 'Microsoft.EntityFrameworkCore.SqlServer' into project '/home/makarovavs/AspNetSqlK8s/src/
MyWebApp.csproj'.
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.sqlserver/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.sqlserver/index.json 240ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.sqlserver/page/0.0.1-alpha/3.1.2.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.sqlserver/page/0.0.1-alpha/3.1.2.json 302
```

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/src$ dotnet add package Microsoft.EntityFrameworkCore.Design
  Determining projects to restore...
  Writing /tmp/tmpJ6kRLG.tmp
info : X.509 certificate chain validation will use the fallback certificate bundle at '/snap/dotnet-sdk/256/sdk/8.0.407/trustedroots/
codesignctl.pem'.
info : X.509 certificate chain validation will use the fallback certificate bundle at '/snap/dotnet-sdk/256/sdk/8.0.407/trustedroots/
timestampctl.pem'.
info : Adding PackageReference for package 'Microsoft.EntityFrameworkCore.Design' into project '/home/makarovavs/AspNetSqlK8s/src/MyW
ebApp.csproj'.
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.design/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.design/index.json 1633ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.design/page/0.0.1-alpha/3.1.3.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.design/page/0.0.1-alpha/3.1.3.json 272ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.design/page/3.1.4/6.0.0-preview.7.21378.
4.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.design/page/3.1.4/6.0.0-preview.7.21378.4
.json 265ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.design/page/6.0.0-rc.1.21452.10/7.0.18.j
son
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.design/page/6.0.0-rc.1.21452.10/7.0.18.js
on 292ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.design/page/7.0.19/10.0.0-preview.2.2516
3.8.json
```

Настройка подключения к базе данных

Добавляем в appsettings.json: {

"ConnectionStrings": {

"DefaultConnection": "Server=sql-server;Database=MyAppDb;User=sa;Password=YourStrong@Passw0rd;"

}

}

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/src$ ls
MyWebApp.csproj  Pages  Program.cs  Properties  appsettings.Development.json  appsettings.json  obj  wwwroot
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/src$ vim appsettings.json
```

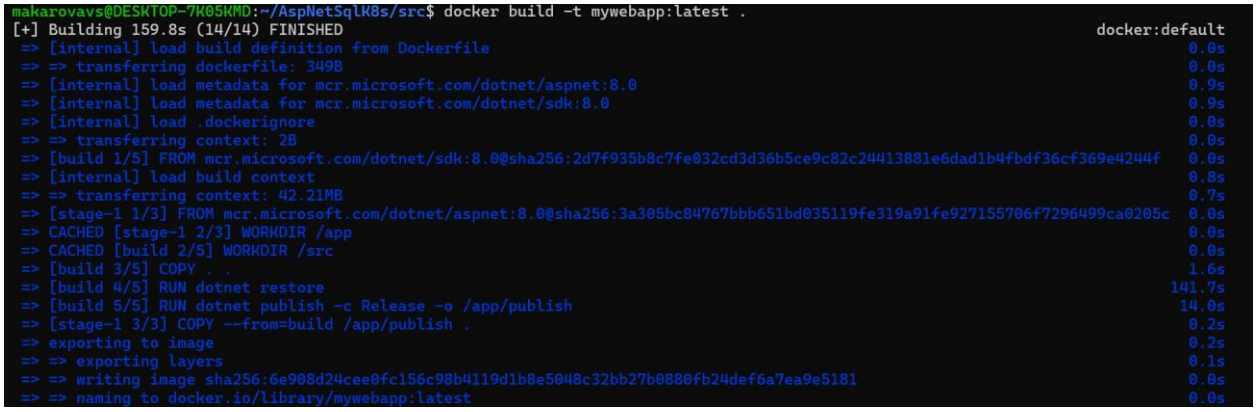
Создание Dockerfile

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/src$ vim Dockerfile
```

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/src$ cat Dockerfile
# Этап сборки
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY . .
RUN dotnet restore
RUN dotnet publish -c Release -o /app/publish

# Этап запуска
FROM mcr.microsoft.com/dotnet/aspnet:8.0
WORKDIR /app
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "MyWebApp.dll"]
```

Сборка образа



Подготовка Kubernetes манифестов

Создаем папку k8s в корне проекта и добавляем файлы:

SQL Server Persistent Volume (sql-pv.yaml)

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s$ mkdir k8s
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s$ cd k8s
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ vim sql-pv.yaml
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ cat sql-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: sql-server-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/mnt/data/sqlserver"
```

SQL Server Persistent Volume Claim (sql-pvc.yaml)

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ vim sql-pvc.yaml
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ cat sql-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: sql-server-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

SQL Server Deployment (sql-deployment.yaml)

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ vim sql-deployment.yaml
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ cat sql-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sql-server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sql-server
  template:
    metadata:
      labels:
        app: sql-server
    spec:
      containers:
      - name: sql-server
        image: mcr.microsoft.com/mssql/server:2019-latest
        ports:
        - containerPort: 1433
        env:
        - name: ACCEPT_EULA
          value: "Y"
        - name: SA_PASSWORD
          value: "YourStrong@Passw0rd"
```

SQL Server Service (sql-service.yaml)

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ vim sql-service.yaml
```

Приложение Deployment (app-deployment.yaml)

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ vim app-deployment.yaml
```

Приложение Service (app-service.yaml)

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ vim app-service.yaml
```

Применение конфигураций

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ kubectl apply -f sql-pv.yaml
persistentvolume/sql-server-pv created
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ kubectl apply -f sql-pvc.yaml
persistentvolumeclaim/sql-server-pvc created
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ kubectl apply -f sql-deployment.yaml
deployment.apps/sql-server created
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ kubectl apply -f sql-service.yaml
service/sql-server created
```

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ kubectl apply -f app-deployment.yaml
deployment.apps/webapp created
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/k8s$ kubectl apply -f app-service.yaml
service/webapp-service created
```

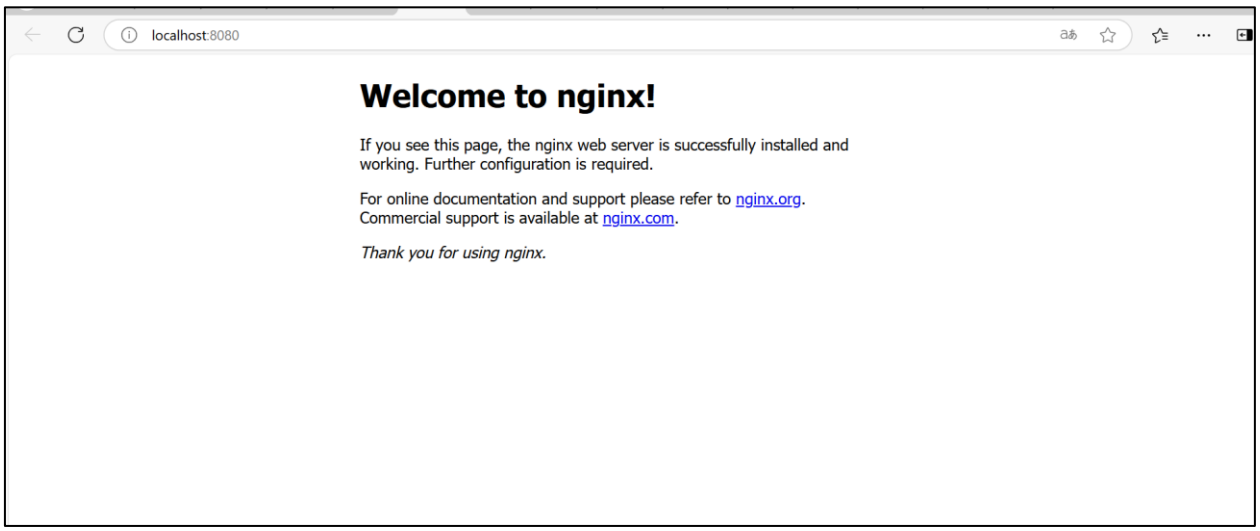
Проверка доступности подов

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/src$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
sql-server-79bf4bb87b-nlgmx   1/1     Running   0          31m
webapp-574975879-5wl9m        1/1     Running   0          114s
webapp-574975879-sg6bb        1/1     Running   0          2m17s
```

Подключение к webapp-service

```
makarovavs@DESKTOP-7K05KMD:~/AspNetSqlK8s/src$ kubectl port-forward svc/webapp-service 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
Handling connection for 8080
Handling connection for 8080
```

Проверка доступности приложения



Вывод: Поставленные задачи были выполнены, а именно:

- Создан Deployment для указанного приложения.
- Создан Service для обеспечения доступа к приложению.
- Проверена доступность приложения через созданный Service.

Контрольные вопросы

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

Pod - Минимальная и самая маленькая вычислительная единица в Kubernetes.

Deployment (Развертывание) - Механизм для управления репликами Pod и их обновлениями.

Service (Сервис) - Абстракция для доступа к группе Pods (обычно управляемых Deployment).

2. Каково назначение Deployment в Kubernetes?

Deployment в Kubernetes предназначен для управления жизненным циклом приложений. Он обеспечивает:

Развёртывание и поддержку заданного количества идентичных Pods (реплик)

Бесшовные обновления (rolling updates) и откаты (rollbacks) версий приложения

Автоматическое восстановление Pods при сбоях, поддерживая желаемое состояние приложения.

3. Каково назначение Service в Kubernetes?

Service в Kubernetes обеспечивает стабильный доступ к приложению, абстрагируясь от изменяющихся Pod. Он:

Предоставляет постоянный IP/DNS-имя для группы Pod (даже при их пересоздании)

Балансирует нагрузку между репликами и определяет тип доступа (внутри/вне кластера через ClusterIP, NodePort или LoadBalancer).

4. Как создать Deployment в Kubernetes?

Создание YAML-манифеста (например, deployment.yaml):

Применение манифеста:

kubectl apply -f deployment.yaml

kubectl get deployments # Список Deployment

kubectl get pods # Список созданных Pods

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

Сервис в Kubernetes — это абстракция, определяющая логический набор подов и политику доступа к ним. Чтобы создать сервис, можно использовать YAML-файл или команду kubectl expose. Существует несколько основных типов сервисов в Kubernetes:

ClusterIP: Это самый распространенный тип сервиса. Он предоставляет внутренний IP-адрес для доступа к подам внутри кластера. Этот тип сервиса не доступен извне кластера по умолчанию.

NodePort: Этот тип сервиса позволяет доступ к подам через определенный порт на каждом узле кластера. Он часто используется для доступа к сервисам извне кластера.

LoadBalancer: Этот тип сервиса интегрируется с балансировщиками нагрузки в облачных провайдерах (например, AWS, GCP) и позволяет автоматически создать внешний балансировщик для доступа к сервису.

ExternalName: Этот тип сервиса позволяет использовать DNS-имя для доступа к сервисам, находящимся вне кластера Kubernetes.