

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

ОТЧЕТ

по дисциплине «Интеграция и развертывание программного обеспечения с
помощью контейнеров»

Лабораторная работа 3.2. Развертывание приложения в Kubernetes

Направление подготовки – 38.03.05 «Бизнес-информатика».

профиль подготовки – «Аналитика данных и эффективное управление»

Выполнила:

St92

Руководитель:

Москва
2025 год

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.

Ход работы:

Установим minikube:

```
dev@dev-vm:~/Downloads/lab3-2$ 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 26.8M 0 0:00:04 0:00:04 --:--:-- 27.5M
dev@dev-vm:~/Downloads/lab3-2$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Добавим пользователя в группу docker

```
dev@dev-vm:~/Downloads/lab3-2$ sudo usermod -aG docker $USER && newgrp docker
dev@dev-vm:~/Downloads/lab3-2$
```

Установим kubectl:

```
dev@dev-vm: ~
dev@dev-vm:~$ sudo snap install kubectl --classic
[sudo] password for dev:
kubectl 1.32.3 from Canonical✓ installed
```

Создадим Deployment для приложения чата на Node.js и Socket.IO.

```
! deployment.yaml x minikube-linux-amd64 ! service.yaml < index.html < Dc
! deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: chat-app
5    labels:
6      app: chat
7  spec:
8    replicas: 1
9    selector:
10     matchLabels:
11       app: chat
12    template:
13     metadata:
14       labels:
15         app: chat
16     spec:
17       containers:
18       - name: chat-app
19         image: chat-app:latest
20         imagePullPolicy: Never # Для использования локально собранного обра
21       ports:
22       - containerPort: 3000
23       env:
24       - name: PORT
25         value: "3000"
```

Рисунок 1 – Содержание Deployment.yaml

```
! deployment.yaml minikube-linux-amd64 ! service.yaml x <
! service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: chat-service
5  spec:
6    selector:
7      app: chat
8    ports:
9      - protocol: TCP
10        port: 80
11        targetPort: 3000
12    type: LoadBalancer
```

Рисунок 2 – Содержание service.yaml

Запустим minikube:

```

dev@dev-vm:~/Downloads/lab3-2$ minikube start --memory=2048mb --driver=docker
🐳 minikube v1.35.0 on Ubuntu 22.04 (vbox/amd64)
🔧 Using the docker driver based on user configuration
🔑 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% 22.66 M
> gcr.io/k8s-minikube/kicbase...: 500.31 MiB / 500.31 MiB 100.00% 13.53 M
🔥 Creating docker container (CPUs=2, Memory=2048MB) ...
🔧 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

```

Рисунок 3

Забилдим образ в окружении:

```

dev@dev-vm:~/Downloads/lab3-2$ docker build -t nodejs-socketio-chat:latest .
2025/04/10 21:40:12 in: [{}string{}]
2025/04/10 21:40:12 Parsed entitlements: []
[+] Building 32.8s (10/10) FINISHED                                docker:default
=> [internal] load build definition from dockerfile                0.1s
=> => transferring dockerfile: 498B                                0.0s
=> [internal] load metadata for docker.io/library/node:18-alpine  1.0s
=> [internal] load .dockerignore                                   0.0s
=> => transferring context: 2B                                       0.0s
=> [1/5] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d1189455 6.8s
=> => resolve docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d1189455 0.1s
=> => sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e 7.67kB / 7.67kB 0.0s
=> => sha256:929b04d7c782f04f615cf785488fed452b6569f87c73ff666ad553a7554f0006 1.72kB / 1.72kB 0.0s
=> => sha256:ee77c6cd7c1886ecc802ad6cedef3a8ec1ea27d1fb96162bf03dd3710839b8da 6.18kB / 6.18kB 0.0s
=> => sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870 3.64MB / 3.64MB 0.6s

```

Рисунок 4 – Выполнение команды для создания образа

```

dev@dev-vm:~/Downloads/lab3-2$ kubectl apply -f deployment.yaml
deployment.apps/nodejs-chat-app created
dev@dev-vm:~/Downloads/lab3-2$ kubectl apply -f service.yaml
service/nodejs-chat-app created
dev@dev-vm:~/Downloads/lab3-2$

```

Рисунок 5 – Создание ресурсов

Проверим доступность

```

dev@dev-vm:~/Downloads/lab3-2$ kubectl get services

```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
chat-service	LoadBalancer	10.102.143.49	<pending>	80:31780/TCP	5m10s
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	19m

Рисунок 6 – Выполнение команды для проверки доступности

Перейдем по ссылке в чат

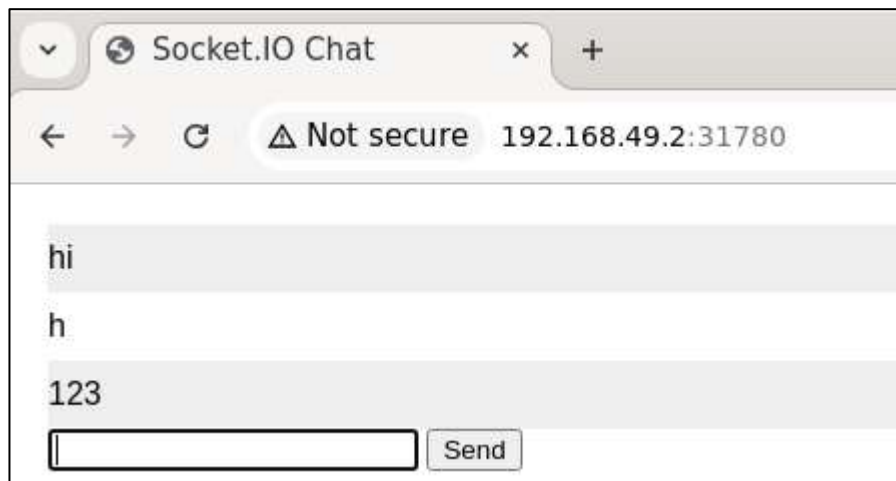


Рисунок 7 – Успешное подключение к чату

Удалим созданные ресурсы:

```
dev@dev-vm:~/Downloads/lab3-2$ kubectl delete -f service.yaml
service "chat-service" deleted
dev@dev-vm:~/Downloads/lab3-2$ kubectl delete -f deployment.yaml
deployment.apps "chat-app" deleted
```

Рисунок 8

Что такое Pod, Deployment и Service в Kubernetes?

Pod — это группа из одного или нескольких контейнеров с общим хранилищем и сетевыми ресурсами, а также спецификациями того, как запускать контейнеры.

Service - Абстрактный способ представить приложение, работающее на наборе Pod, как сетевую службу

Deployment Kubernetes используется для указания Kubernetes, как создавать или изменять экземпляры pod, которые содержат контейнерное приложение.

Каково назначение Deployment в Kubernetes?

Его основные функции:

- Декларативное управление подами (описывает желаемое состояние приложения).

- Масштабирование (увеличение или уменьшение числа реплик подов).
- Обновление и откат (постепенное обновление версий приложения с возможностью отката).
- Обеспечение отказоустойчивости (автоматически заменяет упавшие поды).

Каково назначение Service в Kubernetes?

Его основные функции:

- Постоянный IP и DNS-имя для доступа к набору подов (несмотря на их динамическую природу).
- Балансировка нагрузки между подами.
- Селекторная привязка (Service находит поды по меткам labels).
- Поддержка разных типов доступа:
 - ClusterIP (внутренний доступ в кластере).
 - NodePort (доступ через порт на ноде).
 - LoadBalancer (внешний доступ через облачный балансировщик)

Как создать Deployment в Kubernetes?

```
kubectl create deployment my-first-deployment --image=nginx:alpine
```

Как создать Service в Kubernetes и какие типы Services существуют?

Для создания Service нужно определить YAML-манифест и применить его с помощью `kubectl apply -f`

Типы Services в Kubernetes

Тип Service	Назначение	Пример использования
ClusterIP		
NodePort		
LoadBalancer		
ExternalName		