

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

Выполнила: st_95

Работа на лекции 29.03

Основы работы с Kubernetes

Интеграция и развертывание программного обеспечения с помощью
контейнеров

Направление подготовки

38.03.05 Бизнес-информатика

Профиль подготовки

Аналитика данных и эффективное управление

Москва

2025

Цель: Получить практические навыки работы с кластером Kubernetes, включая развертывание базовых компонентов, настройку мониторинга и работу с service mesh.

Задачи:

- Изучить основные концепции Kubernetes через практические вопросы.
- Научиться анализировать и применять манифесты Kubernetes.

Вариант 8. Kubernetes. Часть 1 (nginx v1.21.0)	Запустите Kubernetes локально (k3s или minikube). Проверьте работу системных контейнеров и приложите скриншот команд: <code>kubectl get po -n kube-system</code> .	Имеется YAML с деплоем для nginx . Измените файл: <ul style="list-style-type: none">– Добавьте кастомные параметры запуска;– Фиксируйте образ на версии 1.21.0;– Добавьте Service для доступа. Приложите итоговый YAML.	Напишите команды <code>kubectl</code> для контейнера: <ul style="list-style-type: none">– Выполнить внутри контейнера команду <code>ps aux</code>;– Просмотреть логи за 5 минут;– Удалить pod;– Пробросить порт для отладки.	Доп. задание*: Создайте YAML для: <ul style="list-style-type: none">– ConfigMap с настройками для nginx (например, изменение стандартного порта);– Deployment, использующий ConfigMap;– Ingress, направляющий запросы по пути <code>/site</code> на сервис.
---	--	---	---	--

Ход работы:

1. Установка minikube (Рис. 1).

```
kate@DESKTOP-76ATD1J:~$ curl -LO https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left     Speed
  0     0    0     0    0     0      0  0 --:--:-- --:--:-- --:--:--    0
  0     0    0     0    0     0      0  0 --:--:-- --:--:-- --:--:--    0
100 119M 100 119M    0     0 41.6M    0  0:00:02  0:00:02 --:--:-- 68.6M
kate@DESKTOP-76ATD1J:~$ sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64
[sudo] password for kate:
kate@DESKTOP-76ATD1J:~$
```

Рис. 1

2. При запуске кластера возникла ошибка, потому что не может определить драйвер (Рис. 2).

```
kate@DESKTOP-76ATD1J:~$ minikube start
minikube v1.35.0 on Ubuntu 24.04 (amd64)
Unable to pick a default driver. Here is what was considered, in preference order:
Alternatively you could install one of these drivers:
  • docker: Not installed: exec: "docker": executable file not found in $PATH
  • kvm2: Not installed: exec: "virsh": executable file not found in $PATH
  • qemu2: Not installed: exec: "qemu-system-x86_64": executable file not found in $PATH
  • podman: Not installed: exec: "podman": executable file not found in $PATH
  • virtualbox: Not installed: unable to find VBoxManage in $PATH

Exiting due to DRV_NOT_DETECTED: No possible driver was detected. Try specifying --driver, or see https://minikube.sigs.k8s.io/docs/start/
```

Рис. 2

Решение - установка драйвера docker (Рис. 3).

```
kate@DESKTOP-76ATD1J:~$ sudo apt-get install docker.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base iptables libip4tc2 libip6tc2 libnetfilter-conntrack3
  libnfnetlink0 libnftables1 libnftnl11 nftables pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools btrfs-progs cgroupfs-mount | cgroup-lite debootstrap docker-buildx docker-compose-v2 docker-doc
  rinse zfs-fuse | zfsutils firewalld
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io iptables libip4tc2 libip6tc2 libnetfilter-conntrack3
  libnfnetlink0 libnftables1 libnftnl11 nftables pigz runc ubuntu-fan
0 upgraded, 16 newly installed, 0 to remove and 59 not upgraded.
Need to get 79.6 MB of archives.
After this operation, 306 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Рис. 3

Повторный запуск кластера с определённым драйвером (Рис. 4).

```

kate@DESKTOP-76ATD1J:~$ minikube start --driver=docker
minikube v1.35.0 on Ubuntu 24.04 (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% 33.21 M
> gcr.io/k8s-minikube/kicbase...: 500.31 MiB / 500.31 MiB 100.00% 16.51 M
Creating docker container (CPUs=2, Memory=3900MB) ...
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
kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
kate@DESKTOP-76ATD1J:~$

```

Рис. 4

3. Установка kubectl

Установка последней версии (Рис. 5).

```

kate@DESKTOP-76ATD1J:~$ curl -LO https://dl.k8s.io/release/`curl -LS 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
100 138 100 138 0 0 534 0 --:--:-- --:--:-- --:--:-- 536
100 7 100 7 0 0 12 0 --:--:-- --:--:-- --:--:-- 12
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 138 100 138 0 0 719 0 --:--:-- --:--:-- --:--:-- 722
100 54.6M 100 54.6M 0 0 43.6M 0 0:00:01 0:00:01 --:--:-- 59.1M
kate@DESKTOP-76ATD1J:~$

```

Рис. 5

Делаем файл исполняемым и переносим бинарный файл в директорию, проверяем установленную версию (Рис. 6).

```

kate@DESKTOP-76ATD1J:~$ chmod +x ./kubectl
kate@DESKTOP-76ATD1J:~$ sudo mv ./kubectl /usr/local/bin/kubectl
kate@DESKTOP-76ATD1J:~$ kubectl version --client
Client Version: v1.32.3
Kustomize Version: v5.5.0
kate@DESKTOP-76ATD1J:~$

```

Рис. 6

4. Проверка kubectl на работоспособность:

- Проверка нодов (Рис. 7).

```

kate@DESKTOP-76ATD1J:~$ kubectl get node
NAME          STATUS    ROLES          AGE      VERSION
minikube      Ready     control-plane   6m25s    v1.32.0

```

Рис. 7

- Вывод списка подов во всех namespace (Рис. 8).

```
kate@DESKTOP-76ATD1J:~$ kubectl get po -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	coredns-668d6bf9bc-gjl8d	1/1	Running	0	8m32s
kube-system	etcd-minikube	1/1	Running	0	8m37s
kube-system	kube-apiserver-minikube	1/1	Running	0	8m37s
kube-system	kube-controller-manager-minikube	1/1	Running	0	8m38s
kube-system	kube-proxy-w9pk8	1/1	Running	0	8m32s
kube-system	kube-scheduler-minikube	1/1	Running	0	8m37s
kube-system	storage-provisioner	1/1	Running	1 (8m ago)	8m35s

Рис. 8

- Вывод списка сервисов в текущем namespace (Рис. 9).

```
kate@DESKTOP-76ATD1J:~$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	11m

```
kate@DESKTOP-76ATD1J:~$
```

Рис. 9

5. Установка графического интерфейса:

Запуска дашборда (Рис. 10).

```
kate@DESKTOP-76ATD1J:~$ minikube dashboard
```

```

Enabling dashboard ...
  Using image docker.io/kubernetesui/dashboard:v2.7.0
  Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
Some dashboard features require the metrics-server addon. To enable all features please run:

    minikube addons enable metrics-server

Verifying dashboard health ...
Launching proxy ...
Verifying proxy health ...
Opening http://127.0.0.1:41187/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
http://127.0.0.1:41187/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/

```

Рис. 10

Прокидываем порт 41187 (Рис. 11)

Port	Forwarded Address	Running Process	Origin
41187	localhost:41187	Process information unavailable	User Forwarded

Рис. 11

Открываем дашборд в браузере (Рис. 12).

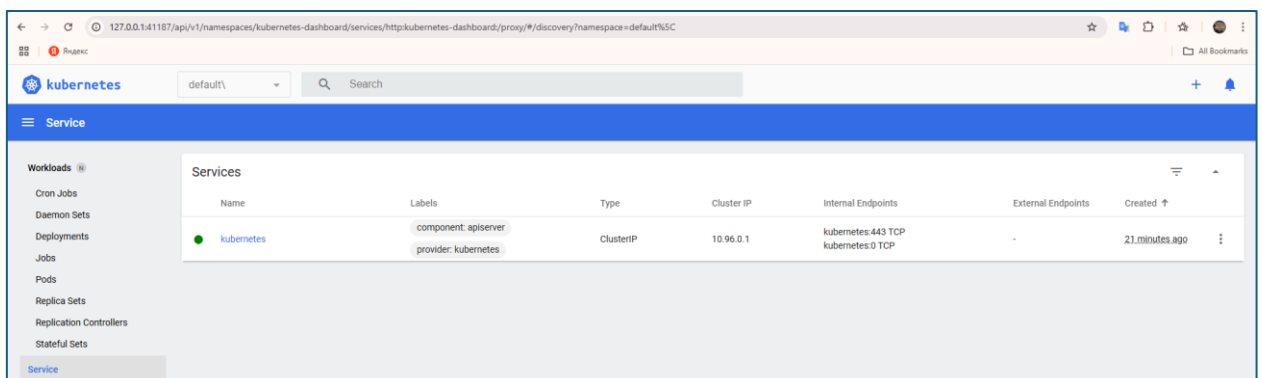


Рис. 12

Выполнение практической части по групповому заданию:

1. **Постановка задачи:** Создать и развернуть веб-сервис на основе FastAPI, использующий Redis в качестве базы данных, с использованием Kubernetes и Minikube для локальной разработки и тестирования.

Ожидаемые результаты:

- Рабочее FastAPI приложение, способное взаимодействовать с Redis для хранения и получения данных.
- Корректно настроенные Kubernetes ресурсы, обеспечивающие надежное и масштабируемое развертывание приложения.
- Возможность доступа приложению через уникальный URL в браузере или с помощью инструментов для тестирования API

2. Дерево проекта:

```
kate@DESKTOP-76ATD1J:~/projects/29_03_lab/CI_CD_25/practice/lab4_1$ tree .
.
├── Dockerfile
├── README.md
├── __init__.py
├── configmap.yml
├── docs
│   ├── 1.png
│   ├── 2.png
│   ├── 33.png
│   ├── 4.png
│   ├── 55.png
│   ├── 66.png
│   ├── 77.png
│   └── __init__.py
├── fastapi-deployment-and-service.yml
├── main.py
├── redis-deployment-and-service.yml
├── requirements.txt
└── secret.yml

2 directories, 17 files
```

3. **Технологический стек:**

Ход работы:

- 1) Запуск кластера с определённым количеством памяти и указанным драйвером «docker» (Рис. 13).

```

kate@DESKTOP-76ATD1J:~/projects/29_03_lab$ minikube start --memory=2048mb --driver=docker
😄 minikube v1.35.0 on Ubuntu 24.04 (amd64)
🌟 Using the docker driver based on existing profile
! You cannot change the memory size for an existing minikube cluster. Please first delete the cluster
👍 Starting "minikube" primary control-plane node in "minikube" cluster
📦 Pulling base image v0.0.46 ...
🔄 Restarting existing docker container for "minikube" ...
🔧 Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
🔍 Verifying Kubernetes components...
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
  ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
  ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
💡 Some dashboard features require the metrics-server addon. To enable all features please run:

    minikube addons enable metrics-server

🌟 Enabled addons: storage-provisioner, dashboard, default-storageclass
🔧 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

```

4.

Рис. 13

2) Настраиваем окружение (Рис. 14).

```

kate@DESKTOP-76ATD1J:~/projects/29_03_lab$ eval $(minikube docker-env)

```

Рис. 14

3) Билдим образ fastapi-app(Рис. 15).

```

kate@DESKTOP-76ATD1J:~/projects/29_03_lab/CI_CD_25/practice/lab4_1$ docker build -t fastapi-app:local .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
Install the buildx component to build images with BuildKit:
https://docs.docker.com/go/buildx/

Sending build context to Docker daemon 524.8kB
Step 1/6 : FROM python:3.10
3.10: Pulling from library/python
7cd785773db4: Pull complete
091eb8249475: Pull complete
255774e0027h: Pull complete

```

Рис. 15

Проверка образа (Рис. 16).

```

kate@DESKTOP-76ATD1J:~/projects/29_03_lab/CI_CD_25/practice/lab4_1$ docker images | grep fastapi-app
fastapi-app      local          aec42603c6ca   About a minute ago   1.03GB

```

Рис. 16

4) Создаём конфигурационные, конфиденциальные и деплоймент файлы (Рис. 17).

```
kate@DESKTOP-76ATD1J:~/projects/29_03_lab/CI_CD_25/practice/lab4_1$ kubectl create -f configmap.yml
configmap/fastapi-config created
kate@DESKTOP-76ATD1J:~/projects/29_03_lab/CI_CD_25/practice/lab4_1$ kubectl create -f secret.yml
secret/fastapi-secret created
kate@DESKTOP-76ATD1J:~/projects/29_03_lab/CI_CD_25/practice/lab4_1$ kubectl create -f fastapi-deployment-and-service.yml
deployment.apps/fastapi-deployment created
service/fastapi-service created
kate@DESKTOP-76ATD1J:~/projects/29_03_lab/CI_CD_25/practice/lab4_1$ kubectl create -f redis-deployment-and-service.yml
deployment.apps/redis-deployment created
service/redis-service created
```

Рис. 17

5) Проверяем запущенные pods (Рис. 18).

```
kate@DESKTOP-76ATD1J:~/projects/29_03_lab/CI_CD_25/practice/lab4_1$ kubectl get pods -w
```

NAME	READY	STATUS	RESTARTS	AGE
fastapi-deployment-cf4dc69bc-bhj9w	1/1	Running	0	6m54s
fastapi-deployment-cf4dc69bc-hcfqp	1/1	Running	0	6m54s
redis-deployment-748ffbc5f5-hh2dv	1/1	Running	0	6m38s

Рис. 18

6) Пробрасываем порт для доступ через браузер (Рис. 19).

```
kate@DESKTOP-76ATD1J:~/projects/29_03_lab/CI_CD_25/practice/lab4_1$ kubectl port-forward svc/fastapi-service 8000:80
Forwarding from 127.0.0.1:8000 -> 8000
Forwarding from [::1]:8000 -> 8000
Handling connection for 8000
Handling connection for 8000
Handling connection for 8000
```

Рис. 19

7) Проверяем доступ через браузер (Рис. 20).

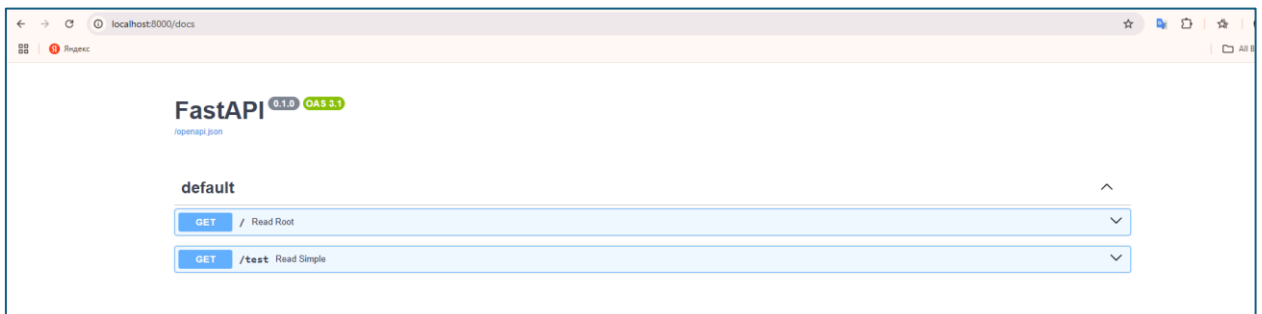


Рис. 20

Отправляем Get запрос и получаем ответ (Рис. 21).

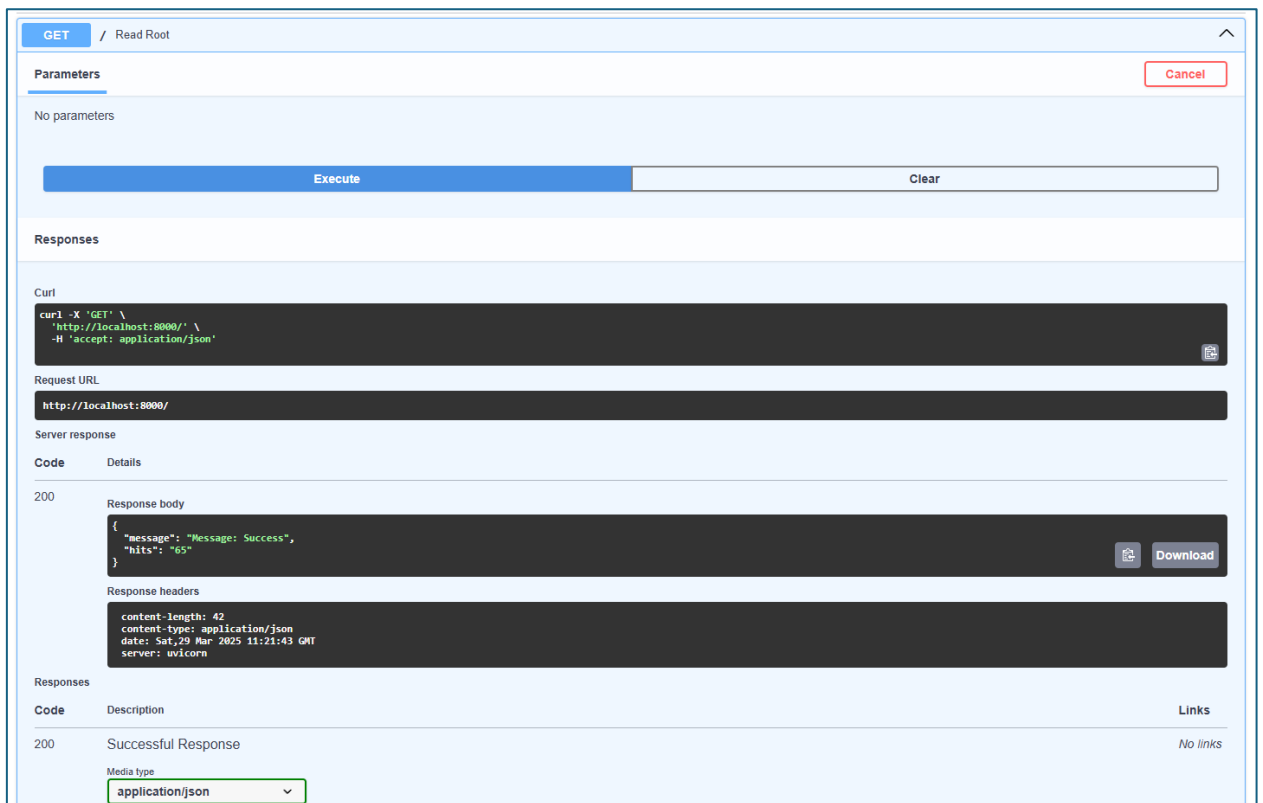


Рис. 21

8) Проверяем сервисы в minikube (Рис. 22)

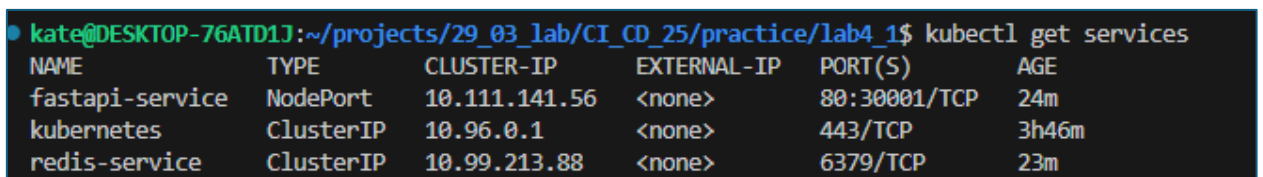


Рис. 22

9) Запускаем дашборд (Рис. 23).

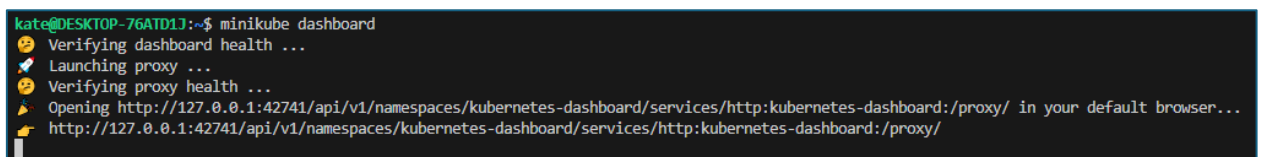


Рис. 23

10) Просмотрим дашборд (Рис. 24).

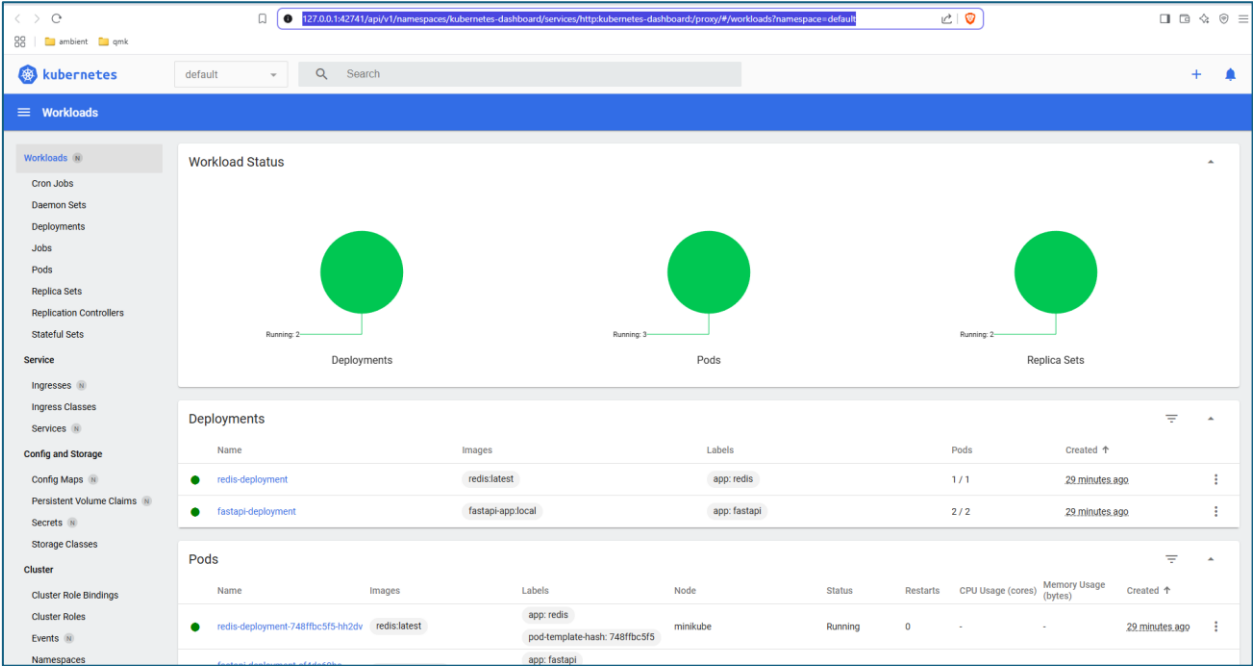


Рис. 24

Запущено 3 пода (Рис. 25).

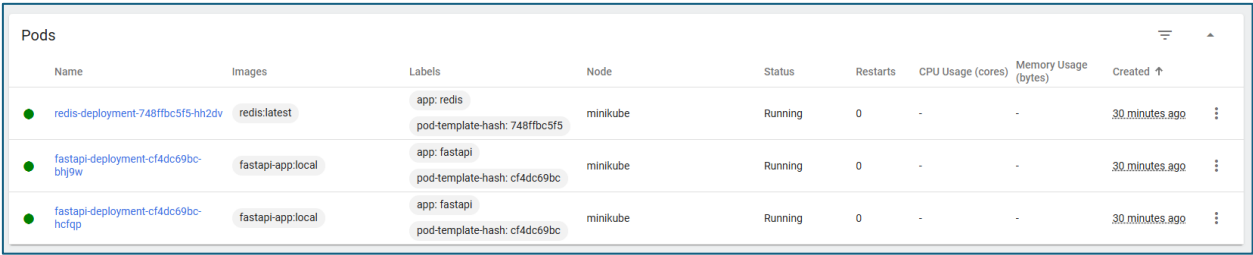


Рис. 25

Выполнение индивидуального задания:

Постановка задачи:

Создать и развернуть веб-сервис на основе nginx, использующий PostgreSQL в качестве базы данных, с использованием Kubernetes и Minikube для локальной разработки и тестирования.

Ожидаемые результаты:

- Рабочее nginx приложение, способное взаимодействовать с PostgreSQL для хранения и получения данных.
- Корректно настроенные Kubernetes ресурсы, обеспечивающие надёжное и масштабируемое развертывание приложения.
- Возможность доступа к приложению через указанный URL в браузере или с помощью инструментов для тестирования API.

Технологический стек:

1. Фронтенд

- Nginx 1.21.0 – веб-сервер для раздачи статики (HTML/JS) и проксирования API-запросов
- HTML5 + JavaScript – клиентская часть (проверка подключения к PostgreSQL)

2. Бэкенд

- FastAPI (Python) – REST API для взаимодействия с PostgreSQL
- Psycopg2 – драйвер PostgreSQL для Python

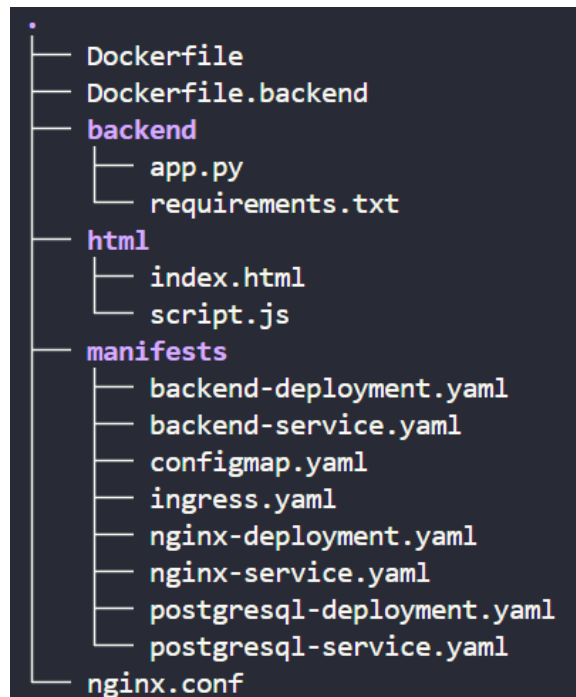
3. База данных

- PostgreSQL 13 – реляционная СУБД

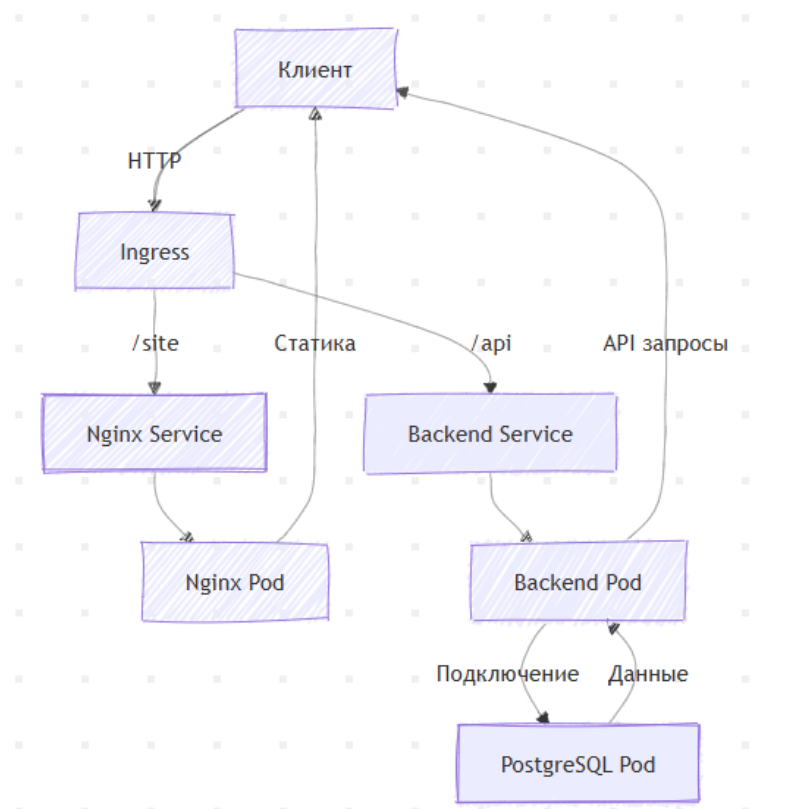
4. Инфраструктура

- Kubernetes (Minikube) – оркестрация контейнеров
- Docker – контейнеризация компонентов
- ConfigMap – хранение конфигурации Nginx
- Ingress – маршрутизация запросов (/site → Nginx, /api → FastAPI)

Дерево проекта:



Архитектурное взаимодействие:



Ход работы:

1) Запуск minikube с определённым количеством памяти и драйвером docker (Рисунок 1).

```
kate@beady:~/nginx-postgres$ minikube start --memory=2048mb --driver=docker
minikube v1.35.0 on Ubuntu 24.04 (amd64)
Using the docker driver based on user configuration
Using Docker driver with root privileges
! For an improved experience it's recommended to use Docker Engine instead of Docker Desktop.
Docker Engine installation instructions: https://docs.docker.com/engine/install/#server
Starting "minikube" primary control-plane node in "minikube" cluster
Pulling base image v0.0.46 ...
Downloading Kubernetes v1.32.0 preload ...
> gcr.io/k8s-minikube/kicbase...: 500.31 MiB / 500.31 MiB 100.00% 25.82 M
> preloaded-images-k8s-v18-v1...: 333.57 MiB / 333.57 MiB 100.00% 11.39 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 1

2) Проверяем работу кластера (Рисунок 2).

```
kate@beady:~/nginx-postgres$ kubectl get nodes
NAME          STATUS    ROLES          AGE    VERSION
minikube      Ready     control-plane   5m7s   v1.32.0
```

Рисунок 2

3) Создаём структуру проекта (Рисунок 3)

```
├── Dockerfile
├── html
│   └── index.html
├── manifests
│   ├── configmap.yaml
│   ├── ingress.yaml
│   ├── nginx-deployment.yaml
│   ├── nginx-service.yaml
│   ├── postgresql-deployment.yaml
│   └── postgresql-service.yaml
└── nginx.conf
```

Рисунок 3

4) Описание Dockerfile (Рисунок 4).



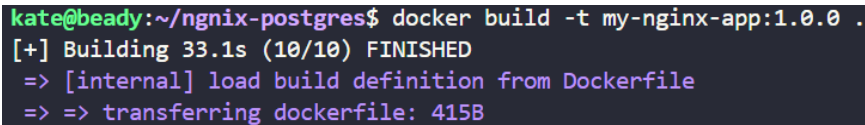
```

Dockerfile > ...
1 FROM nginx:1.21.0
2 COPY nginx.conf /etc/nginx/nginx.conf
3 COPY html /usr/share/nginx/html
4 EXPOSE 8080
5 CMD ["nginx", "-g", "daemon off;"]

```

Рисунок 4

5) Собираем образ (Рисунок 5).



```

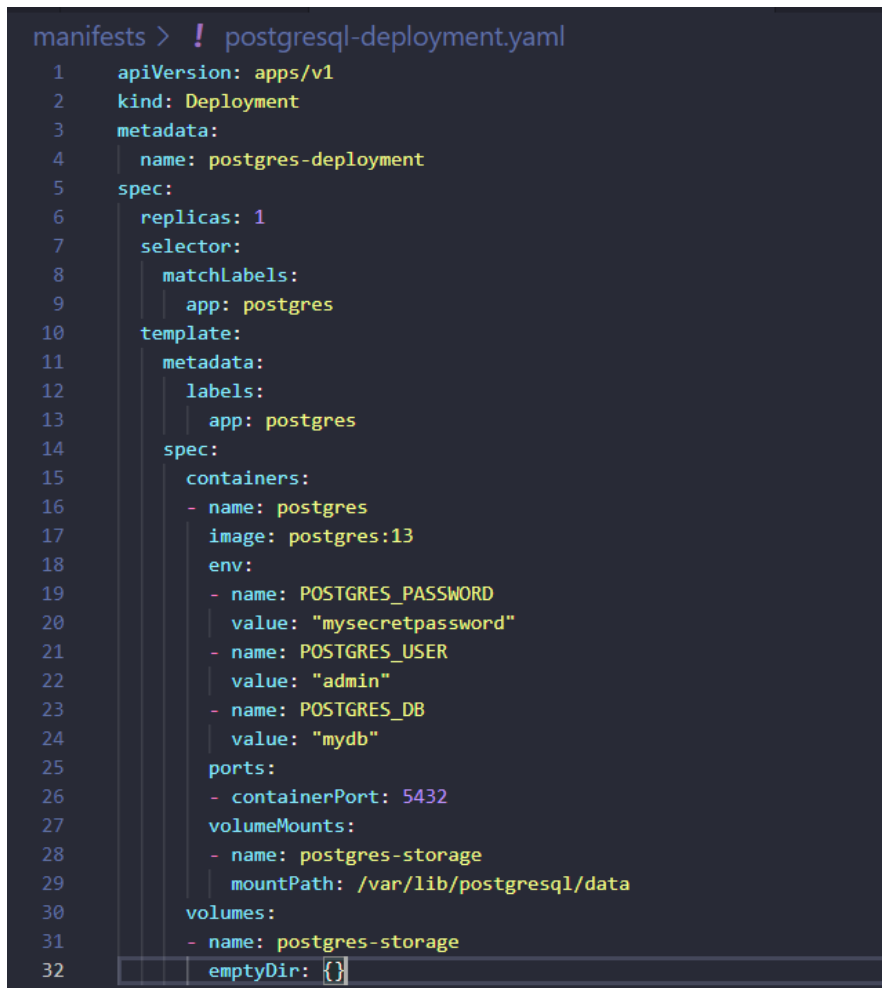
kate@beady:~/nginx-postgres$ docker build -t my-nginx-app:1.0.0 .
[+] Building 33.1s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 415B

```

Рисунок 5

6) Развертывание PostgreSQL:

Опишем deployment (Рисунок 6).



```

manifests > ! postgresql-deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: postgres-deployment
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: postgres
10   template:
11     metadata:
12       labels:
13         app: postgres
14     spec:
15       containers:
16       - name: postgres
17         image: postgres:13
18         env:
19         - name: POSTGRES_PASSWORD
20           value: "mysecretpassword"
21         - name: POSTGRES_USER
22           value: "admin"
23         - name: POSTGRES_DB
24           value: "mydb"
25       ports:
26       - containerPort: 5432
27       volumeMounts:
28       - name: postgres-storage
29         mountPath: /var/lib/postgresql/data
30     volumes:
31     - name: postgres-storage
32       emptyDir: {}

```

Рисунок 6

Применяем деплоймент (Рисунок 7).

```
kate@beady:~/nginx-postgres$ kubectl apply -f manifests/postgresql-deployment.yaml
deployment.apps/postgres created
kate@beady:~/nginx-postgres$
```

Рисунок 7

Пишем Service для PostgreSQL (Рисунок 8).

```
manifests > ! postgresql-service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: postgres-service
5  spec:
6    selector:
7      app: postgres
8    ports:
9      - protocol: TCP
10      port: 5432
11        targetPort: 5432
12    type: ClusterIP
```

Рисунок 8

Применяем (Рисунок 9).

```
kate@beady:~/nginx-postgres$ kubectl apply -f manifests/postgresql-service.yaml
service/postgres created
```

Рисунок 9

7) Создание ConfigMap (Рисунок 10).

```

manifests > ! configmap.yaml
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: nginx-config
5  data:
6    nginx.conf: |
7      events {
8        worker_connections 1024;
9      }
10     http {
11       server {
12         listen 8080;
13         server_name localhost;
14
15         location / {
16           root /usr/share/nginx/html;
17           index index.html;
18         }
19       }
20     }

```

Рисунок 10

Применяем configmap (Рисунок 11).

```

kate@beady:~/nginx-postgres$ kubectl apply -f manifests/configmap.yaml
configmap/nginx-config created

```

Рисунок 11

8) Развёртывание nginx:

Создаём deployment для nginx (Рисунок 12).


```

manifests > ! nginx-deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5  spec:
6    replicas: 2
7    selector:
8      matchLabels:
9        app: nginx
10   template:
11     metadata:
12       labels:
13         app: nginx
14     spec:
15       containers:
16       - name: nginx
17         image: nginx:1.21.0
18         ports:
19         - containerPort: 8080
20         env:
21         - name: NGINX_PORT
22           value: "8080"
23         volumeMounts:
24         - name: nginx-config
25           mountPath: /etc/nginx/nginx.conf
26           subPath: nginx.conf
27       volumes:
28       - name: nginx-config
29         configMap:
30           name: nginx-config

```

Рисунок 12

Применяем (Рисунок 13).

```

kate@beady:~/nginx-postgres$ kubectl apply -f manifests/nginx-deployment.yaml
deployment.apps/nginx created

```

Рисунок 13

Создание service для nginx (Рисунок 14).

```

manifests > ! nginx-service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-service
5  spec:
6    selector:
7      app: nginx
8    ports:
9      - protocol: TCP
10      port: 80
11      targetPort: 8080
12    type: NodePort

```

Рисунок 14

Применяем (Рисунок 15).

```

kate@beady:~/nginx-postgres$ kubectl apply -f manifests/nginx-service.yaml
service/nginx created
kate@beady:~/nginx-postgres$

```

Рисунок 15

9) Настройка Ingress (Рисунок 16).

```

manifests > ! ingress.yaml
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: nginx-ingress
5    annotations:
6      nginx.ingress.kubernetes.io/rewrite-target: /$1
7  spec:
8    rules:
9      - http:
10        paths:
11          - path: /site(/|$)(.*)
12            pathType: Prefix
13            backend:
14              service:
15                name: nginx-service
16                port:
17                  number: 80

```

Рисунок 16

Применяем (Рисунок 17, Рисунок 18).

```
kate@beady:~/nginx-postgres$ minikube addons enable ingress
💡 ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.4
▪ Using image registry.k8s.io/ingress-nginx/controller:v1.11.3
▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.4
🔍 Verifying ingress addon...
🌟 The 'ingress' addon is enabled
```

Рисунок 17

```
kate@beady:~/nginx-postgres$ kubectl apply -f manifests/ingress.yaml
ingress.networking.k8s.io/nginx-ingress created
```

Рисунок 18

Опишем nginx.conf (Рисунок 19).

```
nginx.conf
1  events {
2      worker_connections 1024;
3  }
4
5  http {
6      server {
7          listen 8080;
8          server_name localhost;
9
10         location / {
11             root /usr/share/nginx/html;
12             index index.html;
13         }
14     }
15 }
```

Рисунок 19

Опишем index.html (Рисунок 20).

```

html > <> index.html
1  <!DOCTYPE html>
2  <html Lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Nginx + PostgreSQL App</title>
7      <style>
8          body { font-family: Arial, sans-serif; margin: 40px; }
9          .container { max-width: 800px; margin: 0 auto; }
10         .db-result { margin-top: 20px; padding: 15px; border: 1px solid #ddd; }
11     </style>
12 </head>
13 <body>
14     <div class="container">
15         <h1>Web Service with PostgreSQL</h1>
16
17         <div>
18             <button onclick="testDbConnection()">Test PostgreSQL Connection</button>
19             <button onclick="getDbVersion()">Get PostgreSQL Version</button>
20         </div>
21
22         <div id="db-result" class="db-result">
23             DB results will appear here
24         </div>
25     </div>
26
27     <script>
28         async function testDbConnection() {
29             try {
30                 const response = await fetch('/api');
31                 const data = await response.text();
32                 document.getElementById('db-result').innerText =
33                     `Connection successful! Response: ${data}`;

```

Рисунок 20

10) Проверка работы:

Проверяем поды (Рисунок 21).

```

kate@beady:~/nginx-postgres$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-8496f5d94-blhtf              1/1     Running   0           22m
nginx-8496f5d94-kb6xm              1/1     Running   0           22m
postgres-5c78f459dc-zvwrc          1/1     Running   0           34m

```

Рисунок 21

Проверяем работу сервисов (Рисунок 22).

```

kate@beady:~/nginx-postgres$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP          88m
nginx        NodePort    10.98.63.18  <none>        80:31568/TCP     22m
postgres     ClusterIP   10.98.56.200 <none>        5432/TCP         32m

```

Рисунок 22

Проверяем ingress (Рисунок 23).

```
kate@beady:~/nginx-postgres$ kubectl get ingress
NAME          CLASS  HOSTS  ADDRESS        PORTS  AGE
nginx-ingress  nginx  *      192.168.49.2    80     13m
```

Рисунок 23

Пробрасываем туннель по ingress(Рисунок 24).

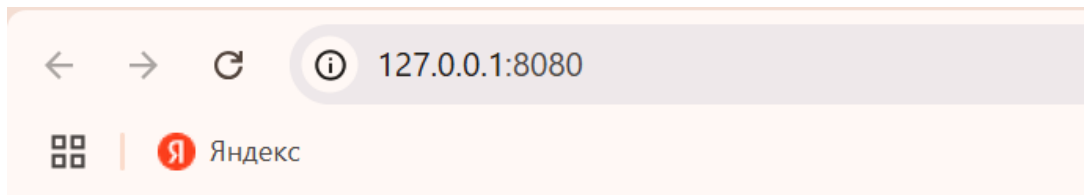
```
o kate@beady:~/nginx-postgres$ minikube tunnel
✓ Tunnel successfully started

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

✳ Starting tunnel for service nginx-service.
! The service/ingress nginx-ingress requires privileged ports to be exposed: [80 443]
🔑 sudo permission will be asked for it.
✳ Starting tunnel for service nginx-ingress.
```

Рисунок 24

Проверка работоспособности в браузере (Рисунок 25).



Welcome to Nginx!

PostgreSQL connection will be here later.

Рисунок 25

Доступность была проверена, далее добавим проверку подключения к PostgreSQL через фронтенд.

1) Опишем зависимости (Рисунок 26).

```
backend > requirements.txt
1 fastapi==0.95.2
2 uvicorn==0.22.0
3 psycopg2-binary==2.9.6
```

Рисунок 26

2) Создадим бекенд на FAST-API (Рисунок 27).

```

backend > app.py
1  from fastapi import FastAPI
2  from fastapi.middleware.cors import CORSMiddleware
3  import psycopg2
4
5  app = FastAPI()
6
7  # Настройки CORS
8  app.add_middleware(
9      CORSMiddleware,
10     allow_origins=["*"],
11     allow_methods=["*"],
12     allow_headers=["*"],
13 )
14
15 # Конфиг PostgreSQL (используем сервисное имя Kubernetes)
16 DB_CONFIG = {
17     "host": "postgres-service",
18     "database": "mydb",
19     "user": "admin",
20     "password": "mysecretpassword"
21 }
22
23 @app.get("/api/check")
24 def check_connection():
25     try:
26         conn = psycopg2.connect(**DB_CONFIG)
27         conn.close()
28         return {"status": "success", "message": "PostgreSQL connection OK"}
29     except Exception as e:
30         return {"status": "error", "message": str(e)}
31
32 @app.get("/api/version")
33 def get_version():
34     try:
35         conn = psycopg2.connect(**DB_CONFIG)
36         cur = conn.cursor()
37         cur.execute("SELECT version();")
38         version = cur.fetchone()[0]
39         conn.close()
40         return {"status": "success", "version": version}

```

Рисунок 27

3) Обновляем фронтенд (Рисунок 28).

```
html > <> index.html > html
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>PostgreSQL Checker</title>
5      <script src="script.js"></script>
6      <style>
7          body { font-family: Arial, sans-serif; margin: 20px; }
8          button { padding: 10px; margin: 5px; }
9          #result { margin-top: 20px; padding: 10px; border: 1px solid #ddd; }
10     </style>
11 </head>
12 <body>
13     <h1>PostgreSQL Connection Test</h1>
14     <button onclick="testConnection()">Test Connection</button>
15     <button onclick="getVersion()">Get Version</button>
16     <div id="result"></div>
17 </body>
18 </html>
```

Рисунок 28

4) Добавим Java-script (Рисунок 29)

```
html > JS script.js > getVersion > [E] response
1  async function testConnection() {
2      const response = await fetch('/check');
3      const data = await response.json();
4      document.getElementById('result').innerHTML =
5          `<strong>Status:</strong> ${data.status}<br>
6          <strong>Message:</strong> ${data.message}`;
7  }
8
9  async function getVersion() {
10     const response = await fetch('/version');
11     const data = await response.json();
12     if (data.status === "success") {
13         document.getElementById('result').innerHTML =
14             `<strong>PostgreSQL Version:</strong><br>${data.version}`;
15     } else {
16         document.getElementById('result').innerHTML =
17             `<strong>Error:</strong> ${data.message}`;
18     }
19 }
```

Рисунок 29

5) Обновим nginx.conf (Рисунок 30)

```

nginx.conf
1  events {
2      worker_connections 1024;
3  }
4
5  http {
6      server {
7          listen 8080;
8
9          location /api/ {
10
11              proxy_pass http://backend-service/api/;
12              proxy_set_header Host $host;
13          }
14
15          location / {
16              root /usr/share/nginx/html;
17              index index.html;
18          }
19      }
20  }

```

Рисунок 30

6) Создаём Dockerfile для бекенда (Рисунок 31).

```

Dockerfile.backend > ...
1  FROM python:3.9-slim
2
3  WORKDIR /app
4
5  COPY backend/requirements.txt .
6  RUN pip install --no-cache-dir -r requirements.txt
7
8  COPY backend/app.py .
9
10 CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]

```

Рисунок 31

7) Добавляем манифесты для бекенда:

- Deployment (Рисунок 32)


```

manifests > ! backend-deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: backend-deployment
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: backend
10   template:
11     metadata:
12       labels:
13         app: backend
14     spec:
15       containers:
16       - name: backend
17         image: backend-app:1.0
18         ports:
19         - containerPort: 8000
20         env:
21         - name: PG_HOST
22           value: "postgres-service"

```

Рисунок 32

- Service (Рисунок 33)

```

manifests > ! backend-service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: backend-service
5  spec:
6    selector:
7      app: backend
8    ports:
9      - protocol: TCP
10        port: 80
11        targetPort: 8000

```

Рисунок 33

- 8) Собираем образ (Рисунок 34).

```

kate@beady:~/nginx-postgres$ docker build -t backend-app:1.0 -f Dockerfile.backend .
[+] Building 96.4s (11/11) FINISHED
=> [internal] load build definition from Dockerfile.backend 0.9s
=> => transferring dockerfile: 255B 0.3s
=> [internal] load metadata for docker.io/library/python:3.9-slim 6.0s
=> [auth] library/python:pull token for registry-1.docker.io 0.0s
=> [internal] load .dockerignore 0.1s
=> => transferring context: 2B 0.0s
=> [1/5] FROM docker.io/library/python:3.9-slim@sha256:e52ca5f579cc58fed41efcbb55a0ed5dccf6c7a156cba76acfb4ab42fc19d 37.1s

```

Рисунок 34

- 9) Применяем ресурсы (Рисунок 35)

```

● kate@beady:~/nginx-postgres$ kubectl apply -f manifests/
deployment.apps/backend-deployment created
service/backend-service created
configmap/nginx-config unchanged
ingress.networking.k8s.io/nginx-ingress unchanged
deployment.apps/nginx-deployment configured
service/nginx-service configured
deployment.apps/postgres-deployment unchanged
service/postgres-service configured

```

Рисунок 35

10) Проверка подов (Рисунок 36)

```

● kate@beady:~/nginx-postgres$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
backend-deployment-69cc759d44-vnmrf 1/1     Running   0           4m10s
debug                                1/1     Running   0           37m
nginx-deployment-5f9c64559-v6ndt     1/1     Running   0           4m9s
postgres-deployment-6dd886f6f7-4dhrz 1/1     Running   0           57m

```

Рисунок 36

11) Проверка работоспособности в браузере (Рисунок 37)

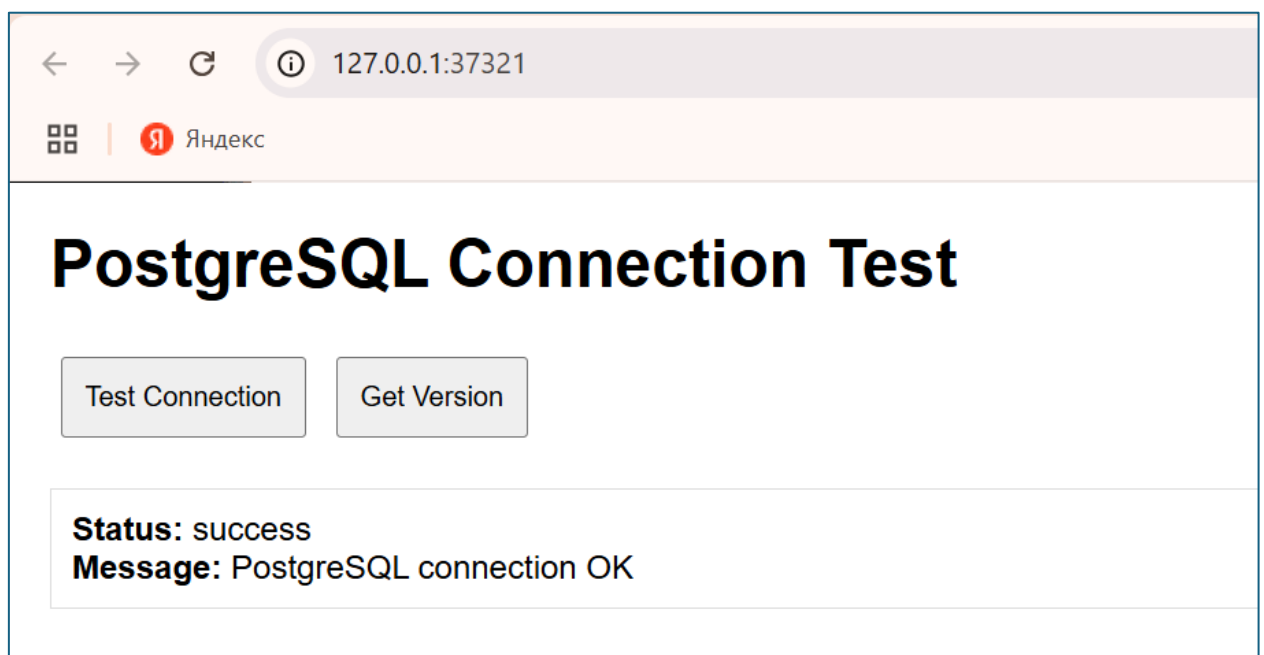
```

● kate@beady:~/nginx-postgres$ minikube service nginx-service --url
http://127.0.0.1:37321
! Because you are using a Docker driver on linux, the terminal needs to be open to run it.

```

Рисунок 37

12) Дёргаем ручку check connection ()



13) Проверяем версию (Рисунок 38)

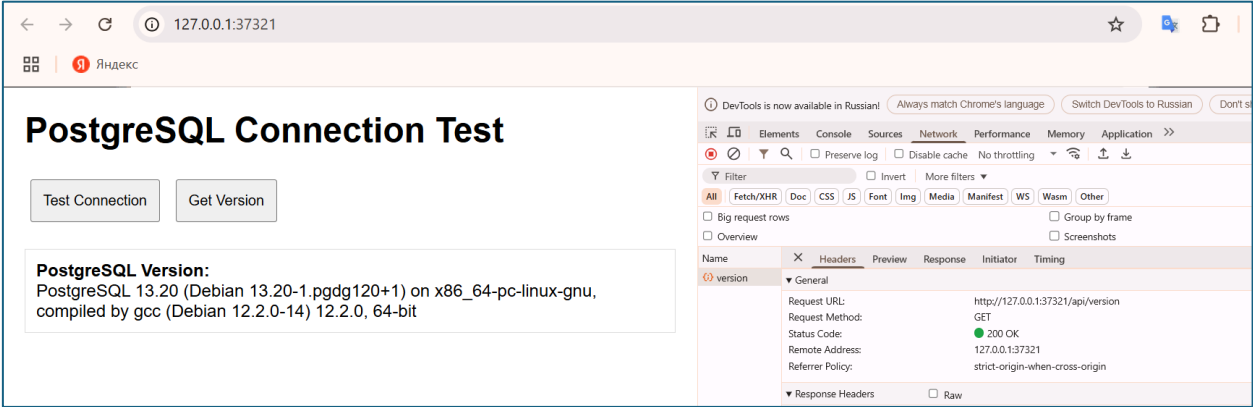


Рисунок 38

Зафиксируем выполнение индивидуального задания:

Вариант 8. Kubernetes. Часть 1 (nginx v1.21.0)	Запустите Kubernetes локально (k3s или minikube). Проверьте работу системных контейнеров и приложите скриншот команды: <code>kubectl get po -n kube-system</code> .	Имеется YAML с деплоем для nginx . Измените файл: <ul style="list-style-type: none">– Добавьте кастомные параметры запуска;– Фиксируйте образ на версии 1.21.0;– Добавьте Service для доступа. Приложите итоговый YAML.	Напишите команды <code>kubectl</code> для контейнера: <ul style="list-style-type: none">– Выполнить внутри контейнера команду <code>ps aux</code>;– Просмотреть логи за 5 минут;– Удалить pod;– Пробросить порт для отладки.	Доп. задание*: Создайте YAML для: <ul style="list-style-type: none">– ConfigMap с настройками для nginx (например, изменение стандартного порта);– Deployment, использующий ConfigMap;– Ingress, направляющий запросы по пути <code>/site</code> на сервис.
---	---	---	---	--

Задание 2 (Рисунок 39)

```

manifests > ! nginx-deployment.yaml
5   spec:
7   selector:
8     matchLabels:
10  template:
11    metadata:
12      labels:
13        app: nginx
14    spec:
15      containers:
16        - name: nginx
17          image: nginx:1.21.0 # Фиксированная версия
18      ports:
19        - containerPort: 8080 # Кастомный порт
20      volumeMounts:
21        - name: nginx-config
22          mountPath: /etc/nginx/nginx.conf
23          subPath: nginx.conf

```

Рисунок 39

Задание 3:

- ВЫПОЛНИТЬ ВНУТРИ КОНТЕЙНЕРА команду ps aux (Рисунок 40)

```

kate@beady:~/nginx-postgres$ kubectl debug -it nginx-deployment-7cf544bb96-cpgpp --image=busybox:1.35 -- ps aux
--profile=legacy is deprecated and will be removed in the future. It is recommended to explicitly specify a profile, for example "--profile=general".
Defaulting debug container name to debugger-gztt2.
PID   USER     TIME   COMMAND
1  root      0:00   ps aux

```

Рисунок 40

- Просмотреть логи за 30 минут (Рисунок 41)

```

kate@beady:~/nginx-postgres$ kubectl logs --since=30m backend-deployment-84694467d-j58dh
INFO:      Started server process [1]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
INFO:      Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
INFO:      10.244.0.24:53890 - "GET /api/check HTTP/1.0" 200 OK
INFO:      10.244.0.24:53904 - "GET /api/version HTTP/1.0" 200 OK
INFO:      10.244.0.24:59368 - "GET /api/check HTTP/1.0" 200 OK
INFO:      10.244.0.24:44866 - "GET /api/check HTTP/1.0" 200 OK
INFO:      10.244.0.24:56594 - "GET /api/version HTTP/1.0" 200 OK

```

Рисунок 41

- Удалить под (Рисунок 42)

```

kate@beady:~/nginx-postgres$ kubectl delete pod debug
pod "debug" deleted

```

Рисунок 42

- Пробросить порт для отладки (Рисунок 43)

```

kate@beady:~/nginx-postgres$ kubectl port-forward backend-deployment-84694467d-j58dh 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080

```

Рисунок 43

Доп задание :

Создайте YAML для:

- ConfigMap с настройками для nginx (например, изменение стандартного порта) (Рисунок 44)

```
manifests > ! configmap.yaml
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: nginx-config
5  data:
6    nginx.conf: |
7      events {
8        worker_connections 1024;
9      }
10     http {
11       server {
12         listen 8080;
13         server_name localhost;
14
15         location / {
16           root /usr/share/nginx/html;
17           index index.html;
18         }
19       }
20     }
```

Рисунок 44

- Deployment, использующий ConfigMap (Рисунок 45)

```
manifests > ! nginx-deployment.yaml
5  spec:
10  template:
11    metadata:
12      labels:
13    spec:
14      containers:
15        - name: nginx
16          image: nginx:1.21.0 # Фиксированная версия
17          ports:
18            - containerPort: 8080 # Кастомный порт
19          volumeMounts:
20            - name: nginx-config
21              mountPath: /etc/nginx/nginx.conf
22              subPath: nginx.conf
23            - name: html
24              mountPath: /usr/share/nginx/html
25          volumes:
26            - name: nginx-config
27              configMap:
28                name: nginx-config
29            - name: html
30              configMap:
31                name: html-content
32            - name: nginx-content
33              configMap:
34                name: nginx-content
```

Рисунок 45

- Ingress, направляющий запросы по пути /site на сервис (Рисунок 46)

```
manifests > ! ingress.yaml
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: nginx-ingress
5    annotations:
6      nginx.ingress.kubernetes.io/rewrite-target: /$1
7  spec:
8    rules:
9      - http:
10        paths:
11          - path: /site(/|$)(.*)
12            pathType: Prefix
13          backend:
14            service:
15              name: nginx-service
16              port:
17                number: 80
```

Рисунок 46