Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
Высшего образования
Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа 2 по Облачным и туманным вычислениям

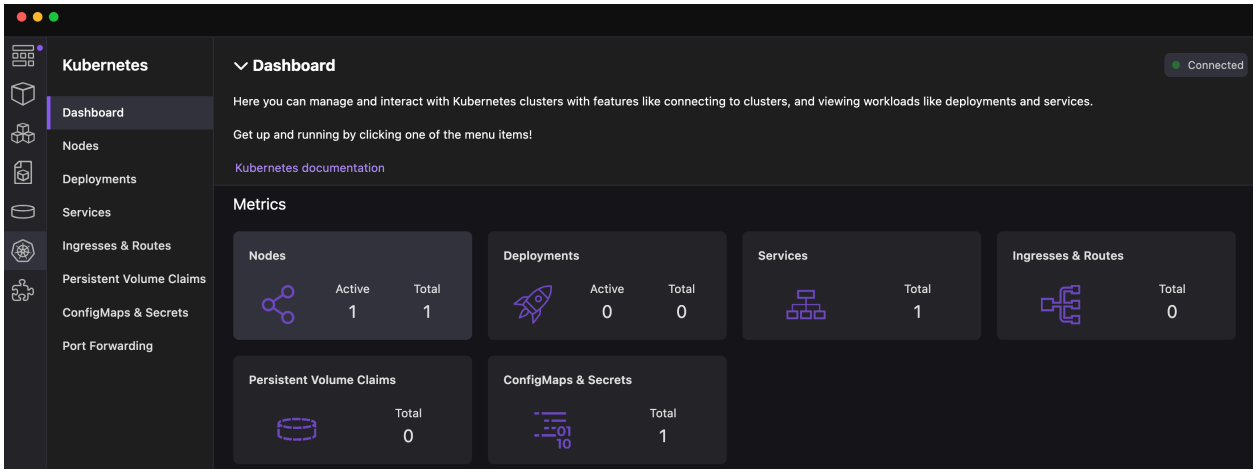
Группа: P3416
Выполнил:
Сиразетдинов А.Н.
Проверил:
Перл А.

Г. Санкт-Петербург
2026

Оглавление

Выполнение

Создадим kind кластер в podman




```
❯ kubectl get nodes

NAME                          STATUS   ROLES           AGE     VERSION
kind-cluster-control-plane    Ready    control-plane   2m29s   v1.35.0
```

Создадим deployment и service файлы для контейнеров

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: flask-counter:v1
          ports:
            - containerPort: 8080
          env:
            - name: REDIS_HOST
              value: redis
```

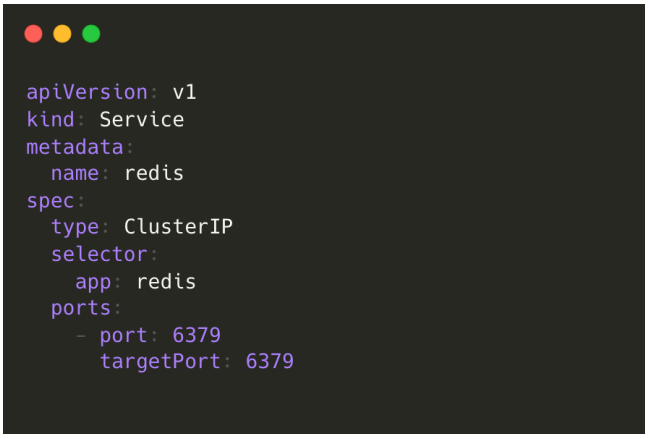
Рисунок 1 frontend-deployment.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: NodePort
  selector:
    app: frontend
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30080
```

Рисунок 2 frontend-service.yml

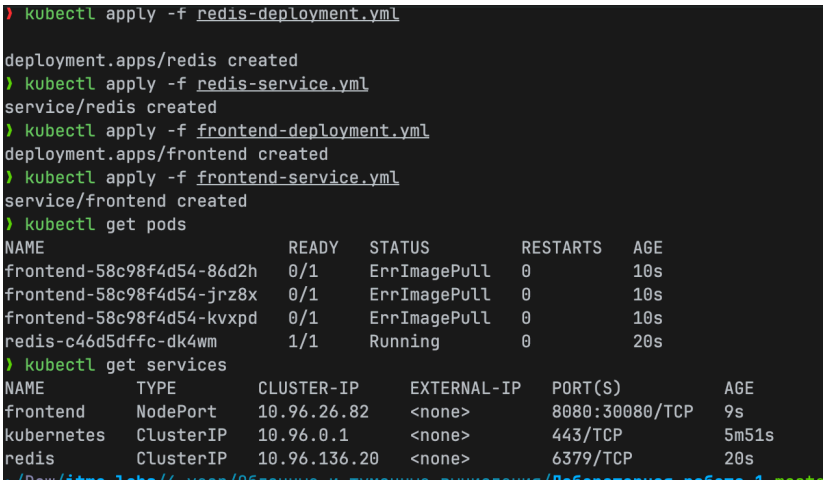
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
        - name: redis
          image: redis:7-alpine
          ports:
            - containerPort: 6379
```

Рисунок 3 redis-deployment.yml



```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis
spec:
  type: ClusterIP
  selector:
    app: redis
  ports:
    - port: 6379
      targetPort: 6379
```

Рисунок 4 redis-service.yml



```
❯ kubectl apply -f redis-deployment.yml

deployment.apps/redis created
❯ kubectl apply -f redis-service.yml
service/redis created
❯ kubectl apply -f frontend-deployment.yml
deployment.apps/frontend created
❯ kubectl apply -f frontend-service.yml
service/frontend created
❯ kubectl get pods
NAME                        READY   STATUS         RESTARTS   AGE
frontend-58c98f4d54-86d2h   0/1     ErrImagePull   0          10s
frontend-58c98f4d54-jrz8x   0/1     ErrImagePull   0          10s
frontend-58c98f4d54-kvxpd   0/1     ErrImagePull   0          10s
redis-c46d5dffc-dk4wm       1/1     Running        0          20s
❯ kubectl get services
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
frontend     NodePort    10.96.26.82    <none>        8080:30080/TCP   9s
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          5m51s
redis        ClusterIP   10.96.136.20   <none>        6379/TCP         20s
```

Соберем контейнер и загрузим его в kind

podman save flask-counter:v1 -o flask-counter-v1.tar

kind load image-archive flask-counter-v1.tar --name kind-cluster

Запустим поды

kubectl apply -f redis-deployment.yaml

kubectl apply -f redis-service.yaml

kubectl apply -f frontend-deployment.yaml

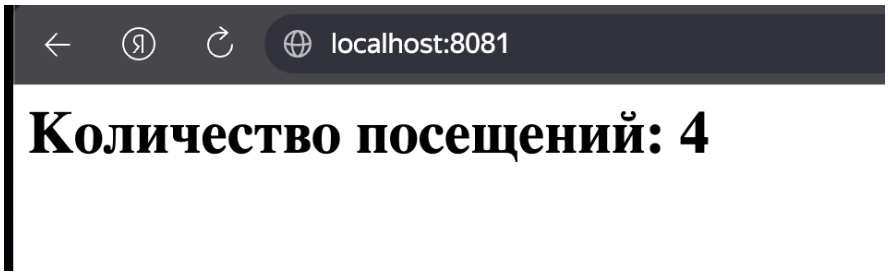
kubectl apply -f frontend-service.yaml

```
❯ kubectl get pods

NAME                        READY   STATUS    RESTARTS   AGE
frontend-b67466b89-9mvtb    1/1     Running   0          111s
frontend-b67466b89-cbsbn    1/1     Running   0          110s
frontend-b67466b89-wtkt4    1/1     Running   0          109s
redis-c46d5dffc-dk4wm       1/1     Running   0          11m
```

Пробросим порты

```
❯ kubectl port-forward service/frontend 8081:8080
Forwarding from 127.0.0.1:8081 -> 8080
Forwarding from [::1]:8081 -> 8080
Handling connection for 8081
Handling connection for 8081
Handling connection for 8081
Handling connection for 8081
Handling connection for 8081
Handling connection for 8081
```



Rolling-update

Изменим цвет фона

```python
from flask import Flask
import redis
import os

app = Flask(__name__)

redis_host = os.getenv("REDIS_HOST", "localhost")
r = redis.Redis(host=redis_host, port=6379, decode_responses=True)

@app.route("/")
def hello():
    count = r.incr("hits")
    return f"<h1 style='background-color: lightgreen'>Hits: {count}</h1>"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

Пересоберем контейнер, загрузим новую версию в kind и обновим версию через kubectl

```
STEP 4/7: RUN pip install --no-cache-dir -r requirements.txt
--> Using cache 45954689c19a98da992233a5853fe97d064bc8423b0687f20ddb744de14e144a
--> 45954689c19a
STEP 5/7: COPY app.py .
--> Using cache bdc95336b2947d719cbb48397e199ee81e8a8746c81f85797105306cbdc57d17
--> bdc95336b294
STEP 6/7: EXPOSE 5000
--> Using cache 5df76f3d9f1ffada9082b059bcc06d1a9d968836f2eb4c94617989230a21565f
--> 5df76f3d9f1f
STEP 7/7: CMD ["python", "app.py"]
--> Using cache 1e92481c4d23e2280b95c7cca23c8f991c6b35a22fd10a7ea40e7100069bdd16
COMMIT flask-counter:v2
--> 1e92481c4d23
Successfully tagged localhost/flask-counter:v2
1e92481c4d23e2280b95c7cca23c8f991c6b35a22fd10a7ea40e7100069bdd16
❯ podman save flask-counter:v2 -o flask-counter-v2.tar

❯ kind load image-archive flask-counter-v2.tar --name kind-cluster

❯ kubectl rollout restart deployment/frontend

deployment.apps/frontend restarted
```

```
❯ kubectl set image deployment/frontend frontend=localhost/flask-counter:v2

deployment.apps/frontend image updated
❯ kubectl rollout status deployment/frontend

deployment "frontend" successfully rolled out
```

Вывод

В лабораторной работе я познакомился с архитектурой Kubernetes, создал свои поды и попробовал использовать rolling-release