

Федеральное государственное автономное образовательное
учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2
по дисциплине
«Облачные и туманные вычисления»

Выполнил студент группы Р3415
Барсуков Максим Андреевич

Преподаватель:
Перл Ольга Вячеславовна

Санкт-Петербург
2025 г.

Содержание

Задание.....	2
Ход работы.....	3
Подготовка окружения.....	3
Создание манифестов Kubernetes.....	4
Конфигурация.....	4
Redis.....	5
Python.....	6
Развертывание (Deployment).....	7
Проверка работы.....	8
Rolling Update.....	9
Заключение.....	11

Задание

Цель: Перенести нагрузку из Лабораторной №1 в Kubernetes. Так как мы избегаем OpenShift, используйте Minikube или Kind (Kubernetes in Docker). Это научит работать с примитивами (Pods, Deployments, Services), которые OpenShift обычно скрывает за абстракциями.

Инструменты: Minikube (или Kind), kubectl.

Ход работы:

- 1. Трансляция (Translation):** Вы должны вручную "перевести" логику Docker Compose в манифесты K8s:
 - deployment.yaml для приложения Python (определение образа и реплик).
 - deployment.yaml для Redis.
 - service.yaml (тип ClusterIP) для внутреннего доступа к Redis.
 - service.yaml (тип NodePort или LoadBalancer) для внешнего доступа к приложению.
- 2. Развертывание:** Применить манифесты к локальному кластеру (kubectl apply).
- 3. Rolling Update (Обновление без простоя):** Вы меняете код Python (например, цвет фона), собирают образ версии v2, обновляют YAML-файл и наблюдают, как Kubernetes выполняет постепенное обновление подов без остановки сервиса.

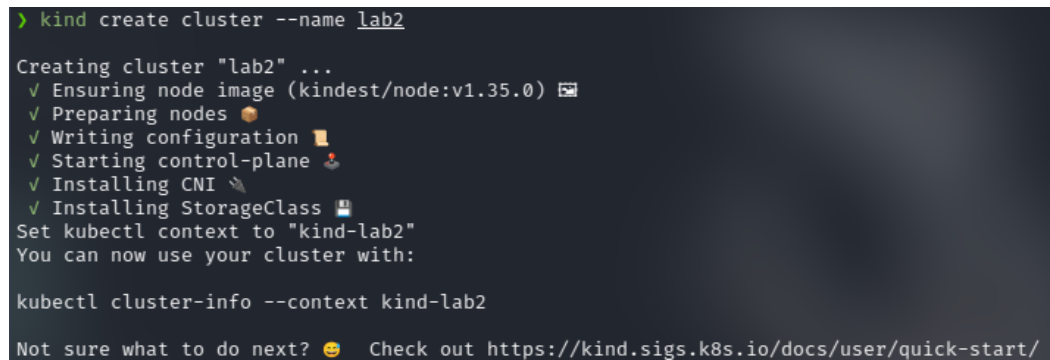
Ключевые знания: Жизненный цикл Pod'a, разница между императивным и декларативным подходом, внутренняя сеть кластера.

Ход работы

Подготовка окружения

Создание кластера показано на рисунке 1:

```
kind create cluster --name lab2
```

A screenshot of a terminal window showing the command 'kind create cluster --name lab2' and its output. The output indicates the successful creation of the 'lab2' cluster, including steps like ensuring the node image, preparing nodes, writing configuration, starting the control plane, and installing CNI and StorageClass. It also provides instructions on how to use the cluster with 'kubectl' and a link to the kind project documentation.

```
> kind create cluster --name lab2
Creating cluster "lab2" ...
✓ Ensuring node image (kindest/node:v1.35.0) 📁
✓ Preparing nodes 📦
✓ Writing configuration 📄
✓ Starting control-plane 🚦
✓ Installing CNI 🛠️
✓ Installing StorageClass 🗄️
Set kubectl context to "kind-lab2"
You can now use your cluster with:

kubectl cluster-info --context kind-lab2

Not sure what to do next? 😊 Check out https://kind.sigs.k8s.io/docs/user/quick-start/
```

Рисунок 1 – Создание кластера kind

Создание манифестов Kubernetes

Конфигурация

Конфигурация (k8s/config.yaml):

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secrets
type: Opaque
stringData:
  REDIS_PASSWORD: password
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  REDIS_HOST: redis
  REDIS_PORT: "${REDIS_PORT}"
  REDIS_DB: "0"
```

Redis

Deployment для Redis (k8s/redis-deployment.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
          envFrom:
            - secretRef:
                name: app-secrets
          command: ["sh", "-c", "redis-server --appendonly yes --requirepass $REDIS_PASSWORD"]
          ports:
            - containerPort: ${REDIS_PORT}
```

Service для Redis (k8s/redis-service.yaml):

```
apiVersion: v1
kind: Service
metadata:
  name: redis
spec:
  type: ClusterIP
  selector:
    app: redis
  ports:
    - port: ${REDIS_PORT}
      targetPort: ${REDIS_PORT}
```

Python

Deployment для приложения (k8s/app-deployment.yaml):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: python-app
spec:
  replicas: ${APP_REPLICAS}
  selector:
    matchLabels:
      app: python-app
  template:
    metadata:
      labels:
        app: python-app
    spec:
      containers:
        - name: python-app
          image: ${APP_IMAGE}
          imagePullPolicy: Never
          ports:
            - containerPort: ${APP_PORT}
          envFrom:
            - configMapRef:
                name: app-config
            - secretRef:
                name: app-secrets
```

Service для приложения (k8s/app-service.yaml):

```
apiVersion: v1
kind: Service
metadata:
  name: python-app-service
spec:
  type: NodePort
  selector:
    app: python-app
  ports:
    - port: ${APP_PORT}
      targetPort: ${APP_PORT}
      nodePort: ${APP_NODE_PORT}
```

Развертывание (Deployment)

Сборка Docker-образа приложения и загрузка его в кластер Kind:

```
docker build -t lab2-app:v1 .
kind load docker-image lab2-app:v1 --name lab2
```



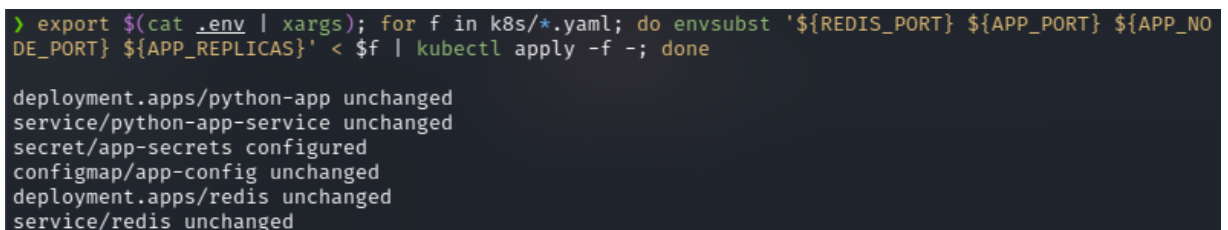
```
> docker build -t lab2-app:v1 .
[+] Building 0.4s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 372B
=> [internal] load metadata for docker.io/library/python:3.11-slim
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/python:3.11-slim@sha256:0b2cfd528ad7c43ba83de95e2788393cf545aae1ed738868e961454bd38a760a
=> [internal] load build context
=> => transferring context: 94B
=> CACHED [2/5] WORKDIR /app
=> CACHED [3/5] COPY app/requirements.txt ./
=> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt
=> CACHED [5/5] COPY app/ ./
=> exporting to image
=> => exporting layers
=> => writing image sha256:3eb18a94c8af5cdc17900f232308acf20cd79c8c1529beefbaf701d6b21406f5
=> => naming to docker.io/library/lab2-app:v1
>
> kind load docker-image lab2-app:v1 --name lab2
Image: "lab2-app:v1" with ID "sha256:3eb18a94c8af5cdc17900f232308acf20cd79c8c1529beefbaf701d6b21406f5" not yet present on node "lab2-control-plane", loading...
>
```

Рисунок 2 – Сборка Docker-образа приложения и загрузка его в кластер Kind

Применение манифестов с подстановкой переменных из .env:

```
REDIS_PASSWORD=password
REDIS_HOST=redis
REDIS_PORT=6379
REDIS_DB=0
APP_PORT=5000
APP_NODE_PORT=31010
APP_REPLICAS=3
APP_IMAGE=lab2-app

export $(cat .env | xargs); for f in k8s/*.yaml; do envsubst
'${REDIS_PORT} ${APP_PORT} ${APP_NODE_PORT} ${APP_REPLICAS}' < $f |
kubectl apply -f -; done
```



```
> export $(cat .env | xargs); for f in k8s/*.yaml; do envsubst '${REDIS_PORT} ${APP_PORT} ${APP_NO
DE_PORT} ${APP_REPLICAS}' < $f | kubectl apply -f -; done

deployment.apps/python-app unchanged
service/python-app-service unchanged
secret/app-secrets configured
configmap/app-config unchanged
deployment.apps/redis unchanged
service/redis unchanged
```

Рисунок 3 – Применение манифестов

Проверка статуса подов:

```
kubectl get pods
```

```
> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
python-app-56478ffd58-8kzrl        1/1     Running   0           5m2s
python-app-56478ffd58-q97h9        1/1     Running   0           5m2s
python-app-56478ffd58-rjrhv        1/1     Running   0           5m2s
redis-84b86b57bb-g85nr             1/1     Running   0           8m17s
>
```

Рисунок 4 – Применение манифестов

Проверка работы

Для доступа к приложению с хост-машины использовался проброс портов, так как Kind запускает ноды в Docker-контейнерах

```
kubectl port-forward service/python-app-service 9000:5000
```

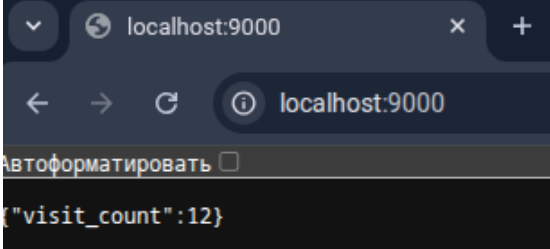
```
> kubectl port-forward service/python-app-service 9000:5000
Forwarding from 127.0.0.1:9000 -> 5000
Forwarding from [::1]:9000 -> 5000
Handling connection for 9000
Handling connection for 9000
Handling connection for 9000
Handling connection for 9000
```

Рисунок 5 – Проброс портов

Проверка через curl:

```
curl http://localhost:9000
```

```
> curl http://localhost:9000
{"visit_count":1}
> curl http://localhost:9000
{"visit_count":2}
> curl http://localhost:9000
{"visit_count":3}
> curl http://localhost:9000
{"visit_count":4}
```



```
localhost:9000
{"visit_count":12}
```

Рисунок 6 – Проверка доступности

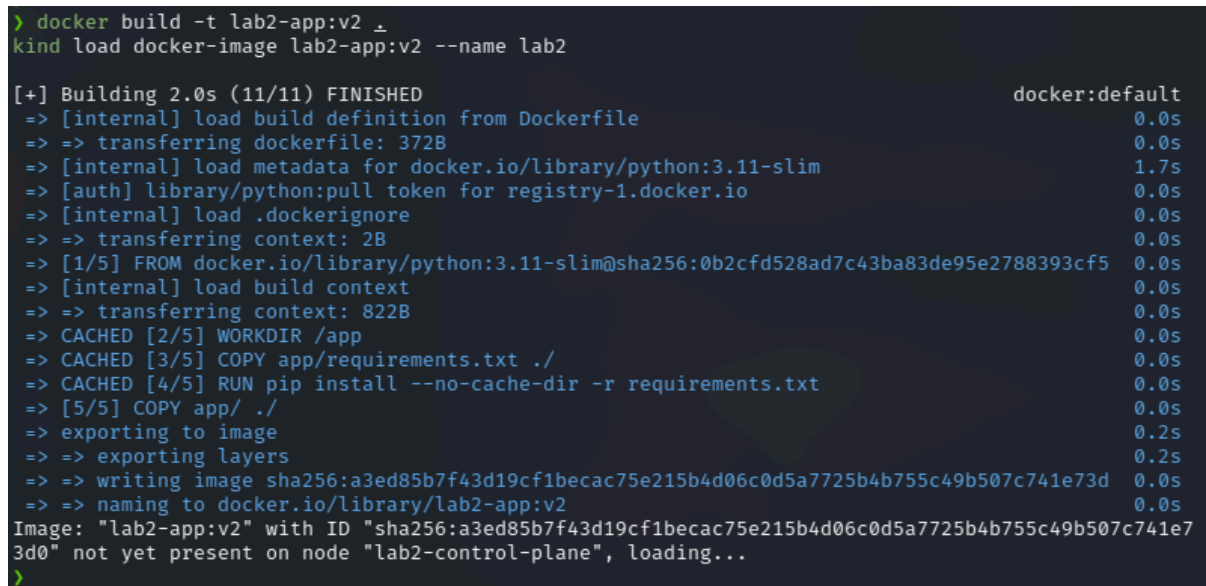
Rolling Update

Для демонстрации обновления без простоя был изменен код приложения (добавлена пометка v2 в вывод сообщения счетчика):

```
return jsonify(visit_count=count, message="Hello"), 200
```

Сборка новой версии образа:

```
docker build -t lab2-app:v2 .
kind load docker-image lab2-app:v2 --name lab2
```



```
> docker build -t lab2-app:v2 .
kind load docker-image lab2-app:v2 --name lab2

[+] Building 2.0s (11/11) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 372B                                0.0s
=> [internal] load metadata for docker.io/library/python:3.11-slim 1.7s
=> [auth] library/python:pull token for registry-1.docker.io       0.0s
=> [internal] load .dockerignore                                   0.0s
=> => transferring context: 2B                                       0.0s
=> [1/5] FROM docker.io/library/python:3.11-slim@sha256:0b2cfd528ad7c43ba83de95e2788393cf5 0.0s
=> [internal] load build context                                   0.0s
=> => transferring context: 822B                                       0.0s
=> CACHED [2/5] WORKDIR /app                                       0.0s
=> CACHED [3/5] COPY app/requirements.txt ./                        0.0s
=> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt 0.0s
=> [5/5] COPY app/ ./                                              0.0s
=> exporting to image                                              0.2s
=> => exporting layers                                              0.2s
=> => writing image sha256:a3ed85b7f43d19cf1becac75e215b4d06c0d5a7725b4b755c49b507c741e73d 0.0s
=> => naming to docker.io/library/lab2-app:v2                      0.0s
Image: "lab2-app:v2" with ID "sha256:a3ed85b7f43d19cf1becac75e215b4d06c0d5a7725b4b755c49b507c741e73d0" not yet present on node "lab2-control-plane", loading...
>
```

Рисунок 7 – Сборка новой версии образа

Обновление версии образа в .env или манифесте и повторное применение конфигурации.

```
export $(cat .env | xargs);
export APP_IMAGE=lab2-app:v2

for f in k8s/*.yaml; do envsubst '${REDIS_PORT} ${APP_PORT}
${APP_NODE_PORT} ${APP_REPLICAS} ${APP_IMAGE}' < $f | kubectl apply -f
-; done
```

Наблюдение за процессом обновления:

```
kubectl rollout status deployment/python-app
```

Waiting for deployment "python-app" rollout to finish: 1 out of 2 new replicas have been updated
Waiting for deployment "python-app" rollout to finish: 1 old replicas are pending termination
deployment "python-app" successfully rolled out

```
> kubectl rollout status deployment/python-app
Waiting for deployment "python-app" rollout to finish: 1 out of 3 new replicas have been updated...
```

Рисунок 8 – Наблюдение за процессом обновления

Проверка новой версии:

```
curl http://localhost:9000
```

```
> kubectl rollout status deployment/python-app
deployment "python-app" successfully rolled out
>
> curl http://localhost:9000
{"visit_count":7}
> curl http://localhost:9000
{"visit_count":8}
> curl http://localhost:9000
{"message":"Hello","visit_count":9}
> curl http://localhost:9000
{"message":"Hello","visit_count":10}
>
```

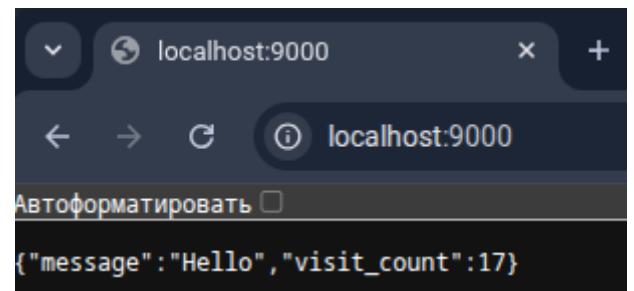


Рисунок 9 – Проверка новой версии

Приложение успешно обновилось, счетчик посещений сохранился (так как Redis вынесен в отдельный сервис), и сервис оставался доступным во время обновления.

Заключение

В рамках лабораторной работы была осуществлена миграция веб-приложения из изолированной среды Docker Compose в оркестрируемый кластер Kubernetes. Практически были освоены и применены ключевые объекты (примитивы) Kubernetes: Pods (поды), Deployments (деплойменты) и Services (сервисы). Особое внимание было уделено стратегии Rolling Update (постепенное обновление), которая была успешно опробована для обновления версии приложения без прерывания его доступности. Работа с декларативными YAML-манифестами наглядно продемонстрировала их преимущества в управлении целевым состоянием инфраструктуры перед императивным подходом к запуску контейнеров.