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 P3415
Барсуков Максим Андреевич

Преподаватель:
Перл Ольга Вячеславовна

Санкт-Петербург
2025 г.

Содержание

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
```

Рисунок 1 – Создание кластера kind

Создание манифестов Kubernetes

Конфигурация

Конфигурация (`k8s/config.yaml`):

```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
```

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
```

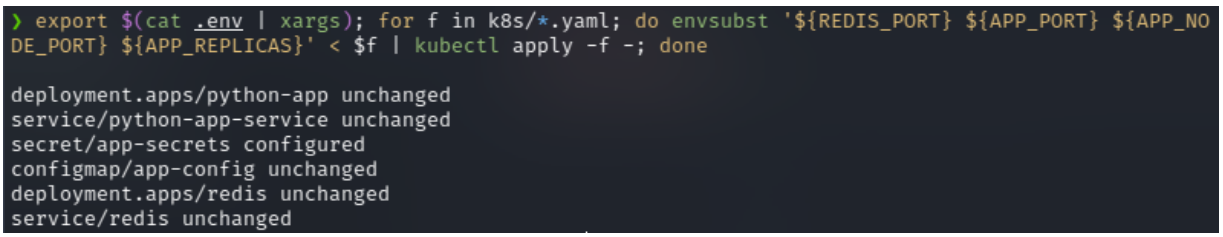


Рисунок 3 – Применение манифестов

Проверка статуса подов:

```
kubectl get pods
```

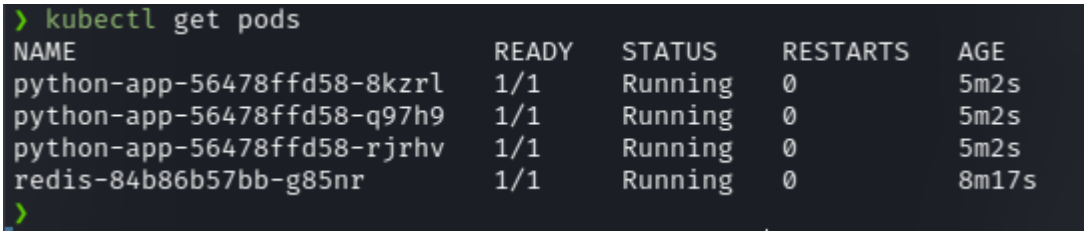


Рисунок 4 – Применение манифестов

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
```

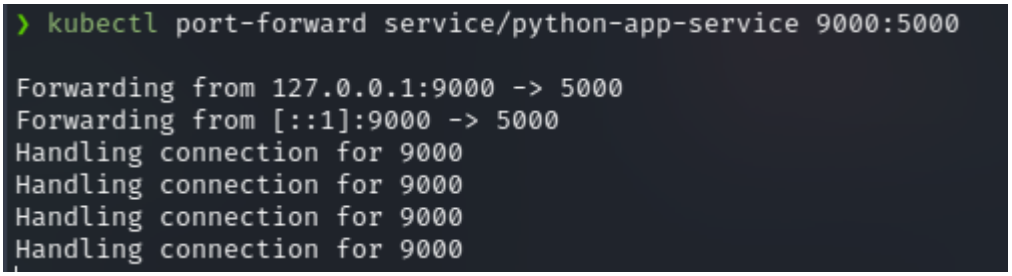


Рисунок 5 – Проброс портов

Проверка через curl:

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

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
```

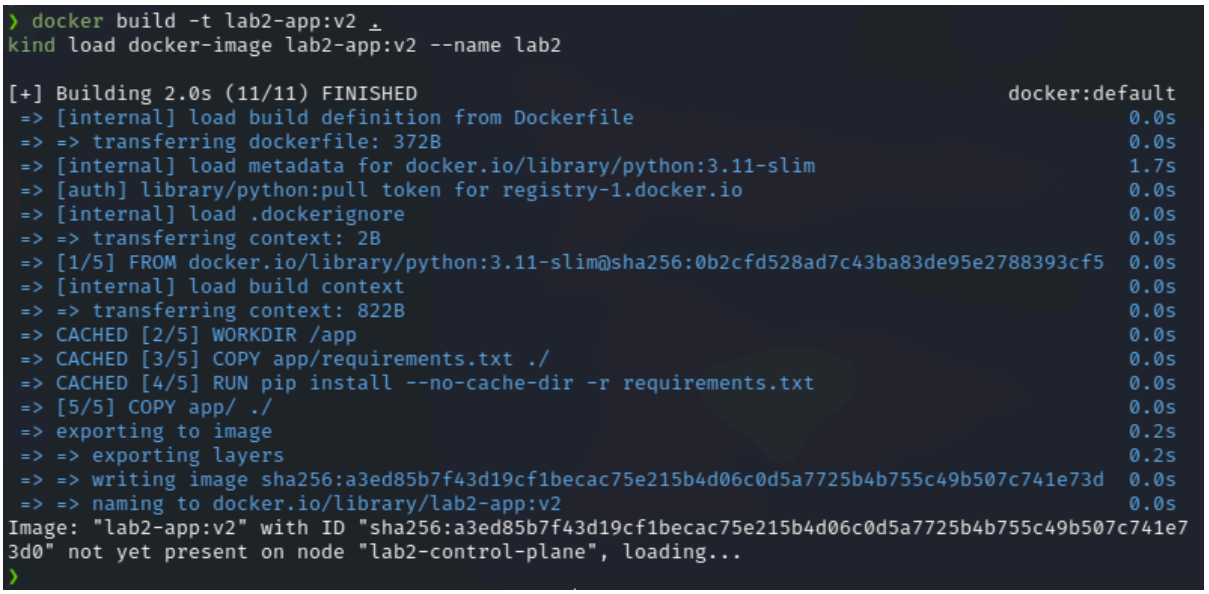


Рисунок 7 – Сборка новой версии образа

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 Waiting for deployment "python-app" rollout to finish: 1 old replicas are
pending termin deployment "python-app" successfully rolled out
```

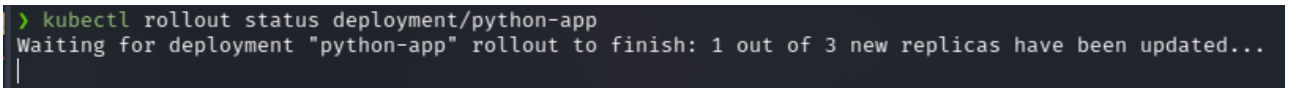
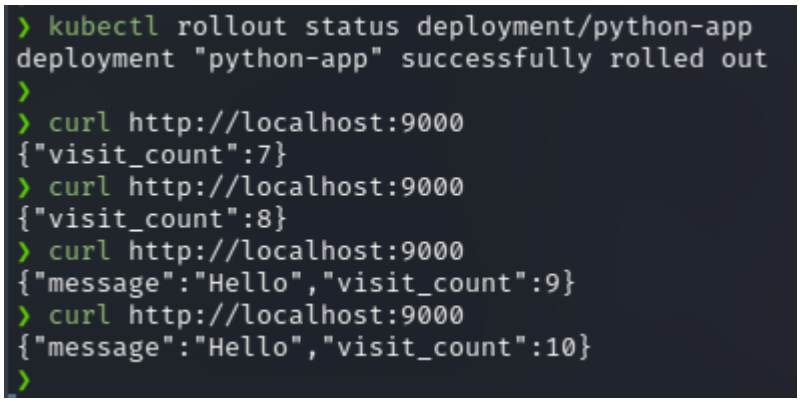


Рисунок 8 – Наблюдение за процессом обновления

Проверка новой версии:

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



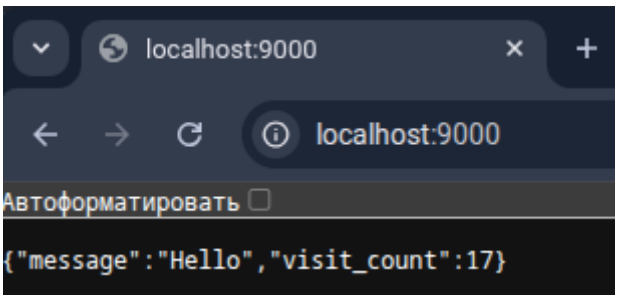


Рисунок 9 – Проверка новой версии

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.