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

ДИСЦИПЛИНА:

Проектный практикум по разработке ETL-решений

**Лабораторная работа 3.2**

**Развертывание приложения в Kubernetes**

Выполнила: Сергеева А. И., группа: АДЭУ-211

Преподаватель: Босенко Т.М.

Москва

2025

**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:**

В рамках 12-го варианта необходимо было развернуть Django-приложение в Kubernetes, используя PostgreSQL для базы данных, Nginx для раздачи статических файлов и проксирования запросов, Deployment и Service.

В папке 3\_2, посвящённой лабораторной работе уже созданы pg\_configmap.yml и postgres-secrets.yml, которые создают ConfigMap и Secret, которые нужны для настройки и запуска PostgreSQL. Также уже есть pg\_service.yml, создающий сервис, который предоставляет доступ к базе данных PostgreSQL через сеть Kubernetes, и pg\_deployment.yml, создающий деплой PostgreSQL. Содержимое уже готовых файлов представлено на рисунках 1-4.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1 – Содержимое pg\_configmap.yml

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2 – Содержимое pg\_deployment.yml

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3 – Содержимое pg\_service.yml

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4 – Содержимое postgres-secrets.yml

Опираясь на них, был создан deployment.yaml на рисунке 5. В нём определяется создание pod django-postgres-nginx. В initContainers, указано, что перед запуском основного контейнера необходимо ждать, пока PostgreSQL не станет доступным на порту 5432. Из образа python:3.9 берётся контейнер для django, для этого устанавливаются библиотеки Django, PostgreSQL и Gunicorn, создаётся новый проект Django с именем myproject, конфигурируется база данных PostgreSQL путём добавления параметров подключения. Также важно было установить список разрешённых хостов для Django, чтобы приложение могло обрабатывать запросы с этих адресов. Была создана директория для статических файлов, выполнена миграция базы данных, а также настроен сбор статических файлов Django. На порте 8000 будет слушаться контейнер с Django. Контейнер Nginx использует образ nginx:alpine с прослушиванием на порту 80. Далее был создан конфигурационный  файл для Nginx, который предоставляет настройку для проксирования запросов к Django и обслуживанию статических файлов. Следующим этапом создаётся контейнер для PostgreSQL, используя образ postgres:alpine. Здесь используются конфигурации из ConfigMap и секретов для настройки базы данных. И в конце определяется том static-volume, который будет использоваться для хранения статических файлов. В результате этот YAML-файл создает развертывание приложения, состоящее из нескольких контейнеров: контейнера с Django-приложением, проксирующего его через Nginx, и контейнера с PostgreSQL для управления базами данных.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5 – Содержимое deployment.yaml

Далее был создан Service, который используется для управления доступом к подам приложения на рисунке 6. django-service будет слушать входящие запросы на порту 80, трафик переправляется также на 80.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6 – Содержимое Service

Далее необходимо было применить конфигурации pg\_configmap.yml, postgres-secrets.yml, pg\_service.yml, pg\_deployment.yml, deployment.yaml, service.yaml, как показано на рисунке 7.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 7 – Применение конфигураций

Далее была выполнена проверка подов, чтобы узнать их состояние, и запущено само приложение на рисунках 8–9. Поды все в состоянии «Running». Судя по тексту на странице, вход выполнен успешно, но не настроены URL-адреса, хотя такая задача и не стояла.

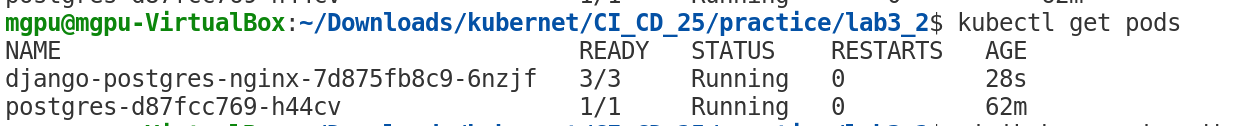


Рисунок 8 – Вывод всех подов

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 9 – Запуск Django

**Общий вывод:**

В ходе работы были созданы манифесты deployment.yaml и service.yaml и применены через kubectl apply. После развертывания Deployment и создания Service типа LoadBalancer удалось проверить доступность приложения через внешний IP. Успешно развернуто многокомпонентное приложение (Django + PostgreSQL + Nginx) в одном Deployment. По рекомендациям необходимо разносить разные сервисы по разным Deployment, но было принято решение всё делать в одном, т. к. в самом задании требовалось предоставить 1, а также это упрощает конфигурацию и развёртывание. Готовые файлы для PostgreSQL значительно упростили развёртывание. Цель и задачи были выполнены.

**Ответы на контрольные вопросы:**

1. Что такое Pod, Deployment и Service в Kubernetes? **Ответ:** Pod - наименьшая и самая простая единица в Kubernetes. Группирует один или несколько контейнеров (например, Django + Nginx). Deployment - объект, управляющий жизненным циклом Pod (создание, обновление, масштабирование). Запускает и поддерживает заданное количество реплик Pod, автоматически восстанавливает Pod при сбоях. Service - это абстракция, определяющая логический набор подов и политику доступа к ним (иногда такой набор подов еще называют микросервисом).

2. Каково назначение Deployment в Kubernetes? **Ответ:** автоматическое управление созданием подов. Deployment говорит кластеру Kubernetes, сколько копий приложения нужно запустить, следить за ним, обновлять при необходимости и восстанавливать в случае сбоя, можно добавлять и удалять реплики. Deployment позволяет переходить от одной версии приложения к другой без прерывания работы системы. Если во время обновления произошла ошибка, можно быстро вернуться к предыдущей, рабочей версии приложения. Самовосстановление при сбое подов. Если приложение перестало отвечать, Kubernetes перезапустит Pod автоматически.

3. Каково назначение Service в Kubernetes**? Ответ:** Единая точка доступа к группе Pod, постоянный IP и DNS – даже если Pod пересоздается, Service сохраняет адрес, публикация приложения – через NodePort или LoadBalancer.

4. Как создать Deployment в Kubernetes? **Ответ:** Создать файл, например, deployment.yaml, написать манифест, который определяет ресурс Deployment в Kubernetes и в конце применить конфигурацию командой: kubectl apply -f deployment.yaml. Также можно создать через командную строку kubectl. Пример: kubectl create deployment my-app --image=nginx:latest --replicas=3 [3](https://signoz.io/guides/kubernetes-deployments/).

5. Как создать Service в Kubernetes и какие типы Services существуют? Ответ: Создание Service в Kubernetes происходит с помощью файла YAML, в котором указывается конфигурация сервиса. Например, можно создать сервис, который перенаправляет запросы на порт 9376 каждого пода, у которого есть метка app=MyApp. Типы Services в Kubernetes включают:

- ClusterIP. Предоставляет доступ к сервису на внутреннем IP-адресе кластера (сервис доступен только внутри кластера). Тип ClusterIP используется по умолчанию.

- NodePort. Предоставляет доступ к сервису на IP-адресе каждого узла (ноды) кластера, на статическом порту (из диапазона 30000–32767). Автоматически создаётся и сервис типа ClusterIP, на который будут маршрутизироваться запросы с NodePort.

- LoadBalancer. Предоставляет доступ к сервису, используя балансировщик (load balancer) облачного провайдера. При этом автоматически создаются сервисы типа NodePort и ClusterIP, на которые будут маршрутизироваться запросы с балансировщика.

- ExternalName. Сопоставляет имя сервиса с содержимым поля externalName (например, foo.bar.example.com), возвращая CNAME-запись.