ЛАБОРАТОРНА РОБОТА № 4

Створення і розгортання програмної інфраструктури на основі docker-compose

Мета роботи: полягає у дослідженні процесу автоматичного розгортання відносно складної програмної інфраструктури розподіленого веб-застосунку за обраним напрямом технології. Зважаючи на те, що сучасні РПС являють собою систему програмних модулів, що взаємодіють між собою і реалізовані на різних технологіях, їх автоматичне розгортання потребує додаткових програмних механізмів, що спрощують процес розгортання і розробки.

Одним з таких механізмів є docker-compose. У цій ЛР відбувається розгортання РПС з різних модулів відповідно до наданого завдання, вивчення синтаксисту файлів docker-compose і тестування отриманих результатів.

Лабораторну роботу можна умовно розподілити на три частини:

- вивчення і тестування складових частин відносно складної РПС, дозволяється використовувати власні напрацювання і досвід роботи за фахом студентів, що навчаються;
- підготовка первинного файлу docker-compose, вивчення елементів синтаксису і формування стилю цього файлу;
- розгортання РПС з використанням отриманого docker-compose, тестування роботи складної програмної системи, виправлення помилок, що були виявлені.

Вхідні дані ЛР4

У якості вхідних даних для ЛР4 є:

- типова база даних, що містить відповідний стандартний контейнер, який може бути активований з docker-compose; для повного завдання (на 20 балів) має бути 2 бази даних (SQL та NoSQL), які містяться у відповідних контейнерах;
- два або три контейнера з базовим веб-застосунком, що побудовані на основі типового веб-фреймворку і взаємодіють з базами даних.

Вихідні дані ЛР4

У якості вихідних даних для ЛР4 є: система каталогів з файлом docker-compose.yml, звіт.

Завдання

1. У якості індивідуального завдання, на першому етапі слід вивчити переваги і недоліки баз даних (є надані у переліку варіантів, але ж дозволяються і власні варіанти), за необхідності побудувати файли docker-compose для баз

даних і протестувати роботу цього файлу на реальній системі, використовуючи доступні засоби тестування БД.

Вивчити веб-застосунок, що побудований на основі фреймворку з наданих варіантів, модифікувати його для роботи з вашими БД, або одразу працювати з обраним самостійно варіантом. При необхідності провести його дослідження з docker-compose, аналогічно БД.

2. Підготувати файл docker-compose, що дозволяє побудувати узагальнену систему з парою контейнерів з БД і двома-трьома контейнерами з веб-застосунку (для front/back end), які взаємодіють відповідно до функціоналу з БД.

3. Провести розгортання РПС з використанням отриманого docker-compose. Продумати послідовність запуску контейнерів. Після розгортання провести тестування роботи складної програмної системи, що запущена з використанням результатів п.2. У випадку наявності, провести виправлення помилок. При неможливості поєднати БД і фреймворк веб-застосунку, провести аналіз помилок та обрати іншу БД або фреймворк.

Стек технологій, обирається студентами за власними вподобаннями.

Для SQL бази даних можна обрати postgres:14.1-alpine, mysql або іншу.

Для NoSQL БД (*повне завдання*) гарно підійде MongoDB, але вибір теж не обмежений.

У якості фреймворку можна обрати Django або Spring, хоча можна створити застосунок і на свій смак.

Варіанти завдання

Легке (на 10 балів)

Для розподіленого веб-застосунку створити yml файл та налаштувати його для коректного розгортання окремих контейнерів з базою даних, бекендом та фронтендом. Запустити контейнерну систему.

Повне

Можна вибрати з наявних або придумати завдання самостійно, але головним є наявність декількох таблиць (можна навіть обійтись лише двома, але обов’язково використати SQL та NoSQL БД). Створити додаток з власним REST-full API за допомогою якого мають виконуватись CRUD операції:

Варіант №1. Запити агентства з реклами.

Таблиці: Клієнти (Код, ПІБ, Вік, Стать + можливо ще щось) – для NoSQL. Співробітники (Код, ПІБ, Вік, Стать). Співробітники і посади (Код, посади, Оклад, Обов’язки, Вимоги).

Відділ кадрів (Зв’язує таблиці «Співробітники» і «Співробітники і посади» по полю «Код посади»).

Фільтри: Фільтр для відображення співробітників окремих посад (на основі запиту «Відділ кадрів»); клієнтів.

Варіант №2. Фірма з продажу персональних комп'ютерів.

Таблиці: Клієнти (Код, ПІБ, Вік, Стать, тип, вартість). Товар (Тип персональних комп'ютерів, вартість). Продавці (Код, ПІБ, Вік, Стать + можливо ще щось) – для NoSQL.

Замовлення (Зв'язує таблиці «Клієнти» і «Товар» у контексті замовлення).

Фільтр: Фільтр для відображення клієнтів та замовлень; продавців

Варіант №3. Продаж автомобілів

Таблиці: Клієнти (Код, ПІБ, Вік, Стать, тип автомобілю, вартість). Автомобілі (Код, тип автомобілю, вартість, пробіг, технічний стан). Продавці (Код, ПІБ, Вік, Стать + можливо ще щось) – для NoSQL.

Замовлення (Зв'язує таблиці «Клієнти» і «Автомобілі» у контексті замовлення).

Фільтр: Фільтр для відображення клієнтів, замовлень; продавців.

Варіант №4. Кінотеатр

Таблиці: Глядач (Код, ПІБ, Вік, Стать, замовлення). Квиток (Код, дата, номер місця, час, назва фільму). Продавці (Код, ПІБ, Вік, Стать + можливо ще щось) – для NoSQL.

Замовлення (Зв'язує Таблиці «Глядач» і «Квиток» у контексті замовлення).

Фільтр: Фільтр для відображення замовлень по певним датам; продавців.

Програма проведення експерименту ЛР4

Залежно від обраних технологій та завдання хід проведення експерименту може відрізнятись, тому орієнтовна програма наступна.

1. Перший етап.

Перед початком досліджень перевіряємо працездатність Docker compose для обраного типу операційної системи. Для деяких ОС засіб Docker compose постачається за замовчуванням разом з Docker, для деяких ОС потрібно його додатково інсталювати. При необхідності провести додаткову інсталяцію з використанням стандартних засобів, для обраної операційної системи. Створюємо робочий каталог проекту. У робочий каталог проекту розміщуємо файл docker-compose.yml.

2. Другий етап.

Користуючись мануалом [4 д.], створюємо два мінімальних проекти Django у робочому каталозі проекту (РКП) разом з веб-застосунком на базі типового мануала. Достатньо виконати простий проект на рівні двох частин мануалу [4 д.] «Writing your first Django app, part 1», «Writing your first Django app, part 2».

Результатом мають бути два внутрішніх каталоги у РКП з проектами Django. Провести тестування двох проектів на працездатність. Як це зробити показано у мануалі [4 д.] частина 1.

Користуючись мануалом [4 д.] переналаштувати отримані контейнери на роботу з базами даних.

3. Третій етап.

Для кожного проекту у РКП слід додати Dockerfile з відповідними налаштуваннями, з урахуванням БД. Виконати всі потрібні дії для підготовки системи контейнерів.

4. Четвертий етап.

Відповідно до функціоналу слід внести зміни до файлу docker-compose.yml. Передбачено, що будуть працювати чотири контейнера. Два контейнера з проектом Django і два контейнера з базами даних. Після внесення змін до файлу docker-compose.yml перевіряємо результат docker-compose build. При успішній побудові доступним є тест двох проектів Django. У випадку помилок, внести відповідні зміни до файлів docker-compose.yml і провести додаткове тестування.

5. П’ятий етап.

Далі налаштовуємо БД. Для цього вносимо відповідні зміни до файлу docker-compose.yml. Оновлюємо налаштування у двох проектах Django налаштовуємо додаткові елементи. Після внесених змін проводимо тестування результатів. Після отримання результату, потрібно зафіксувати його у вигляді скриншоту. Результат додати до звіту.

6. Сформувати звіт.

Теоретичні відомості ЛР4

На теперішній час активно розвивається технологія Docker і споріднена з нею технологія Docker compose. Використання технології Docker compose доцільно у виробничому та перед-виробничому (staging) процесі, у ході тестування і підтримки, процесах безперервної інтеграції і створення мікросервісних архітектурних рішень.

Для роботи зі складними РПС, що побудовані на основі СК Docker використовується система Docker Compose. За допомогою цього інструменту відбувається автоматизація процесу розгортання сервісів та заданої конфігурації. Зручність цієї системи полягає у тому, що всі елементи можуть бути створені та запущені однією командою з терміналу (консолі).

Доступні команди управління життєвим циклом РПС різного рівня складності, а саме: запуск, зупинка та відновлення сервісів, додавання нових сервісів, визначення їх технічного стану, документування рішень, тощо.

Система Docker Compose має наступні базові кроки виконання:

- визначення загальної архітектури РПС та базових елементів;

- документування результатів і створення відповідного файлу у якому записаний перелік всіх елементів автоматизації;

- безпосередня робота і запуск файлу автоматизації Docker compose;

- контроль і моніторинг результатів під час роботи РПС.

Приклад простого файлу показаний на рис.1. Він містить простий приклад роботи з базою даних. У контейнері відбувається налаштування бази даних, що приведені далі. У такому вигляді система docker-compose не дає можливості переконатися у її перевагою перед технологією Dockerfile. Проте цей приклад, наведений з коментарями, надає можливість чітко визначити основу і принципи побудови docker-compose.

```
version: '3.8'
services: #Визначаємо потрібні сервіси для роботи РПС
  db: #Сервіс для роботи з базою даних
    image: postgres:14.1-alpine
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    volumes:
      - db:/var/lib/postgresql/data # Надає шлях до файлу,
                                    # що призначений для автоматизації команд
  web:
    build:
      context: labsite
    depends_on:
      - db
    ports:
      - '8000:8000'
volumes:
  db:
    driver: local
```

Рисунок 1 – Приклад простої роботи з базою даних.

Більш цікавий приклад наведений далі на рис.2. На даному прикладі наведено формування відносно складної РПС, що містить базу даних і підключений до неї фреймворк Django. Фреймворк містить типове початкове

рішення з веб-застосунком polls і helloworld. Крім того у РПС включена база даних postgres.

```yaml
version: '3.8'
services:
  db:
    image: postgres:14.1-alpine
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    volumes:
      - db:/var/lib/postgresql/data
  polls:
    build:
      context: labsite
    command: bash -c "python manage.py ..."
    depends_on:
      - db
    ports:
      - '8000:8000'
  helloworld:
    build:
      context: mysite
    command: bash -c "python manage.py ..."
    ports:
      - '8001:8001'
volumes:
  db:
    driver: local
```

Рисунок 2 – Складна розподілена програмна система

Зважаючи на рішення, що приведені на рис.1 і рис.2 можна визначитися з типовими вихідними даними для реалізації лабораторної роботи.

Базовий функціонал Docker compose передбачає можливість збереження даних томів під час створення контейнерів. Тобто будуть збережені всі томи, що використовуються створеними сервісами і у такий спосіб відбувається захист від втрат. Є можливість створення копії томів зі старого контейнера в новий створений контейнер.

Наступною важливою перевагою Docker compose є те, що він кешує конфігурацію яка є базою для створення контейнера. Ця система дозволяє будувати кілька ізольованих середовищ на одному хості.

При цьому використовується назва проекту, яка дозволяє реалізувати ізоляцію певних контейнерів відповідно до завдання і архітектури. Можна використовувати технологію ізоляції у наступних контекстах, а саме:

- унеможливити перешкоджанню різних проектів, що можуть використовувати однакові служби на загальному хості або хості розробника;

- створювати кілька незалежних копій одного середовища виконання, наприклад, потрібно запустити стабільну копію для кожної гілки проекту та зробити аналогічну для налагодження і вдосконалення;

- зробити неможливим процес, коли збірки заважатимуть роботі одна одної на сервері, для цього встановлюється унікальний номер збірки.

Система Файл Docker compose надає для розробників, тестувальників і системних інженерів наступні можливості: надає багатий функціонал і можливості для розробників, а саме: налаштовувати всі залежності служб РПС (кеші, черги, бази даних, API веб-служб тощо) за допомогою однієї команди (docker compose up). У цьому варіанті можна створити та запустити один або кілька контейнерів для кожної залежності.

Для тестувальників РПС забезпечується зручний спосіб створення та видалення ізольованих середовищ автоматизованого тестування потрібного набору тестів за допомогою відповідних команд.

Хід і алгоритм встановлення системи Docker compose залежить від операційної системи. Для операційної системи родини Windows Docker Compose може бути встановлений за допомогою програмного застосунку Docker Desktop.

Для операційних систем родини Linux передбачено встановлення відповідно до технологій. Наприклад інсталяція Compose plugin для Ubuntu передбачає виконання кроків, що надані далі. На першому етапі треба отримати нову стабільну версію Docker Compose, що можна зробити шляхом завантаження з офіційного репозиторію Github – https://github.com/docker/compose. Далі потрібно зробити інструмент доступним глобально, шляхом виконання команди [5, 7-9]:

$ sudo curl -L

На наступному етапі потрібно надати відповідні дозволи для виконання команди у середовище операційної системи. Це відбувається шляхом зміни відповідних прав доступу, а саме:

$ sudo chmod +x /usr/local/bin/docker-compose

Для перевірки працездатності системи виконується наступна команда:

$ docker-compose --version.

Відображення поточної версії програми свідчить про її правильну інсталяцію і готовність до роботи, інші повідомлення свідчать про потребу повторного встановлення або налаштування.

Опишемо приклад створення і простого прикладу файлу docker-compose.yml, що взятий з офіційного джерела [5, 7-9]. У цьому файлі відбувається автоматизація створення контейнера з веб-сервером, що побудований на базі образу Nginx із Docker Hub, загальнодоступного реєстру Docker. У якості прикладу цей веб-серевер буде хостити один статичний файл.

На першому етапі потрібно створити новий робочий каталог, у обраному місці операційної системи, перейти до нього. У цьому каталозі потрібно створити каталог app і розмістити у ньому статичний файл index.html. Все це робиться шляхом виконання наступної послідовності команд:

$ mkdir ~/compose-demo

$ cd ~/compose-demo

$ mkdir app

$ nano app/index.html

Далі у файлі index.html реалізувати контент, що буде відображатися. На цьому завершується підготовчий етап створення демо-інфраструктури. На наступному етапі створюється безпосередньо елемент системи Docker Compose, а саме файл docker-compose.yml. Як варіант можна використати наступну команду:

$ nano docker-compose.yml

Після відкриття текстового редактора до файлу потрібно додати контент, що приведений далі і взятий з офіційного джерела [5, 7-9].

docker-compose.yml

version: '3.7'

services:

web:

image: nginx:alpine

ports:

- "8000:80"

volumes:

- ./app:/usr/share/nginx/html

Розглянемо складові елементи файлу docker-compose.yml. Він включає версії конфігурації (version).

Далі йде блок сервісів (services), якій містить відповідні налаштування, що формують складові елементи контейнерів, або контейнеру, який представлений у даному прикладі. Фактично це є єдиний контейнер з web-сервером на основі образу nginx:alpine.

Крім того передбачено у блоці services внутрішній блок для переспрямування портів (директива ports). Усі запити на порт 8000 хост-комп'ютера (системи, з якої запускається Docker Compose) будуть переспрямовані на веб-контейнер на порту 80 Nginx.

Директива volumes створить спільний том між головною машиною та контейнером. Це надасть контейнеру спільний доступ до каталогу програми. Зміст тому буде розміщено за адресою /usr/share/nginx/html усередині контейнера.

Звичайно, що складні РПС містять набагато більше блоків у файлах docker-compose.yml. Приклад можна побачити на рис.1, 2.

Після створення файлу docker-compose.yml його можна запустити у середовище Docker Compose. Наступна команда активує записи у docker-compose.yml та запустить контейнерне середовище у фоновому режимі:

$ docker-compose up -d

При необхідності система Docker Compose шукає образи або на локальній системі або його з Docker Hub.

Для тестування РПС, що активована після роботи docker-compose.yml, потрібно запустити команду "docker-compose ps". Ця команда виводить всю інформацію відповідно РПС, що активована.

Для доступу до демонстраційного додатку, що створений у вищеописаному прикладі потрібно набрати у браузері localhost:8000. Відповіддю на це буде відображення сторінки, що записана у файлі index.html.

Контрольні запитання до лабораторної роботи № 4

1. Поясніть у чому відмінність технології Docker і Docker compose?
2. Які етапи створення складної розподіленої програмної системи з використанням технології Docker compose?
3. Розкрийте основу структури файлу docker-compose.yml.
4. Інструкції Docker compose.
5. Який зв'язок файлу docker-compose.yml і dockerfile?
6. Розкрийте і опішить структуру складного розподіленого проекту, що побудований за технологією Docker compose?
7. Які команди подаються для розгортання складного розподіленого проекту та як забезпечити правильний порядок розгортання його компонентів з використанням Docker compose?
8. Наведіть порядок створення складного розподіленого проекту з використанням Docker compose?

СПИСОК ЛІТЕРАТУРИ

Базова література:

1. Проектування інформаційних систем: Загальні питання теорії проектування ІС (конспект лекцій) [Електронний ресурс]: навч. посіб. для студ. спеціальності 122 «Комп'ютерні науки» / КПІ ім. Ігоря Сікорського; уклад.: О. С. Коваленко, Л. М. Добровська. – Електронні текстові дані (1 файл: 2,02 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2020. – 192с. https://ela.kpi.ua/bitstream/123456789/33651/1/PIS_KL.pdf
2. Інфраструктура програмного забезпечення WEB-застосувань. Лабораторний практикум [Електронний ресурс]: навч. посіб. для студ. спеціальностей 121 «Інженерія програмного забезпечення», 126 «Інформаційні системи та технології» / КПІ ім. Ігоря Сікорського; автор.: М.М. Букасов, Д.О. Галушко, П.Ю. Катін, Я.В. Хіцко. – Електронні текстові дані (1 файл: 1,1 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2023. – 53 с. https://ela.kpi.ua/bitstream/123456789/53028/1/IPZ_WEB-zastosuvan_LP.pdf

Додаткова література:

1. ALAN DENNIS, BARBARA HALEY WIXOM, ROBERTA M. ROTH. System Analisis amd desigm.Fifth Edition. John Wiley & Sons, Inc.2012. 563 c.

2. Django документація [Електронний ресурс] – https://docs.djangoproject.com/en/3.2/.
3. Adam Freeman. Essential Docker for ASP.NET Core MVC.ISBN-13 (pbk): 978-1-4842-2777-0 ISBN-13 (electronic): 978-1-4842-2778-7.2017p.
4. https://docs.djangoproject.com/en/4.1/intro/tutorial01/
5. https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04
6. Rebeka Mukherjee. Python Scripting for System Administration. Department of Computer Science and Engineering Netaji Subhash Engineering College, Kolkata.
7. https://docs.docker.com/samples/django/
8. https://docs.docker.com/compose/reference/
9. Adam Freeman. Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages.2020.
10. Docker документація [Електронний ресурс] – https://docs.docker.com/get-started/.
11. Matthes E. Python Crash Course (2nd Edition) : A Hands-On, Project-Based Introduction to Programming / Eric Matthes. – San Francisco, United States: No Starch. Press, US, 9. – 544 c. – (2nd Edition).
12. Thomas D. The Pragmatic Programmer : your journey to mastery, 20th Anniversary Edition / D. Thomas, A. Hunt. – Boston, United States: Pearson Education. (US), 2020. – 352 c.
13. Learn Python the Hard Way : A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code – New Jersey, United States: Pearson Education. (US), 2013. – 320 c.
14. Learning React : Modern Patterns for Developing React Apps – Sebastopol, United States: O'Reilly Media, Inc, USA, 2020. – 300 c.
15. Docker : Complete Guide To Docker For Beginners And Intermediates, 2020. – 140 c.
16. Docker: Up & Running : Shipping Reliable Containers in Production – Sebastopol, United States: O'Reilly Media, Inc, USA, 2018. – 347 c.
17. Docker homepage - http://www.docker.com/
18. Docker Hub - https://hub.docker.com
19. Docker blog - http://blog.docker.com/
20. Docker documentation - http://docs.docker.com/
21. Docker Getting Started Guide - http://www.docker.com/gettingstarted/
22. Docker code on GitHub - https://github.com/docker/docker
23. Docker on Twitter - http://twitter.com/docker
24. Get Docker help on Stack Overflow - http://stackoverflow.com/search?q=docker
25. Valeria Cardellini. Matteo Nardelli. Container-based virtualization: Docker. Università degli Studi di Roma "Tor Vergata" Dipartimento di Ingegneria Civile e Ingegneria Informatica Corso di Sistemi Distribuiti e Cloud Computing A.A. 2017/18.

26. Adam Freeman. Pro Angular 6 .ISBN-13 (pbk): 978-1-4842-3648-2 ISBN-13 (electronic): 978-1-4842-3649-9/2018 p.