Міністерство освіти і науки України
Національний технічний університет України „КПІ”
Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки
інформації та управління

ЗВІТ

з лабораторної роботи № 5
дисципліни
“ТЕХНОЛОГІЇ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ В УМОВАХ ВЕЛИКИХ ДАНИХ”
на тему:

„Реалізація паралельних обчислень з використанням хмарних технологій”

Виконали студенти
- *ІП-01мн Семченко Андрій*
- *ІП-01мн Кошовець Євген*
- *ІТ-01мн Васюк Владислав*
- *ІТ-01мн Минзар Богдан*

(№ групи, прізвище, ім’я, по батькові)

Прийняв *доц. Жереб К. А.*
(прізвище, ім’я, по батькові)

Київ 2021

ЗМІСТ

1 ПОСТАНОВКА ЗАДАЧІ

Необхідно реалізувати вирішення обраної задачі з використанням хмарних технологій. Зокрема, варто розбити задачу на окремі частини (мікросервіси), які виконуються в незалежних контейнерах з можливістю запуску кількох екземплярів мікросервіса. Можна додатково порівняти реалізацію на основі монолітної архітектури та мікросервісної архітектури. Бажано розгорнути реалізовану програму на якомусь public cloud (AWS, Google Cloud, Microsoft Azure, Heroku, …) – але якщо такої можливості немає, варто хоча б запустити локально з різною кількістю екземплярів. Результатом виконання даної лабораторної роботи є працююча програма, а також звіт про використані технології та можливості, з результатами вимірів.

В якості задачі було обрано проблему пошуку вразливостей, що є у публічних проектах. В якості сховища публічних проектів програмного забезпечення було обрано платформу Github.

Ідея полягає у тому, щоб шляхом аналізу вмісту публічного репозиторію знайти потенційні вразливості, що дозволяють втрутитись у роботу програмного забезпечення. Причому вразливості не тільки у самій реалізації програмного забезпечення, але і вразливості, спричинені недбалим обігом sensitive data, наприклад:

- Зберігання ключів доступу у файлах, що відстежуються VCS
- Зберігання ключів доступу прямо у тексті програмного забезпечення
- Зберігання бекапів у файлах, що відстежуються VCS

Зберігання sensitive data у файлах, що відстежуються системою контролю версій призводить до того, що будь хто може завантажити ці дані і використати для втручання у роботу програмного забезпечення, викрадення даних користувачів, тощо.

2 ВИКОРИСТАНІ БІБЛІОТЕКИ, ФРЕЙМВОРКИ

Додаток було розроблено мовою програмування Java. Був використаний фреймворк Spring[1].

Для роботи з Github було використано бібліотеку JGit [2].

Для балансування запитів поміж воркерами було використано nginx [3].

Робота з потоками була організована стандартними засобами мови програмування Java:

- Callable interface
- Thread Executor Services
- CompleatableFuture

Для логування роботи програми була використана бібліотека Log4j2. Робота з JSON організована засобами бібліотеки Jackson Json.

Для спрощення читабельності коду застосовано бібліотеку Lombok, що автоматично генерує boilerplate код.

3 ОПИС РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Загальна концепція

Програмне забезпечення завантажує вміст публічного Github-репозиторію у тимчасову директорію. Далі, кожен файл даної директорії аналізується на предмет наявності в ньому певних паттернів, що можуть свідчити про вразливість. Сам паттерн описується за допомогою регулярного виразу.

Було прийняте рішення розподілити на такі мікросервіси, як master та worker.

Master – існує один,приймає запити ззовні та розпаралеллює задачі поміж воркерами.

Worker – може бути n, представляє собою трішки перероблену першу лабораторну, отримує на вхід до 4х репозиторіїв, оброблює їх та повертає результати до мастера.

Отже, паралельна обробка використовується як при розподілі запитів поміж воркерами, так і всередині самих воркерів.

3.2 Деталі роботи програмного забезпечення

Загальний принцип роботи:

1. Програма на даний момент налаштована на запуск із використанням docker-compose. При запуску можна вказати кількість воркерів(за замовченням один).
2. Користувач відпраляє запит на Master на ендпоінт “<server adress>:8088/master” (порт можно змінити в налаштуванні) додавши у тіло запиту json із списком репозиторіїв, які він хоче провірити на вразливості.
3. Мастер ділить репозиторії у групи до 4х штук у кожній та ассинхронно, використовуючи CompleatableFuture, кидає запити на nginx.
4. Ngix розподіляє по завантаженості запити на різні воркери.
5. На воркери приходять запити із аналогічним головному запиту json-ом із списком репозиторіїв. Після завантаження та аналізу кожного воркера повертає у відповіть json із списком репортів до кожного із репозиторіїв.
6. Мастер чекає поки всі воркери оброблять свою порцію. Після того як всі відповіді повернулись мастер збирає іх всі разом та повертає результат у відповідь користувачу.

3.3 Можливі покращення

Наша команда вирішила, що дана реалізація є оптимальною для здачі лабораторної роботи. Проте звісно є ряд аспектів, які можно покращити. Нижче буде вказано деякі з них:

1. Запит користувача не повинен чекати довгий час виконання роботи. Як варіант можна повертати користувачу деякий ідентифікатор використовуючи який він міг би перевіряти статус обробки та при отримані позитивного статусу дістати результат.
2. Дотати можливість скейлити не тільки кількість воркерів, але й кількість майстрів при необхідності.
3. Додати месседж брокер для спілкування мастера та воркерів.

4 РОЗГОРТАННЯ ТА ЗАПУСК ЗАСТОСУНКУ

4.1 Розгортання застосунку

В поточній реалізації додаток розгорнуто локально, на одному фізичному вузлі за допомогою docker-compose.

Щоб мікросервісний додаток міг працювати на декількох фізичних вузлах (серверах), треба модифікувати його.

Так, потрібно змінити конфігурацію, щоб взаємодія була не через внутрішню мережу Docker (локальна) а через Інтернет, тобто використати публічні IP адреси.

Також потрібно переглянути балансування навантаження — наприклад розгорнути Nginx на окремому вузлі, чи скористатись рішеннями від cloud провайдерів (наприклад DigitalOcean чи Amazon AWS надають load balancer як сервіс), або ж зробити балансування запитів за допомогою DNS, тобто вказати декілька IP-адрес під одним доменним ім'ям, щоб отримати round robin балансування при HTTP запиті, відправленому за доменним іменем.

На даний момент кожний із мікросерів має свій докер файл. Їх можна знайти у репозиторії у відповідних проектах Мастер та Воркер. Також є docker-compose налаштування та налаштування nginx.

При запуску необхідно передати наступний аргумент:

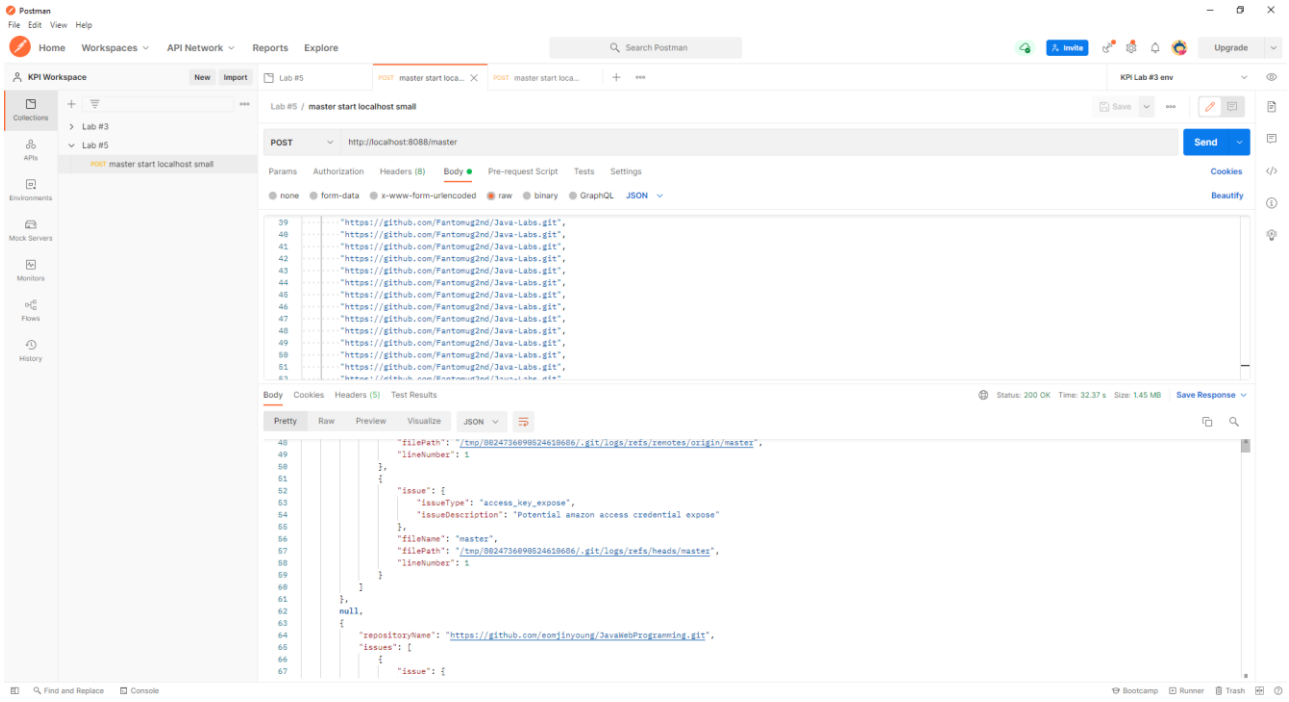
1. Кількість воркерів

Приклад команди для запуску :

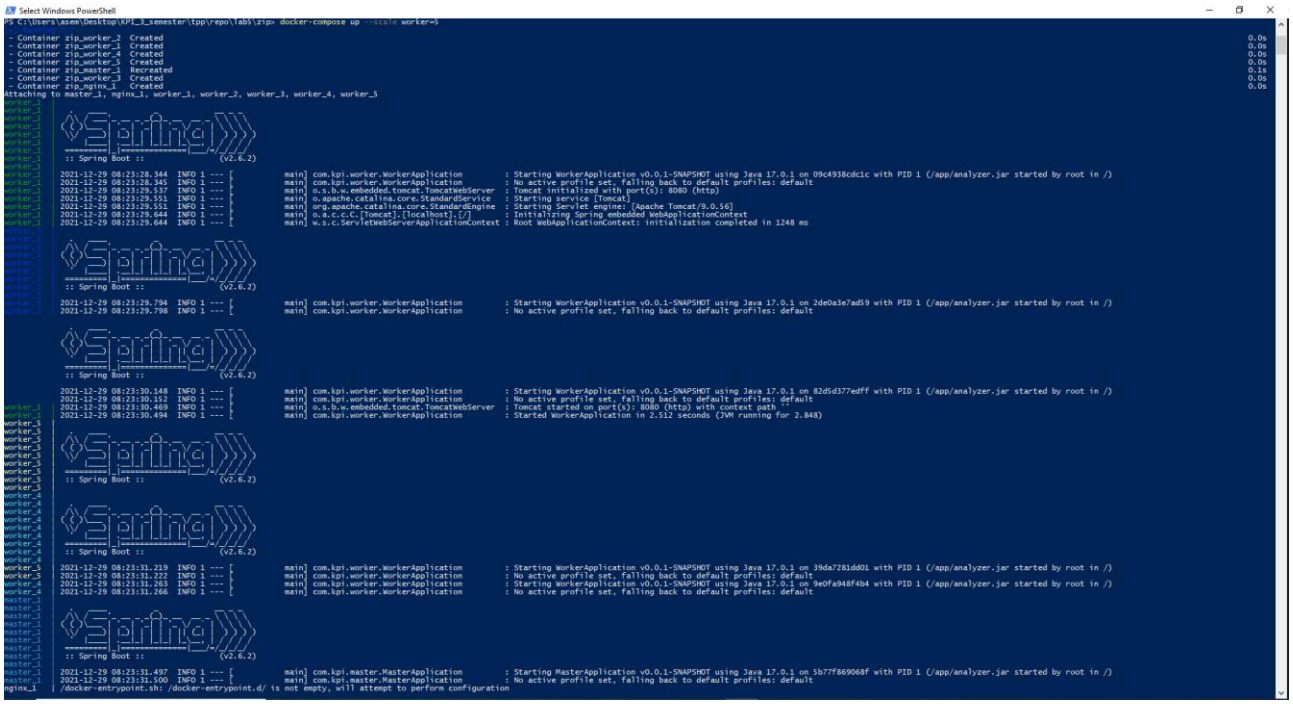
```
docker-compose up -scale worker=5
```

4.2 Запуск застосунку

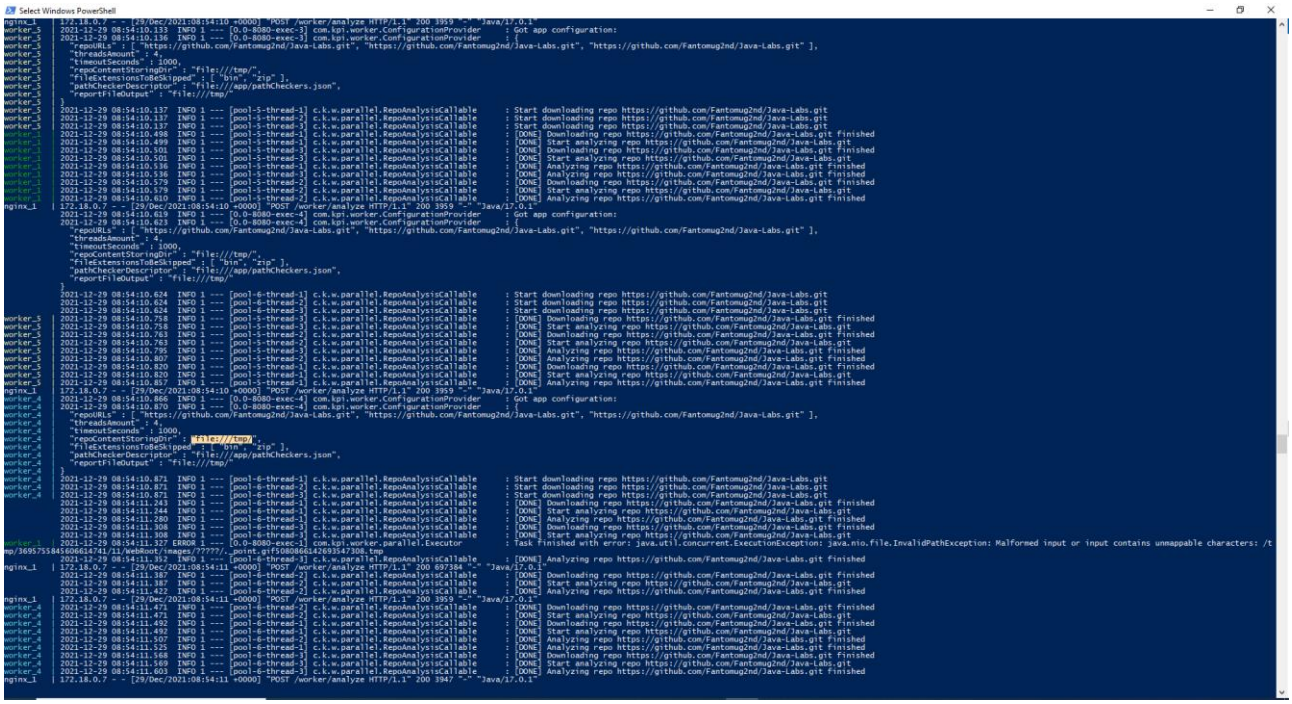
Приклад успішного запиту у Postman:



Запуск мастера та 5-ти воркерів із використанням docker-compose:

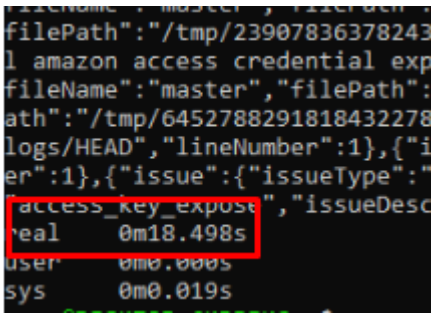


Обробка репозиторіїв різними воркерами:



4.3 Порівняння часу

Приклад заміру часу виконання даного застосунку:



Так як воркер написаний на основі першої лабораторної, то буде доцільно порівняти із її найкращим часом виконання., який був **22.39с** .

Заміри відбулися на тому самому комп’ютері.

Кількість воркерів	1	2	3	4	5	6	7
Час, с	56.32	32.41	26.2	22.13	18.5	19.33	19.24

Бачимо, що в нашому випадку оптимальна кількість воркерів виявилась 5. При цьому важко стверджувати, що це оптимальна кількість.
З однієї сторони при 5 воркерах застосунок працює значно швидше, ніж при меншій кількості. При цьому бачимо, що при збільшенні час в +- тому самому діапазоні. Тому треба проводити експерименти із різною кількістю та наповненістю репозиторіїв.

В будь-якому разі бачимо пришвидшення на 17 % у порівнянні із ЛР1.

5 ВИСНОВОК

В рамках даної лабораторної роботи було розроблено програмне забезпечення, що проводить аналіз Github-репозиторіїв на наявність в них типових вразливостей, яке було разбите на мікросервіси.

В рамках реалізованого алгоритму вдалося отримати пришвидшення на 17 % у порівнянні із багатопоточною монолітною реалізацією.

6 ПОСИЛАННЯ

[1] https://spring.io/projects/spring-framework
[2] https://www.eclipse.org/jgit/
[3] www.nginx.org